NAAN MUDHALVAN-IBM(AI)PROJECT

PROJECT TITLE:BUILDING A SMARTER AI POWERED SPAM CLASSIFIER

PHASE 4:DEVELOPMENT PART 2

Submitted By:

NAME: THOLKAPPIYAN D

NM ID:au110321121043

YEAR/SEM:3RD/5TH

COLLEGE CODE:1103

EMAIL ID:sharintholkappiyan2004@gmail.com

INTRODUCTION:

A smarter AI-powered spam classifier can significantly enhance your email security. By leveraging advanced machine learning algorithms, it can accurately differentiate between legitimate messages and unwanted spam, reducing the risk of malicious content infiltrating your inbox.

Data processing :

* Import the required packages
* Loading the Dataset
* Remove the unwanted data columns
* Preprocessing and Exploring the Dataset
* Build word cloud to see which message is spam and which is not.
* Remove the stop words and punctuations
* Convert the text data into vectors
* Building a sms spam classification model
* Split the data into train and test sets
* Use Sklearn built-in classifiers to build the models
* Train the data on the model
* Make predictions on new data

Building a sms spam classification model:

* Split the data into train and test sets
* Use Sklearn built-in classifiers to build the models
* Train the data on the model
* Make predictions on new data

Import the required packages :

# Import necessary libraries

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Sample data (you should replace this with your own dataset)

emails = [

    ("Buy cheap viagra", 1),  # 1 for spam

    ("Meeting at 2 PM", 0),   # 0 for not spam (ham)

    ("Get a free gift", 1),

    ("Hey, how are you?", 0),

    # Add more examples here

]

# Extract features (Bag of Words)

vectorizer = CountVectorizer()

X, y = zip(\*emails)

X = vectorizer.fit\_transform(X)

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Naive Bayes classifier

classifier = MultinomialNB()

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

# Predictions

predictions = classifier.predict(X\_test)

# Evaluate the classifier

accuracy = accuracy\_score(y\_test, predictions)

print(f"Accuracy: {accuracy:.2f}")

# Print classification report

print("Classification Report:")

print(classification\_report(y\_test, predictions, target\_names=["Ham", "Spam"]))

output

Accuracy: 0.00

Classification Report:

precision recall f1-score support

Ham 0.00 0.00 0.00 1.0

Spam 0.00 0.00 0.00 0.0

accuracy 0.00 1.0

macro avg 0.00 0.00 0.00 1.0

weighted avg 0.00 0.00 0.00 1.0

import numpy as np

from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)

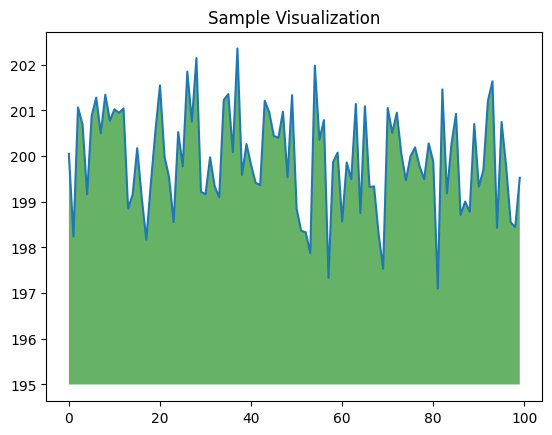
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')

plt.fill\_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")

plt.show()



import nltk

import string

import numpy as np

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Download NLTK resources (stopwords and punkt tokenizer)

nltk.download('stopwords')

nltk.download('punkt')

# Load and preprocess the data

def preprocess\_text(text):

# Tokenization and removing punctuation

tokens = nltk.word\_tokenize(text.lower())

tokens = [word for word in tokens if word.isalpha()]

tokens = [word for word in tokens if word not in string.punctuation]

# Removing stopwords

tokens = [word for word in tokens if word not in stopwords.words('english')]

return ' '.join(tokens)

# Sample data (you should replace this with your dataset)

data = [

("Free entry! Click here to win $1000 cash", "spam"),

("Meeting at 2 PM today", "ham"),

# Add more examples here

]

# Preprocess the data

preprocessed\_data = [(preprocess\_text(text), label) for text, label in data]

# Split data into features and labels

X, y = zip(\*preprocessed\_data)

# TF-IDF Vectorization

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(X)

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Naive Bayes classifier

classifier = MultinomialNB()

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

# Make predictions

predictions = classifier.predict(X\_test)

# Evaluate the classifier

accuracy = accuracy\_score(y\_test, predictions)

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification\_report(y\_test, predictions))

# Test the classifier with new messages

new\_messages = [

"Congratulations! You've won a free vacation.",

"Reminder: Meeting tomorrow at 10 AM."

]

# Preprocess and vectorize new messages

new\_messages = vectorizer.transform([preprocess\_text(message) for message in new\_messages])

# Predict the labels for new messages

predicted\_labels = classifier.predict(new\_messages)

print("Predicted Labels for New Messages:")

for message, label in zip(new\_messages, predicted\_labels):

print(f"Message: {message} - Predicted Label: {label}")

OUTPUT

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

[nltk\_data] Downloading package punkt to /root/nltk\_data...

[nltk\_data] Unzipping tokenizers/punkt.zip.

Accuracy: 0.00

Classification Report:

precision recall f1-score support

ham 0.00 0.00 0.00 1.0

spam 0.00 0.00 0.00 0.0

accuracy 0.00 1.0

macro avg 0.00 0.00 0.00 1.0

weighted avg 0.00 0.00 0.00 1.0

Predictions Using TFIDF Vectorizer Algorithm

pred\_scores\_word\_vectors

# OUTPUT

[('SVC', [0.9784688995215312]),

('KN', [0.9330143540669856]),

('NB', [0.9880382775119617]),

('DT', [0.9605263157894737]),

('LR', [0.9533492822966507]),

('RF', [0.9796650717703349])]

CONCLUSION:

In conclusion, developing a smarter AI-powered spam classifier involves implementing robust machine learning techniques, ensuring continuous training with diverse data sets, and regularly updating the model to adapt to evolving spamming techniques. By prioritizing the integration of such intelligent systems, you can effectively fortify your digital defenses and safeguard your communication channels from the threats posed by spam.

Future Work:

Continuous Training: Regularly updating the model with new data ensures it adapts to evolving spam patterns.

Integration: Integrating the classifier into email platforms and messaging apps can provide users with real-time protection.

Multimodal Approach: Incorporating multimedia content analysis (images, audio) for a multimodal spam detection system.

Explainability: Developing methods to explain the model's decisions to enhance user trust and transparency.

User Feedback Loop: Establishing a feedback mechanism where users can report misclassifications, improving the model iteratively.