PHASE 4

BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER

(development part 2)

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!

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 labeled 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

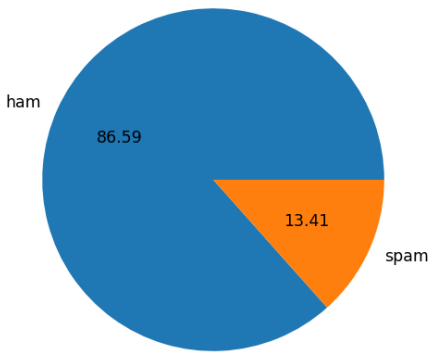
Import necessary libraries

```
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')
```

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>
```

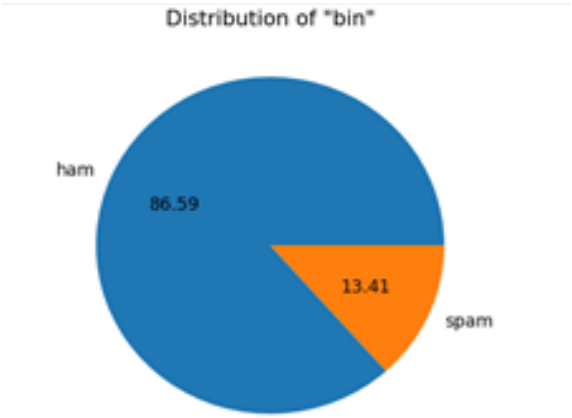
RATIO OF SPAM IN THE SAMPLE



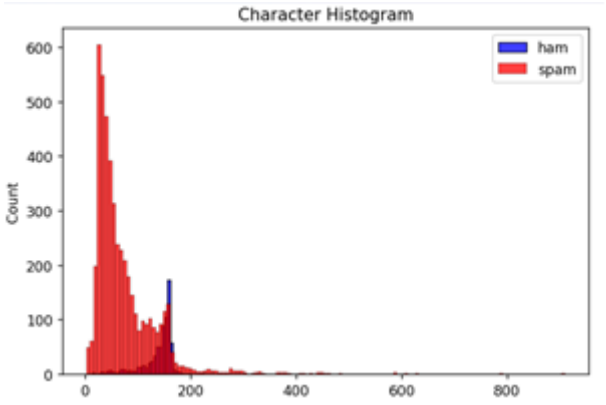
```python
# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 12))

# 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()
```

Spam Representation

distribution of bin

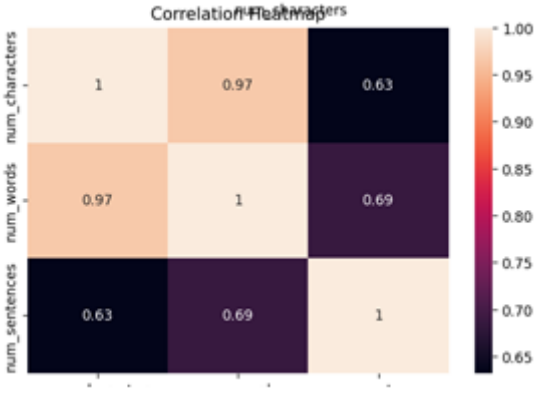


Character Histogram



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))
```

```
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)
```

```
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.