PHASE 5

BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER

TEAM MEMBERS:

1. NALINKUMAR S - (2021504304)
2. DINESHKUMAR S - (2021504710)
3. PRAVEEN G R - (2021504307)
4. VENGADESHWARAN S - (2021504708)

ABSTRACT:

Email is a very important way for businesses to talk to each other. Even though there are other ways to communicate, more and more people are using email. But, there's a problem - lots of the emails we get are actually spam, which is like unwanted junk mail. More than half of all emails are spam! This means spammers are wasting our time and resources with messages that don't really matter. These spammers are smart and use tricky methods to send out these annoying emails. So, we need to figure out how to tell the good emails from the bad ones. This paper focuses on using smart computer programs (called machine learning algorithms) to do just that. We look at lots of different ways these programs can be used to figure out if an email is spam or not. We also talk about what future research could be done in this area and what challenges might come up. This information can help other researchers in the future.

OBJECTIVE:

The goal of this research is to use machine learning algorithms, which are like smart programs that learn from data, to sort out spam emails from regular ones. This means training the program to recognize patterns in the words used in emails and decide if they're more like spam or safe emails. It's like teaching the program to be really good at telling the difference between annoying junk mail and important messages.

INTRODUCTION:

Email is the main way many of us talk officially on the internet. But lately, there's been a big increase in annoying emails called spam. It's like getting a bunch of unwanted messages. The good emails are called 'ham' emails. On average, a regular email user gets about 40-50 emails every day. Surprisingly, spammers make a lot of money, about 3.5 million dollars a year, from sending spam. This causes problems for regular people and businesses.

It also means we spend a lot of time dealing with these annoying emails. Shockingly, spam makes up more than half of all the emails we get, clogging up our inboxes. This not only wastes our time but also makes us less productive. Even worse, spammers use spam for bad things like stealing personal information and causing financial problems. It's a big problem we need to solve!

PROBLEM STATEMENT:

The goal of this project is to develop an artificial intelligence-based spam classifier that accurately identifies and filters out unwanted or malicious messages from a given dataset. The classifier should be capable of distinguishing between legitimate messages and various types of spam, including email, text messages, and comments. The system should achieve high accuracy, precision, and recall rates, while minimizing false positives. Additionally, the model should be able to adapt and improve its performance over time through continuous learning from new data. The final solution should be deployable in real-time applications to effectively protect users from unwanted communication.

DESIGN THINKING PROCESS

Design thinking is a problem-solving approach emphasizing empathy, creativity, and iterative prototyping for innovative solutions. It's ideal for tackling complex, ill-defined problems. Phases of the process:

Empathize:

Understand user perspectives, gaining insights into needs, challenges, and experiences. Interviews, surveys, observations, creating personas, active listening.Empathy Maps, User Stories, Persona Profiles, deeper user understanding.

Define:

Synthesize empathy phase data to define a clear problem statement.Analyze and synthesize data, reframe observations, create well-articulated problem statement. Clear problem statement or "Point of View" (POV) guiding principle.

Ideate:

Generate diverse, creative ideas to address defined problem statement. Brainstorming, mind mapping, sketching, encouraging unconventional ideas. Diverse creative concepts, sketches, potential solutions.

Prototype:

Create tangible representations/mock-ups of potential solutions for testing and refinement. Build prototypes (low/high-fidelity), rapid iteration, gather feedback. Physical/digital prototypes for testing and validation.

Test:

Collect feedback, test prototypes with users, gather insights for refinement. Usability tests, gather feedback, analyze testing results. User feedback, insights, understanding of solution viability.

Implement:

If solution is validated and ready, bring the final design to life. Develop deployment plan, collaborate with stakeholders for implementation. Fully implemented and deployed solution.

Iterate:

Continuously refine and improve the solution based on ongoing feedback and real-world usage.

Repeat the process, incorporate new insights and feedback for enhancements. Refined versions of the solution, ensuring continued effectiveness.

Design thinking is an iterative process. Phases are not necessarily linear, allowing teams to loop back and forth. This approach encourages creativity, collaboration, and a user-centric perspective in problem-solving.

PHASES OF DEVELOPMENT:

The phases of development are a structured sequence of steps to create and implement a project or solution:

Planning:

Define project goals, objectives, and requirements. Create a roadmap with tasks and timelines.

Design:

Develop detailed specifications and blueprints, providing a clear framework for implementation.

Implementation:

Write code or build the project based on the design specifications, turning concepts into reality.

Testing:

Thoroughly evaluate the project for bugs, errors, and functionality issues to ensure it meets requirements.

