**Building A Smarter**

**AI-Powered Spam Classifier**

TEAM MEMBERS

731321104005 - K.DHANASREE

731321104007 - T.DHIVYA

731321104027 - M.RANGESHKUMAR

731321104028 - M.SARANYA

Phase 5 submission document

**Title:** Smarter AI-Powered Spam Classifier

**Phase 5:** Project Documentation & Submission

**Topic**: In this section we will document the complete

project and prepare it for submission.

**Problem Statement**

* The problem at hand involves differentiating between spam messages, which are unsolicited or unwanted, and legitimate messages that users typically want to receive.
* By developing an effective spam classifier, we can enhance communication experiences and prevent users from being exposed to potentially harmful or unwanted content.

**Design thinking process**

1. Empathize:

* Understand the user's pain points related to spam messages.
* Gather insights from users' experiences with current spam filters.
* Create user personas representing typical users, their needs, and preferences in spam filtering.

2. Define:

* Clearly define the problem statement: "Develop an intelligent spam classifier to protect users from unwanted and potentially harmful messages in digital communication platforms."
* Establish project objectives and success criteria, such as a target accuracy rate and user satisfaction levels.

3. Ideate:

* Engage in brainstorming sessions with the project team to generate innovative solutions for the spam classifier.
* Explore the potential of integrating NLP and ML techniques for spam classification.
* Identify potential data sources, tools, and technologies for implementation.

4. Prototype:

* Create a preliminary design for the spam classifier.
* Define the architecture, data flow, and interaction with users.
* Begin collecting a dataset of labeled spam and non-spam messages for model training.
* Experiment with data preprocessing, feature extraction, and ML model selection.
* Develop a small-scale prototype to test core components.

5. Test:

* Test the prototype with sample data to evaluate its effectiveness in detecting spam messages.
* Gather feedback from testers to identify strengths and weaknesses.
* Conduct user testing to understand how users interact with the prototype.
* Gather feedback on the user interface and overall experience.

6. Implement:

* Begin the actual development of the spam classifier based on the prototype's success.
* Implement data preprocessing steps, including text cleaning, lowercasing, and tokenization.
* Utilize NLP techniques for text representation, such as TF-IDF, word embeddings, or BERT embeddings.
* Extract features to be used in the spam classification model.
* Choose appropriate ML models (e.g., Random Forest, SVM) for classification.
* Train the selected models using the preprocessed and represented data.

7. Evaluate:

* Assess the model's performance using metrics like accuracy, precision, recall, and F1-score.
* Fine-tune the model and system based on the evaluation results.

8. Launch:

* Deploy the spam classifier for real-world use.
* Ensure scalability, reliability, and user-friendliness in the final product.

9. Monitor and Iterate:

* Continuously monitor the system's performance and user feedback.
* Make necessary improvements and updates to adapt to evolving spam patterns.

**Phases of Development**

❖ Data Collection:

• Obtain a diverse and labeled dataset of messages, comprising both spam and legitimate messages.

❖ Data Preprocessing:

• Clean, tokenize, and preprocess the raw text data to prepare it for feature extraction.

• Perform the following preprocessing steps : Lowercasing: Convert all text to lowercase to ensure uniformity.

• Tokenization: Split text into words or tokens.

• Removal of special characters and stop words.

• Lemmatization or stemming: Reducing words to their base or root form.

❖ Feature Extraction:

• Use TF-IDF or other techniques to convert preprocessed text data into numerical features.

❖ Model Selection & Model Training:

• Select and train a machine learning model, such as Multinomial Naive Bayes, using the extracted features.

• Model Selection:

• Start with a simple algorithm like Multinomial Naive Bayes, and evaluate its performance.

• Model Training:

• Train the selected model using the TF-IDF features and the corresponding labels from the preprocessed dataset.

❖ Model Evaluation:

• Evaluate the trained model's performance using metrics like accuracy, precision, recall, and F1-score to measure its effectiveness in distinguishing between spam and legitimate messages.

❖ Deployment:

• Deploy the trained model as a service, accessible through a user interface.

• The deployment can be in the form of a command-line tool or a simple web application where users can input a message and get a prediction on whether it's spam or not.

❖ User Interface:

• Provide a user interface (web-based or command-line) for users to input messages and receive spam classification predictions.

