BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER

TEAM MEMBER
311121205047-RAYMON H NATHAN

PHASE 1- DOCUMENT SUBMISSION

Project: Building a Smarter AI-Powered Spam Classifier

OBJECTIVE:

A smarter AI-powered spam classifier is to create a highly accurate, adaptable, and efficient system for identifying and filtering out spam across various forms of digital communication. This system aims to minimize false positives, provide real-time detection, and ensure compliance with privacy regulations while continuously improving its performance to stay ahead of evolving spam tactics.

Phase 1: Problem Definition and Design Thinking

Building a smarter AI-powered spam classifier involves clearly defining the problem, understanding user needs, brainstorming solutions, gathering and preparing data, selecting technology, prototyping, considering ethics, defining success metrics, documenting the process, fostering collaboration, and assessing potential risks. This foundational phase ensures that the project aligns with its goals, user expectations, and ethical considerations while setting the stage for the subsequent development and training of the AI model.

Problem Definition:

This challenge requires creating a sophisticated system capable of accurately distinguishing spam from legitimate messages to enhance user experiences. The primary objective is to significantly improve the precision and recall rates of spam detection, reducing both false positives and false negatives. Addressing this problem is crucial to alleviate user frustration, bolster online security, and boost overall productivity by efficiently managing and reducing the influx of unwanted and potentially harmful content in email and messaging platforms. Key components of this endeavor include defining the classifier's scope, gathering high-quality training data, selecting appropriate AI technologies, considering ethical implications, and establishing robust evaluation metrics for assessing the classifier's effectiveness.

Design Thinking:

1.Data Collection:

We will need a dataset containing labeled examples of spam and nonspam messages. We can use a Kaggle dataset for this purpose.

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

2.Data Preprocessing:

The text data needs to be cleaned and preprocessed. This involves removing special characters, converting text to lowercase, and tokenizing the text into individual words.

Code for data prepossessing:

```
import pandas as pd
import re
import nltk
from nltk.tokenize import word_tokenize

# Assuming you have a CSV file with a 'text' column containing the text data.
# Replace 'your_dataset.csv' with your CSV file's name.
df = pd.read_csv('your_dataset.csv')

# Define a function to clean and preprocess text
def preprocess_text(text):
    # Remove special characters and numbers using regex
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Convert text to lowercase
    text = text.lower()

    # Tokenize the text into individual words using NLTK's word_tokenize
    words = word_tokenize(text)
```

```
    # Join the tokenized words into a single string
    text = ' '.join(words)

    return text

# Apply the preprocess_text function to the 'text' column
df['cleaned_text'] = df['text'].apply(preprocess_text)

# Print the cleaned and tokenized text
print(df['cleaned_text'])
```

3.Feature Extraction : Convert the tokenized words into numerical feature using techniques like TF-IDF(Term Frequency -Inverse -Document - Frequency)

Procedure:

To convert the tokenized words into numerical features using the TF-IDF (Term Frequency-Inverse Document Frequency) technique, you can use the TfidfVectorizer from the Scikit-learn library. Here's Python code to do this:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample DataFrame with 'cleaned_text' column obtained from previous steps
data = {'cleaned_text': ["this is a sample document", "another document", "yet another example"]}
df = pd.DataFrame(data)

# Create a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the cleaned text data
tfidf_matrix = tfidf_vectorizer.fit_transform(df['cleaned_text'])

# Convert TF-IDF matrix to a DataFrame (optional)
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vectorizer.get_feature_names_out())
```