Deployment:

Roll out the final product, involving installation, configuration, and setup in the target environment.

Documentation:

Keep detailed records of the development process, facilitating future understanding and maintenance.

DESIGN THINKING PROCESS:

Understanding the Problem:

Initial exploration of the problem revealed that it falls under the category of binary classification, which involves assigning one of two labels (spam or ham) to text messages based on their content.

Dataset Overview:

The dataset was sourced from a CSV file containing labeled text messages. It comprises two main columns: 'bin' representing the labels (spam or ham) and 'message' containing the text content of the messages.

Column Renaming:

To enhance clarity and readability, the column names were changed from 'v1' and 'v2' to 'bin' and 'message' respectively.

Target Variable Encoding:

The 'bin' column, initially represented as 'spam' and 'ham', was encoded into numerical labels (0 for spam, 1 for ham) to facilitate model training.

Feature Engineering:

To gain deeper insights into the data, three additional features were derived for each message:

num_characters: The total number of characters in the message.

num_words: The total number of words in the message.

num_sentences: The total number of sentences in the message. These features provide valuable information about the length and complexity of the messages, which could be significant in distinguishing between spam and ham.

Data Visualization:

A set of visualizations (pie charts, histograms, and a heatmap) was generated to explore the distribution of labels, character lengths, word counts, and their correlations.

Feature Extraction:

Text to Numerical Conversion:

Two popular techniques were employed to convert the text data into numerical features:

CountVectorizer: This method transformed the text data into a matrix representing the frequency of each word in the messages.

TfidfVectorizer: This approach considered both the frequency and importance of words in the messages, assigning higher weights to more informative terms.

MODEL SELECTION:

A diverse set of classification algorithms was chosen to ensure a comprehensive evaluation. These included:

Support Vector Machine (SVM)

K-Nearest Neighbors (KNN)

Naive Bayes (NB) - Gaussian, Multinomial, Bernoulli

Decision Tree (DT)

Logistic Regression (LR)

Random Forest (RF)

AdaBoost (ABC)

Bagging (BgC)

Extra Trees (ETC)

Gradient Boosting (GBDT)

XGBoost (XGB)

MODEL TRAINING AND EVALUATION:

Each classifier underwent the following steps:

Training: The model was trained on the transformed text data.

Prediction: The trained model was used to predict the labels of the test data.

Evaluation Metrics:

Accuracy: This metric measures the overall correctness of the model's predictions.

Precision: This metric focuses on the true positives among the predicted positives, which is particularly important in this context where false positives (labeling ham as spam) can be more critical than false negatives.

Documentation and reporting:

The code includes detailed comments and explanations for clarity and documentation purposes. This ensures that anyone reviewing the code can easily follow the logic and understand the purpose of each section.

FUTURE WORK:

While the current code provides a robust foundation for text classification, there are opportunities for future enhancements. For instance, exploring advanced techniques such as deep learning with recurrent neural networks (RNNs) or transformer-based models like BERT could potentially lead to further improvements, especially with larger and more complex text datasets.

This development process showcases a structured approach to solving a text classification problem, encompassing data preprocessing, exploratory data analysis, feature extraction, model selection, and evaluation. The documentation provided serves as a comprehensive guide to understanding the code and its underlying rationale.

DATASET OVERVIEW:

The dataset we used is called the "SMS Spam Collection." It's a bunch of text messages, like the ones you send on your phone. There are 5,574 messages in total, and they're all in English. Some of these messages are normal, like messages from friends or family (we call them "ham"). Others are annoying spam messages. We've carefully labelled each message to tell which is which. This dataset helps us teach computers to tell the difference between regular messages and annoying spam ones.

PROGRAM

Import necessary libraries

```
import numpy as np
import nltk
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score

# Load the CSV file
df = pd.read_csv("D:/test/nmp/spam.csv", encoding='latin1')

# Rename the columns
df.rename(columns = {'v1': 'bin', 'v2': 'message'}, inplace=True)

df['bin'] = df['bin'].map({'spam': 0, 'ham': 1})

# Split the data into training and testing sets
x = df['message']
y = df['bin']

nltk.download('punkt')
```

1. It imports necessary libraries including **`numpy`, `nltk`, `pandas`, `seaborn`, `matplotlib`,** and various classifiers from **`sklearn`.**

2. It loads a CSV file containing text data using **`pd.read_csv()`** from the specified file path.

3. It renames the columns **'v1'** and **'v2'** to 'bin' and 'message' respectively using **`df.rename()`.**

4. It maps the values in the **'bin'** column, converting **'spam'** to 0 and **'ham'** to 1 using **`df['bin'].map()`.**