*As we embark on this journey into the realm of machine learning for*

*house price prediction, we will explore the various techniques, data*

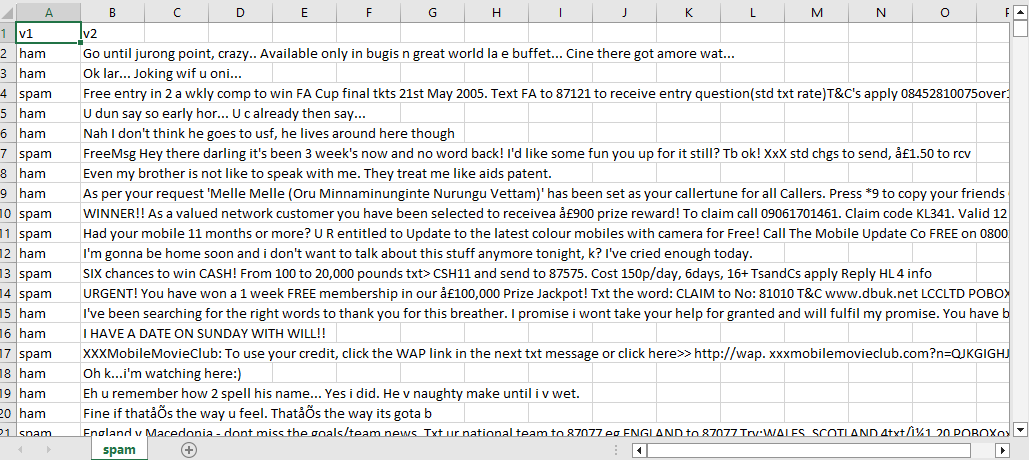
*sources, and challenges involved.*

**Dataset Link:**

**(**

**<https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset )>**

**Given data set:**

****

**5573 Rows x 2Columns**

DESIGN INTO INNOVATION

• Through advanced NLP preprocessing and feature extraction, coupled with ML-based classification, we aspire to discern and flag spam messages intelligently.

• Our innovative approach enables swift adaptation to emerging spam trends, providing an effective shield against unwanted communications. The objective is to elevate user experience by ensuring inbox relevance and minimizing the spam menace in digital communication.

**PYHON PROGRAM:**

Preprocessing data

import numpy as np

import pandas as pd

df = pd.read\_csv('C:\Users\dhanasree\AI-spam\_classifier\AI-spam\_classifier\spam\_ham\_dataset.csv')

df.sample(5)

df.shape

**Data cleaning**

df.info()

df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)

df.sample(5)

df.rename(columns={'v1':'target','v2':'text'},inplace=True)

df.sample(5)

**OUTPUT :**

A screenshot of a chat

Description automatically generated

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['target'] = encoder.fit\_transform(df['target'])

df.head()

**OUTPUT :**

A screenshot of a chat

Description automatically generated

df.isnull().sum()

df.duplicated().sum()

**OUTPUT :**

target 0

text 0

dtype: int64

df = df.drop\_duplicates(keep='first')

df.duplicated().sum()

df.shape

**OUTPUT :**

(5169, 2)

**EDA**

df.head()

**OUTPUT :**

A screenshot of a chat

Description automatically generated

df['target'].value\_counts()

**OUTPUT :**

0 4516

1 653

Name: target, dtype: int64

import matplotlib.pyplot as plt

plt.pie(df['target'].value\_counts(), labels=['ham','spam'],autopct="%0.2f")

plt.show()

**OUTPUT :**

A blue and orange pie chart

Description automatically generated

import nltk

nltk.download('punkt')

**OUTPUT :**

True

df['num\_characters'] = df['text'].apply(len)

df.head()

df['num\_words'] = df['text'].apply(lambda x:len(nltk.word\_tokenize(x)))

df.head()

**OUTPUT :**

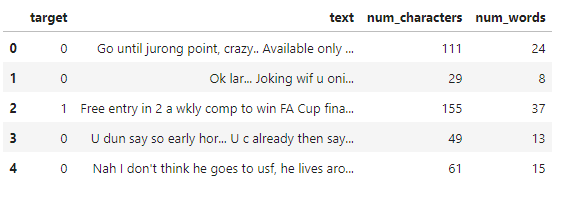
A screenshot of a chat

Description automatically generated

df['num\_sentences'] = df['text'].apply(lambda x:len(nltk.sent\_tokenize(x)))

df.head()

**OUTPUT :**



df[['num\_characters','num\_words','num\_sentences']].describe()

