TITLE: BUILDING SMARTER AI POWERED SPAM CLASSIFIER

I Project Overview :

In this pre-final year project, we will explore a range of innovative ideas to bolster the capabilities of our AI-powered spam classifier. The project's focus will be on implementing advanced techniques and features that transcend conventional spam classification methods. Below, we outline some of the key innovation ideas

Building a smart AI-powered spam classifier is a common and well-studied problem in the field of natural language processing (NLP). You can use various machine learning algorithms and NLP techniques to tackle this problem. Here's a general outline of how you can approach it:

II Problem Description:

You want to build a spam classifier that can automatically identify and filter out spam messages from a given dataset of messages.

Data:

You will need a labeled dataset of messages, where each message is labeled as either "spam" or "not spam" (ham). This dataset should serve as your training data.

III Steps to Build a Spam Classifier:

Data Preprocessing:

Tokenization: Split messages into individual words or tokens.

Lowercasing: Convert all tokens to lowercase to ensure case-insensitivity.

Remove Stop Words: Eliminate common words like "the," "and," "is," etc.

Text Cleaning: Remove special characters, symbols, and any irrelevant information.

Feature Engineering:

Bag of Words (BoW): Convert each message into a vector of word frequencies (term frequency).

TF-IDF (Term Frequency-Inverse Document Frequency): Use TF-IDF vectorization to give more weight to important words and reduce the influence of common words.

Select a Machine Learning Algorithm:

Common algorithms for text classification include Naive Bayes, Support Vector Machines (SVM), Random Forest, and more recently, deep learning models like LSTM or CNN.

Split Data and Train Model:

Split your dataset into training and testing sets (e.g., 80% for training, 20% for testing).

Train the selected machine learning model on the training data.

IV Evaluate Model:

Use metrics like accuracy, precision, recall, and F1-score to assess the model's performance.

Perform cross-validation to ensure the model's generalization ability.

Tune Hyperparameters:

Fine-tune the model's hyperparameters to optimize its performance.

Test on New Data:

Test the trained model on new, unseen data to verify its performance in a real-world scenario.

Deployment:

Integrate the spam classifier into your application or email system to automatically filter spam messages.

Continuous Monitoring:

Continuously monitor the performance of your spam classifier and retrain it with new data periodically.

V Tools and Libraries:

Python is a commonly used language for implementing NLP models.

Libraries like scikit-learn, NLTK (Natural Language Toolkit), and spaCy are valuable for NLP tasks.

Deep learning libraries like TensorFlow and PyTorch can be useful for more complex models.

Remember that building an effective spam classifier often involves a combination of techniques, including text preprocessing, feature engineering, and selecting an appropriate algorithm. It's also crucial to have a good-quality labeled dataset for training.

VI Program code:

# Import necessary libraries

import numpy as np

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

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

from sklearn.naive\_bayes import MultinomialNB

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

# Sample data (replace with your dataset)

messages = ["Free Viagra now!!!", "Hello, how are you?", "Get rich quick!", "Meeting at 2 pm"]

labels = [1, 0, 1, 0] # 1 for spam, 0 for non-spam

# Data preprocessing

vectorizer = CountVectorizer()

tfidf\_transformer = TfidfTransformer()

X = vectorizer.fit\_transform(messages)

X\_tfidf = tfidf\_transformer.fit\_transform(X)

# Split the data into training and testing sets

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

# Train a Naive Bayes classifier

clf = MultinomialNB()

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

# Make predictions on the test set

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

# Evaluate the model

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

report = classification\_report(y\_test, y\_pred, target\_names=["non-spam", "spam"])

# Print the results

print("Accuracy: {:.2f}%".format(accuracy \* 100))

print("Classification Report:\n", report)

VII output:

Accuracy: 100.00%

Classification Report:

precision recall f1-score support

non-spam 1.00 1.00 1.00 1

spam 1.00 1.00 1.00 1

micro avg 1.00 1.00 1.00 2

macro avg 1.00 1.00 1.00 2

weighted avg 1.00 1.00 1.00 2

ANNexure

Program (code):

-\*- coding: utf-8 -\*-

# coding: utf-8

#Naive Bayes

import os

import io

import numpy

from pandas import DataFrame

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

#Function to read files (emails) from the local directory

def readFiles(path):

for root, dirnames, filenames in os.walk(path):

for filename in filenames:

path = os.path.join(root, filename)

inBody = False

lines = []

f = io.open(path, 'r', encoding='latin1')

for line in f:

if inBody:

lines.append(line)

elif line == '\n':

inBody = True

f.close()

message = '\n'.join(lines)

yield path, message

def dataFrameFromDirectory(path, classification):

rows = []

index = []

for filename, message in readFiles(path):

rows.append({'message': message, 'class': classification})

index.append(filename)

return DataFrame(rows, index=index)

#An empty dataframe with 'message' and 'class' headers

data = DataFrame({'message': [], 'class': []})

#Including the email details with the spam/ham classification in the dataframe

data = data.append(dataFrameFromDirectory('C:/Users/surya/Desktop/DecemberBreak/Data Science with Python & R/DataScience/DataScience-Python3/emails/spam', 'spam'))

data = data.append(dataFrameFromDirectory('C:/Users/surya/Desktop/DecemberBreak/Data Science with Python & R/DataScience/DataScience-Python3/emails/ham', 'ham'))

data = data.append(dataFrameFromDirectory('C:/Users/surya/Desktop/DecemberBreak/emails/spam', 'spam'))

data = data.append(dataFrameFromDirectory('C:/Users/surya/Desktop/DecemberBreak/emails/ham', 'ham'))

#Head and the Tail of 'data'

data.head()

print(data.tail())

vectoriser = CountVectorizer()

count = vectoriser.fit\_transform(data['message'].values)

print(count)

target = data['class'].values

print(target)

classifier = MultinomialNB()

classifier.fit(count, target)

print(classifier)

exampleInput = ["Hey. This is John Cena. You can't see me", "Free Viagra boys!!", "Please reply to get this offer"]

excount = vectoriser.transform(exampleInput)

print(excount)

prediction = classifier.predict(excount)

print(prediction)

output:

(0, 20104) 1 [0->1st sentence; 20104->word id; 1-> no. of times that the word occurs in the sentence]

(0, 15629) 1

(0, 30882) 1

(0, 50553) 1

(0, 36099) 1

(0, 44217) 1

(0, 58467) 1

(0, 51216) 1

(0, 10966) 1

(0, 47038) 1

(0, 46816) 1

(0, 54656) 1

(0, 43219) 2

(0, 16635) 1

(0, 38953) 1

(0, 14434) 1

(0, 16777) 1

(0, 36134) 1

(0, 35030) 1

(0, 46819) 1

(0, 12870) 1

(0, 58727) 1

(0, 22787) 1

(0, 22197) 2