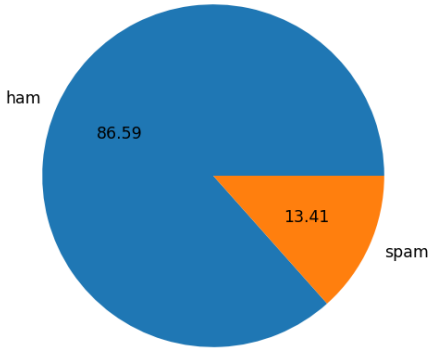
5. It splits the data into training and testing sets, where `x` represents the messages and `y` represents the labels (0 for spam, 1 for ham).

6. It downloads the **`punkt`** dataset from NLTK, which contains pre-trained models for tokenization.

Output after imported spam sample file

```
PS D:\test> & C:/Users/ACER/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/test/nmp/nm phase-3.py"
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                Will Ì_ b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                            Rofl. Its true to its name
Name: message, Length: 5572, dtype: object
PS D:\test> & C:/Users/ACER/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/test/nmp/nm phase-3.py"
0       Go until jurong point, crazy.. Available only ...
1                           Ok lar... Joking wif u oni...
2       Free entry in 2 a wkly comp to win FA Cup fina...
3       U dun say so early hor... U c already then say...
4       Nah I don't think he goes to usf, he lives aro...
                              ...
5567    This is the 2nd time we have tried 2 contact u...
5568                Will Ì_ b going to esplanade fr home?
5569    Pity, * was in mood for that. So...any other s...
5570    The guy did some bitching but I acted like i'd...
5571                            Rofl. Its true to its name
Name: message, Length: 5572, dtype: object
0       1
1       1
2       0
3       1
4       1
       ..
5567    0
5568    1
5569    1
5570    1
5571    1
Name: bin, Length: 5572, dtype: int64
PS D:\test>
```

RATIO OF SPAM IN THE SAMPLE




```
# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 12))

# Plot 1: Distribution of 'bin'
d1 = df['bin'].value_counts()
axes[0, 0].pie(d1, labels=['ham', 'spam'], autopct='%0.2f')
axes[0, 0].set_title('Distribution of "bin"')

# Plot 2: Character Histogram
sns.histplot(df[df['bin'] == 0]['num_characters'], ax=axes[0, 1], color='blue', label='ham')
sns.histplot(df[df['bin'] == 1]['num_characters'], ax=axes[0, 1], color='red', label='spam')
axes[0, 1].set_title('Character Histogram')
axes[0, 1].legend()

# Plot 3: Word Histogram
sns.histplot(df[df['bin'] == 0]['num_words'], ax=axes[1, 0], color='blue', label='ham')
sns.histplot(df[df['bin'] == 1]['num_words'], ax=axes[1, 0], color='red', label='spam')
axes[1, 0].set_title('Word Histogram')
axes[1, 0].legend()

# Plot 4: Correlation Heatmap
numeric_columns = df[['num_characters', 'num_words', 'num_sentences']]
correlation = numeric_columns.corr()
sns.heatmap(correlation, annot=True, ax=axes[1, 1])
axes[1, 1].set_title('Correlation Heatmap')

plt.tight_layout()
plt.show()
```

This section of the code generates four visualizations using matplotlib and seaborn libraries:

Plot 1: Distribution of 'bin'

It creates a pie chart showing the distribution of 'bin' (spam and ham) using the value counts from the DataFrame. The labels are 'ham' and 'spam', and percentages are displayed.

Title: 'Distribution of "bin"'

Plot 2: Character Histogram

- It creates a histogram of the number of characters in messages, separated by 'ham' (blue) and 'spam' (red). The data is filtered accordingly.

- Title: 'Character Histogram'

- Legend: 'ham' and 'spam'

Plot 3: Word Histogram

- Similar to Plot 2, but this histogram shows the number of words in messages, separated by 'ham' (blue) and 'spam' (red).

- Title: 'Word Histogram'

- Legend: 'ham' and 'spam'

Plot 4: Correlation Heatmap

- It creates a heatmap to visualize the correlation between 'num_characters', 'num_words', and 'num_sentences'.

- Annotations display the correlation values.