**OUTPUT :**

A screenshot of a chat

Description automatically generated

df[df['target'] == 0][['num\_characters','num\_words','num\_sentences']].describe()

**OUTPUT :**

A table of numbers and text

Description automatically generated

df[df['target'] == 1][['num\_characters','num\_words','num\_sentences']].describe()

**OUTPUT :**

A table with numbers and letters

Description automatically generated

import seaborn as sns

plt.figure(figsize=(12,6))

sns.histplot(df[df['target'] == 0]['num\_characters'])

sns.histplot(df[df['target'] == 1]['num\_characters'],color='red')

A graph with red and blue lines

Description automatically generated

sns.pairplot(df,hue='target') A group of blue and orange dots

Description automatically generated

sns.heatmap(df.corr(),annot=True)

A screenshot of a graph

Description automatically generated

**Data Preprocessing**

def transform\_text(text):

text = text.lower()

text = nltk.word\_tokenize(text)

y = []

for i in text:

if i.isalnum():

y.append(i)

text = y[:]

y.clear()

for i in text:

if i not in stopwords.words('english') and i not in string.punctuation:

y.append(i)

text = y[:]

y.clear()

for i in text:

y.append(ps.stem(i))

return " ".join(y)

transform\_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.")

df['text'][10]

from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

ps.stem('loving')

'love'

df['transformed\_text'] = df['text'].apply(transform\_text)

df.head()

**OUTPUT :**

A screenshot of a computer

Description automatically generated

from wordcloud import WordCloud

wc = WordCloud(width=500,height=500,min\_font\_size=10,background\_color='white')

spam\_wc = wc.generate(df[df['target'] == 1]['transformed\_text'].str.cat(sep=" "))

plt.figure(figsize=(15,6))

plt.imshow(spam\_wc)

A close-up of a text

Description automatically generated

ham\_wc = wc.generate(df[df['target'] == 0]['transformed\_text'].str.cat(sep=" "))

plt.figure(figsize=(15,6))

plt.imshow(ham\_wc)

**OUTPUT :**

<matplotlib.image.AxesImage at 0x16f87f6c280>

df.head()

spam\_corpus = []

for msg in df[df['target'] == 1]['transformed\_text'].tolist():

for word in msg.split():

spam\_corpus.append(word)

len(spam\_corpus)

**OUTPUT :**

9941

from collections import Counter

sns.barplot(pd.DataFrame(Counter(spam\_corpus).most\_common(30))[0],pd.DataFrame(Counter(spam\_corpus).most\_common(30))[1])

plt.xticks(rotation='vertical')

plt.show()

**OUTPUT :**

A graph of a number of colored bars

Description automatically generated with medium confidence

ham\_corpus = []

for msg in df[df['target'] == 0]['transformed\_text'].tolist():

for word in msg.split():

ham\_corpus.append(word)

len(ham\_corpus)

**OUTPUT :**

35303

from collections import Counter

sns.barplot(pd.DataFrame(Counter(ham\_corpus).most\_common(30))[0],pd.DataFrame(Counter(ham\_corpus).most\_common(30))[1])

plt.xticks(rotation='vertical')

plt.show()

**OUTPUT :**

A colorful graph with black text

Description automatically generated

df.head()

**OUTPUT :**

A screenshot of a computer

Description automatically generated

**BUILD LOADING AND PREPROCESSING THE DATASET**

Model building

from sklearn.feature\_extraction.text import CountVectorizer,TfidfVectorizer

cv = CountVectorizer()

tfidf = TfidfVectorizer(max\_features=3000)

X = tfidf.fit\_transform(df[&#39;transformed\_text&#39;]).toarray()

X.shape

**OUTPUT :**

(5169, 3000)

y = df[&#39;target&#39;].values

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=2)

from sklearn.naive\_bayes import GaussianNB,MultinomialNB,BernoulliNB

from sklearn.metrics import accuracy\_score,confusion\_matrix,precision\_score

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))

**OUTPUT :**

A screenshot of a computer code

Description automatically generated

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))

**OUTPUT :**

0.971953578336557

[[896 0]

[ 29 109]]

1.0

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))

**OUTPUT :**

0.9835589941972921

[[895 1]

[ 16 122]]

0.991869918699187

**Model Training**

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

svc = SVC(kernel=&#39;sigmoid&#39;, gamma=1.0)

knc = KNeighborsClassifier()

mnb = MultinomialNB()

dtc = DecisionTreeClassifier(max\_depth=5)

lrc = LogisticRegression(solver=&#39;liblinear&#39;, penalty=&#39;l1&#39;)