```
# Print the TF-IDF DataFrame
print(tfidf_df)
```

In this code:

1.We create a sample DataFrame `df` with a 'cleaned_text' column containing tokenized and cleaned text data.

2.We create a `TfidfVectorizer` instance, which will transform the tokenized text data into TF-IDF features.

3.We fit and transform the 'cleaned_text' data using the vectorizer, resulting in a TF-IDF matrix (`tfidf_matrix`).

4.Optionally, we convert the TF-IDF matrix into a DataFrame (`tfidf_df`) for better readability.

Now, `tfidf_df` contains the TF-IDF features for each document (row) in your dataset, where each column represents a unique term from the corpus. You can use these features as input for training machine learning models or further analysis in your spam classifier project.

4.Model Selection :

We can experiment with various machine learning algorithm such as Naive Bayes ,SVM,and deep learning neural networks.

1. Naive Bayes:

- Naive Bayes classifiers are simple and often perform well on text classification tasks like spam detection.
- Use the `MultinomialNB` class from Scikit-learn for text classification.
- Split your data into training and testing sets (if not already done).
- Train the Naive Bayes classifier on your training data.

- Evaluate its performance using metrics like accuracy, precision, recall, and F1-score.

2. Support Vector Machines (SVM):

- SVMs are versatile and can be effective for text classification tasks.
- Use the SVC class from Scikit-learn for SVM-based classification.
- Preprocess your text data and convert it into numerical features (e.g., TF-IDF) as shown in previous steps.
- Split your data into training and testing sets.
- Train the SVM classifier on your training data.
- Evaluate its performance using classification metrics.

3. Deep Learning Neural Networks:

- Deep learning models like neural networks, particularly recurrent neural networks (RNNs) or convolutional neural networks (CNNs), can capture complex patterns in text data.
- You can use deep learning frameworks such as TensorFlow or PyTorch for this.
- Preprocess your text data and convert it into numerical input suitable for neural networks.
- Design and build your neural network architecture, considering layers like embedding layers, LSTM or CNN layers, and output layers for binary classification.
- Split your data into training and testing sets.
- Train the neural network on your training data.
- Evaluate its performance using classification metrics.

Naive Bayes classifier using Scikit-learn:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Assuming you have a DataFrame with 'cleaned_text' and 'label' columns
X = tfidf_matrix  # Your TF-IDF features
y = df['label']   # Your spam/non-spam labels

# Split the 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)

# Create and train a Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print evaluation metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

5.Evaluation :

We will measure the models performance using metrics like accuracy,precision,recall and F1-score.

1.Accuracy:

- Accuracy measures the ratio of correctly classified instances to the total number of instances.
- It gives an overall view of your model's performance but may not be suitable for imbalanced datasets.
- It can be calculated using the formula:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Where:

- TP: True Positives (spam correctly classified as spam)

- TN: True Negatives (non-spam correctly classified as non-spam)
- FP: False Positives (non-spam incorrectly classified as spam)
- FN: False Negatives (spam incorrectly classified as non-spam)

2.Precision:

- Precision measures the accuracy of positive predictions made by the model.
- It is the ratio of true positives to the total positive predictions.
- High precision indicates that when the model predicts spam, it's often correct.
- It can be calculated using the formula:

$$\text{Precision} = TP / (TP + FP)$$

3.Recall (Sensitivity or True Positive Rate):

- Recall measures the model's ability to identify all relevant instances in the dataset.
- It is the ratio of true positives to the total actual positives.
- High recall indicates that the model can catch most of the actual spam.
- It can be calculated using the formula:

$$\text{Recall} = TP / (TP + FN)$$

4.F1-Score:

- F1-score is the harmonic mean of precision and recall.
- It balances precision and recall, providing a single metric for evaluating the model's overall performance.
- It can be calculated using the formula:

$$\text{F1-Score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

To evaluate your spam classifier, you can use these metrics by comparing the model's predictions (e.g., `y_pred`) against the true labels (e.g., `y_test`) for your test dataset. Here's how you can calculate these metrics using Python:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Assuming you have predictions (y_pred) and true labels (y_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

6.Iterative Improvement:

Fine tune the model and experiment with hyperparameters to improve its accuracy

SVM classifier

```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Split your data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, df['label'], test_size=0.2,
random_state=42)

# Define the hyperparameters grid to search
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'kernel': ['linear', 'rbf'],  # Kernel type
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1],  # Kernel coefficient for 'rbf' kernel
}

# Create the SVM classifier
svm_classifier = SVC()
```

```
# Use GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid,
scoring='accuracy', cv=5, verbose=2, n_jobs=-1)

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print(f"Best Hyperparameters: {best_params}")

# Get the best model with the tuned hyperparameters
best_svm_classifier = grid_search.best_estimator_

# Evaluate the best model on the test data
y_pred = best_svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print the evaluation results
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```