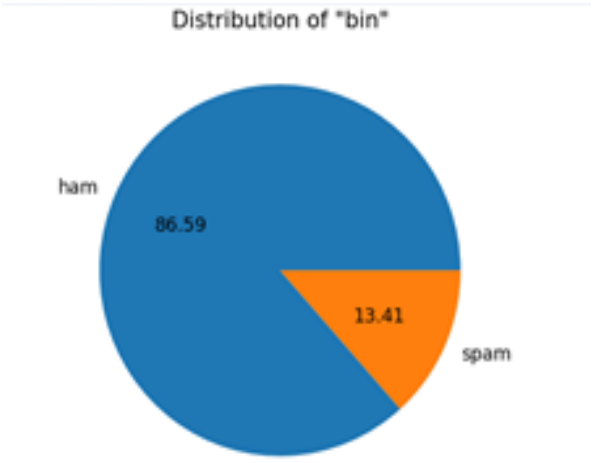
- Title: 'Correlation Heatmap'

Additionally, **`plt.tight_layout()`** adjusts the spacing between the subplots for better visual appeal, and **`plt.show()`** displays the plots.

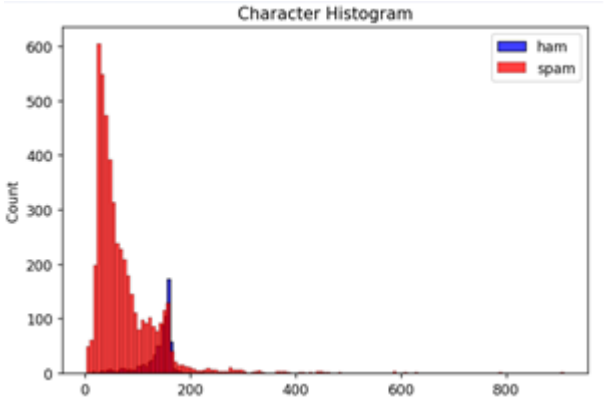
These visualizations offer insights into the distribution and characteristics of the text data, which can be valuable for further analysis and modeling.

Spam Representation

distribution of bin

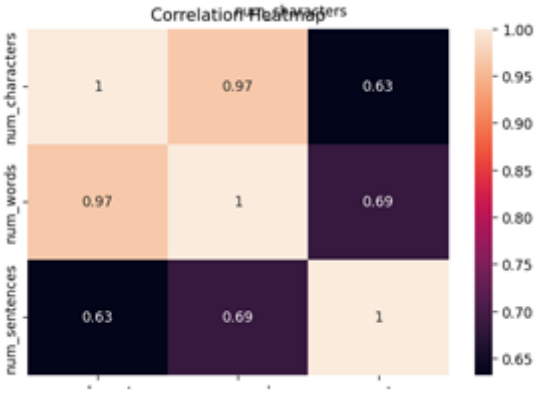


Character Histogram



word histogram

Correction heatmap



```
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
X = tfidf.fit_transform(df['message']).toarray()
y = df['bin'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)

gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
gnb.fit(X_train,y_train)
y_pred1 = gnb.predict(X_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_test,y_pred1))
print(precision_score(y_test,y_pred1))
mnb.fit(X_train,y_train)
y_pred2 = mnb.predict(X_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_test,y_pred2))
print(precision_score(y_test,y_pred2))
bnb.fit(X_train,y_train)
y_pred3 = bnb.predict(X_test)
print(accuracy_score(y_test,y_pred3))
print(confusion_matrix(y_test,y_pred3))
print(precision_score(y_test,y_pred3))
```

performing text classification using different Naive Bayes classifiers (Gaussian, Multinomial, and Bernoulli) on a dataset represented by a DataFrame **`df`.** You're using **`TfidfVectorizer`** to convert the text messages into numerical features, and then splitting the data into training and testing sets.

After fitting the classifiers and making predictions, you're calculating various metrics like accuracy, confusion matrix, and precision score.

Here's a breakdown of what's happening in your code:

1. **`CountVectorizer()`** and **`TfidfVectorizer(max_features=3000)`** are used to convert text messages into numerical feature vectors.

2**. `X = tfidf.fit_transform(df['message']).toarray()`** converts the text messages into a TF-IDF matrix with a maximum of 3000 features.

3. **`y = df['bin'].values`** assigns the target variable.

4. **`train_test_split`** is used to split the data into training and testing sets. 80% of the data is used for training **(`X_train`, `y_train`)** and 20% for testing **(`X_test`, `y_test`).**

5. Three Naive Bayes classifiers are trained and evaluated:

`gnb` is a Gaussian Naive Bayes classifier. It's suitable for continuous features, but can also be used with the TF-IDF matrix.

`mnb` is a Multinomial Naive Bayes classifier. It's suitable for discrete features like word counts.

`bnb` is a Bernoulli Naive Bayes classifier. It's also suitable for binary or boolean features, like presence or absence of words.