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)

clfs = {

&#39;SVC&#39; : svc,

&#39;KN&#39; : knc,

&#39;NB&#39;: mnb,

&#39;DT&#39;: dtc,

&#39;LR&#39;: lrc,

&#39;RF&#39;: rfc,

&#39;AdaBoost&#39;: abc,

&#39;BgC&#39;: bc,

&#39;ETC&#39;: etc,

&#39;GBDT&#39;:gbdt,

&#39;xgb&#39;: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\_classifier(svc,X\_train,y\_train,X\_test,y\_test)

**OUTPUT :**

(0.9729206963249516, 0.9741379310344828)

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(&quot;For &quot;,name)

print(&quot;Accuracy - &quot;,current\_accuracy)

print(&quot;Precision - &quot;,current\_precision)

accuracy\_scores.append(current\_accuracy)

precision\_scores.append(current\_precision)

performance\_df=pd.DataFrame({&#39;Algorithm&#39;:clfs.keys(),&#39;Accuracy&#39;:accuracy\_scores,&#39;Precision&#39;:precision\_scores}).sort\_values(&#39;Precision&#39;,ascending=False)

**OUTPUT :**

A screenshot of a computer

Description automatically generated

For xgb

Accuracy - 0.9700193423597679

Precision - 0.9421487603305785

performance\_df

**OUTPUT :**

A screenshot of a computer screen

Description automatically generated

performance\_df1 = pd.melt(performance\_df, id\_vars = &quot;Algorithm&quot;)

performance\_df1

**OUTPUT :**

A screenshot of a computer

Description automatically generated

A screenshot of a data table

Description automatically generated

sns.catplot(x = &#39;Algorithm&#39;, y=&#39;value&#39;,

hue = &#39;variable&#39;,data=performance\_df1, kind=&#39;bar&#39;,height=5)

plt.ylim(0.5,1.0)

plt.xticks(rotation=&#39;vertical&#39;)

plt.show()

A graph of a graph of data

Description automatically generated with medium confidence

temp\_df=pd.DataFrame({&#39;Algorithm&#39;:clfs.keys(),&#39;Accuracy\_max\_ft\_3000&#39;:accuracy\_scores,&#39;Precision\_max\_ft\_3000&#39;:precision\_scores}).sort\_values(&#39;Precision\_max\_ft\_3000&#39;,ascending=False)

temp\_df=pd.DataFrame({&#39;Algorithm&#39;:clfs.keys(),&#39;Accuracy\_scaling&#39;:accuracy\_scores,&#39;Precision\_scaling&#39;:precision\_scores}).sort\_values(&#39;Precision\_scaling&#39;,ascending=False)

new\_df = performance\_df.merge(temp\_df,on=&#39;Algorithm&#39;)

new\_df\_scaled = new\_df.merge(temp\_df,on=&#39;Algorithm&#39;)

temp\_df=pd.DataFrame({&#39;Algorithm&#39;:clfs.keys(),&#39;Accuracy\_num\_chars&#39;:accuracy\_scores,&#39;Precision\_num\_chars&#39;:precision\_scores}).sort\_values(&#39;Precision\_num\_chars&#39;,ascending=False)

new\_df\_scaled.merge(temp\_df,on=&#39;Algorithm&#39;)

svc = SVC(kernel=&#39;sigmoid&#39;, gamma=1.0,probability=True)

mnb = MultinomialNB()

etc = ExtraTreesClassifier(n\_estimators=50, random\_state=2)

from sklearn.ensemble import VotingClassifier

voting = VotingClassifier(estimators=[(&#39;svm&#39;, svc), (&#39;nb&#39;, mnb), (&#39;et&#39;, etc)],voting=&#39;soft&#39;)

voting.fit(X\_train,y\_train)

**OUTPUT :**

VotingClassifier(estimators=[('svm',

SVC(gamma=1.0, kernel='sigmoid',

probability=True)),

('nb', MultinomialNB()),

('et',

ExtraTreesClassifier(n\_estimators=50,

random\_state=2))],

voting='soft')

y\_pred = voting.predict(X\_test)

print(&quot;Accuracy&quot;,accuracy\_score(y\_test,y\_pred))

print(&quot;Precision&quot;,precision\_score(y\_test,y\_pred))

**OUTPUT :**

Accuracy 0.9816247582205029

Precision 0.9917355371900827

estimators=[(&#39;svm&#39;, svc), (&#39;nb&#39;, mnb), (&#39;et&#39;, etc)]

final\_estimator=RandomForestClassifier()

from sklearn.ensemble import StackingClassifier

clf = StackingClassifier(estimators=estimators, final\_estimator=final\_estimator)

clf.fit(X\_train,y\_train)

y\_pred = clf.predict(X\_test)

print(&quot;Accuracy&quot;,accuracy\_score(y\_test,y\_pred))

print(&quot;Precision&quot;,precision\_score(y\_test,y\_pred))

**OUTPUT :**

Accuracy 0.9787234042553191

Precision 0.9328358208955224

import pickle

pickle.dump(tfidf,open(&#39;vectorizer.pkl&#39;,&#39;wb&#39;))

pickle.dump(mnb,open(&#39;model.pkl&#39;,&#39;wb&#39;))

Evaluation

from flask import Flask, render\_template,request

import pickle

(import numpy as np

app=Flask(\_\_name\_\_)

word\_list=pickle.load(open(&#39;mystrings.pkl&#39;,&#39;rb&#39;))

clf=pickle.load(open(&#39;model.pkl&#39;,&#39;rb&#39;))

@app.route(&#39;/&#39;)

def home():

return render\_template(&#39;index.html&#39;)

@app.route(&#39;/predict&#39;, methods=[&#39;POST&#39;])

def predict():

email=request.form.get(&#39;email&#39;)

input\_mail = []

for i in word\_list:

input\_mail.appendemail.count(i[0]))

x=clf.predict(np.array(input\_mail).reshape(1, 3000))

x=x[0]

return render\_template(&#39;index.html&#39;, label=str(x))

if \_\_name\_\_==&quot;\_\_main\_\_&quot;:

app.run(debug=True)

**OUTPUT :**

Accuracy 0.9787234042553191

Precision 0.9328358208955224

Advantages:

* Enhanced Spam Detection Accuracy: The project aims to leverage NLP and ML techniques, resulting in a spam classifier with significantly improved detection accuracy. This accuracy is crucial in reducing false positives and ensuring that legitimate messages are not mistakenly classified as spam.
* Adaptability: By integrating ML models, the spam classifier can adapt to evolving spam tactics. It can continuously learn from new spam patterns and improve its detection capabilities over time.
* User-Centric Design: The project focuses on user experience and feedback, creating a user-friendly interface and incorporating a feedback loop. This user-centric approach ensures that the spam classifier aligns with users' needs and preferences.
* Customizable and Scalable: The classifier can be customized to suit specific communication platforms, making it versatile and adaptable to different use cases. Additionally, it can scale to handle varying message volumes.
* Reduced False Positives: Advanced NLP and ML techniques can reduce false positives by considering the context and semantics of messages, which is challenging for rule-based systems.
* Protection Against Evolving Threats: The project keeps up with emerging spam tactics, offering a more robust defense against new and sophisticated threats.

Disadvantages:

* Complexity: The integration of NLP and ML techniques introduces complexity to the project. It requires specialized knowledge and expertise in these fields, potentially making development and maintenance more challenging.
* Resource Intensive: Training ML models, especially deep learning models, can be resource-intensive, requiring powerful hardware and significant computational resources.
* Data Quality Dependency: The effectiveness of the spam classifier heavily depends on the quality and quantity of the training data. Insufficient or biased data can lead to suboptimal results.
* Overfitting Risk: Complex ML models are susceptible to overfitting if not properly regularized and validated. This can result in poor generalization to new data.

Benefits:

* Enhanced User Experience: Users benefit from reduced spam intrusion, leading to cleaner and more relevant inboxes, ultimately improving their digital communication experience.
* Time and Resource Savings: Users save time by not having to sift through spam messages. Organizations benefit from reduced resource wastage on spam-related issues.
* Improved Security: A more accurate spam classifier enhances digital communication security, reducing the risk of phishing, malware, and other security threats.
* Adaptation to Changing Threats: The project's adaptability ensures that it can evolve alongside spam tactics, providing continuous protection against new threats.
* Customization and Versatility: Organizations can customize and integrate the spam classifier into their existing communication platforms, tailoring it to their specific needs and requirements.
* Knowledge Gains: The project contributes to a deeper understanding of NLP and ML techniques, which can be applied to other data-driven tasks and projects.