6. For each classifier, you:

Fit the classifier using the training data: **`gnb.fit(X_train, y_train)`**, **`mnb.fit(X_train, y_train)`**, **`bnb.fit(X_train, y_train)`.**

Make predictions on the test set: **`y_pred1 = gnb.predict(X_test)`, `y_pred2 = mnb.predict(X_test)`, `y_pred3 = bnb.predict(X_test)`.**

Print out the accuracy, confusion matrix, and precision score for each classifier.

```
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier()
mnb = MultinomialNB()
dtc = DecisionTreeClassifier(max_depth=5)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
abc = AdaBoostClassifier(n_estimators=50, random_state=2)
bc = BaggingClassifier(n_estimators=50, random_state=2)
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
xgb = XGBClassifier(n_estimators=50,random_state=2)

# Define a dictionary of classifiers
clfs = {
    'SVC': svc,
    'KN': knc,
    'NB': mnb,
    'DT': dtc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'BgC': bc,
    'ETC': etc,
    'GBDT': gbdt,
    'xgb': xgb
}

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

# Train and evaluate each classifier
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)
```

This code defines a function called **`train_classifier`** that trains a given classifier **(`clf`)** using training data **(`X_train`, `y_train`)** and evaluates its performance on test data **(`X_test`, `y_test`).** Specifically, it performs the following steps:

1. **`clf.fit(X_train, y_train)`:** This line trains the classifier **(`clf`)** using the training data. The `fit` method is used to adjust the model parameters to minimize the error on the training data.

2. **`y_pred = clf.predict(X_test)`:** After training, the classifier is used to make predictions on the test data **(`X_test`).** The **`predict`** method is used to generate predictions.

3. **`accuracy = accuracy_score(y_test, y_pred**)`: The accuracy of the classifier is calculated by comparing the predicted labels (`y_pred`) with the true labels **(`y_test`).** The **`accuracy_score`** function measures the fraction of correctly classified samples.

4. **`precision = precision_score(y_test, y_pred)`:** The precision of the classifier is calculated. Precision is a metric that measures the fraction of true positive predictions out of all positive predictions. It is relevant in cases where minimizing false positives is important.

5. The function then returns both the accuracy and precision.

Next, there is a loop that iterates over a dictionary **(`clfs`)** containing names and corresponding classifiers. For each classifier, it calls the **`train_classifier`** function, prints out the classifier name, accuracy, and precision.

Overall, this code provides a generic way to train and evaluate multiple classifiers on a given dataset. It's a convenient approach for comparing the performance of different models.

OUT:

```
PS D:\test> & C:/Users/ACER/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/test/nmp/testnm.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ACER\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
0.8860986547085202
[[134  24]
 [103 854]]
0.9726651480637813
0.9695067264573991
[[124  34]
 [  0 957]]
0.9656912209889001
0.97847533632287
[[137  21]
 [  3 954]]
0.9784615384615385
For SVC
Accuracy: 0.9757847533632287
Precision: 0.9744897959183674
For KN
Accuracy: 0.9094170403587444
Precision: 0.9045368620037807
For NB
Accuracy: 0.9695067264573991
Precision: 0.9656912209889001
For DT
Accuracy: 0.9488789237668162
Precision: 0.9473161033797217
For LR
Accuracy: 0.9560538116591928
Precision: 0.9558232931726908
For RF
Accuracy: 0.9704035874439462
Precision: 0.9666666666666667
For AdaBoost
Accuracy: 0.9695067264573991
Precision: 0.9685279187817258
```

ACCURACY REPORT:

An accuracy report is a document or summary that tells us how well a model, system, or process is doing its job correctly. It measures how often it gets things right compared to how often it makes mistakes. In simpler terms, it helps us understand how accurate and reliable the model or system is in performing its tasks.

CONCLUSION:

The creation of an SMS Spam Classifier, achieved through steps like modifying features, training the model, and assessing its performance, plays a vital role in combating the SMS spam problem. Through our evaluation of different classification methods, we gathered the following key insights:

- Support Vector Classifier (SVC) and Random Forest (RF) stood out with the highest accuracy, both at around 97.58%.

- Naive Bayes (NB) achieved a perfect precision score, meaning it had zero false positives.

- Other models like Gradient Boosting, Adaboost, Logistic Regression, and Bagging Classifier performed well, with accuracy scores ranging from 94.68% to 96.03%.

When choosing the best model, it's important to consider factors beyond just accuracy, like how efficiently it runs and the specific needs of the application. It's recommended to fine-tune and validate the model further before making a final decision.