**IBM NaanMudhalvan – Artificial Intelligence Group 4**

**Phase 1:**

**Project Submission**

**Topic:**

**Building a Smarter AI-Powered Spam Classifier**

**Team members:**

**K. Shreenithi (922121106088)**

**K. Vikashini (922121106110)**

**R. Vishnu Priya (922121106111)**

**K. Sivasankari (922121106090)**

**M. Sasmitha Shree (922121106082)**

**College name:**

**SSM INSTITUTE OF ENGINEERING AND TECHNOLOGY, DINDIGUL**

**College code: 9221**

**Phase 1:**

**Problem Definition and Design Thinking**

**Problem Definition:**

Building a smarter AI-powered spam classifier involves developing a machine learning or artificial intelligence system that can accurately differentiate between spam and legitimate messages in various communication channels, such as emails, messages, or comments. This typically includes:

1. Data Collection

2. Feature Extraction

3. Model Selection

4. Training

5. Evaluation

6. Fine-tuning

7. Deployment

8. Continuous monitoring

The goal is to create an AI system that can effectively filter out spam messages while minimizing false positives, ensuring that legitimate messages are not incorrectly marked as spam. This process requires ongoing maintenance and improvement to stay ahead of spammers’ tactics and maintain a high level of accuracy.

**Design Thinking:**

Building a smarter AI-powered spam classifier involves several steps. Here’s a high-level overview of the process:

1. ***\*\*Define the Problem\*\*:***

- Clearly define the problem you want to solve. In this case, it’s classifying emails as spam or not spam (ham).

2. ***\*\*Data Collection\*\*:***

- Gather a large and diverse dataset of labeled emails. You need both spam and non-spam examples to train and evaluate your classifier.

3. ***\*\*Data Preprocessing\*\*:***

- Clean and preprocess the data. This involves tasks like removing HTML tags, special characters, and normalizing text (e.g., converting all text to lowercase).

4. ***\*\*Feature Extraction\*\*:***

- Convert the text data into numerical features that machine learning algorithms can work with. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) and word embeddings like Word2Vec or GloVe.

5. ***\*\*Split Data\*\*:***

- Divide your dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set assesses the model’s performance.

6. ***\*\*Model Selection\*\*:***

- Choose an appropriate machine learning algorithm or model for text classification. Common choices include Naïve Bayes, Support Vector Machines, Decision Trees, and deep learning models like Recurrent Neural Networks (RNNs) or Transformers.

7. ***\*\*Model Training\*\*:***

- Train the selected model on the training data. Optimize hyperparameters and monitor performance on the validation set.

8. ***\*\*Evaluation\*\*:***

- Assess the model’s performance using evaluation metrics like accuracy, precision, recall, F1-score, and ROC curves. Adjust the model and features as needed.

9. ***\*\*Tuning and Optimization\*\*:***

- Fine-tune the model to improve performance. This may involve adjusting hyperparameters, using techniques like cross-validation, or experimenting with different algorithms.

10. \****\*Testing\*\*:***

- Evaluate the final model on the testing set to get an unbiased estimate of its performance.

11. ***\*\*Deployment\*\*:***

- Integrate the trained model into your email system or application, making it capable of classifying incoming emails in real-time.

12. ***\*\*Monitoring\*\*:***

- Continuously monitor the classifier’s performance in a production environment and implement mechanisms for feedback and retraining.

13. ***\*\*Feedback Loop\*\*:***

- Implement a feedback loop where misclassified emails are used to retrain and improve the model over time.

14. ***\*\*Security and Privacy\*\*:***

- Ensure that the spam classifier doesn’t inadvertently compromise user privacy or security.

15. \****\*Scalability\*\*:***

- Design the system to handle a large volume of emails efficiently as your user base grows.

16. ***\*\*Regular Updates\*\*:***

- Keep the spam classifier up to date by periodically retraining it with new data and adapting to evolving spam tactics.

17. ***\*\*User Interface\*\*:***

- If applicable, create a user-friendly interface for users to report false positives or false negatives to further improve the system.

Building a smarter AI-powered spam classifier is an ongoing process that involves iterative improvements and constant vigilance against new spam techniques. It requires a combination of data, machine learning expertise, and software engineering skills.

**Coding with explanation:**

**Code:**

Building a smarter AI-powered spam classifier involves coding various components using Python and the scikit-learn library:

**# Step 1: Import necessary libraries**

Import pandas as pd

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

From sklearn.feature\_extraction.text import TfidfVectorizer

From sklearn.naive\_bayes import MultinomialNB

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

**# Step 2: Data Collection**

Data = pd.read\_csv(‘spam\_data.csv’)

**# Step 3: Data Preprocessing**

Import pandas as pd

Import re

Import nltk

From nltk.corpus import stopwords

From nltk.tokenize import word\_tokenize

Data = pd.read\_csv(‘spam\_data.csv’)

Def clean\_text(text):

Cleanr = re.compile(‘<.\*?>’)

Text = re.sub(cleanr, ‘’, text)

Text = re.sub(r’[^a-zA-Z]’, ‘ ‘, text)

Return text

Data[‘text’] = data[‘text’].apply(clean\_text)

Nltk.download(‘punkt’)

Data[‘text’] = data[‘text’].apply(lambda x: word\_tokenize(x.lower()))

Nltk.download(‘stopwords’)

Stop\_words = set(stopwords.words(‘english’))

Data[‘text’] = data[‘text’].apply(lambda x: [word for word in x if word not in stop\_words])

Data[‘text’] = data[‘text’].apply(lambda x: ‘ ‘.join(x))

**# Step 4: Feature Extraction**

Tfidf\_vectorizer = TfidfVectorizer(max\_features=5000) # Adjust max\_features as needed

X = tfidf\_vectorizer.fit\_transform(data[‘text’])

Y = data[‘label’]

**# Step 5: Split Data**

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

**# Step 6: Model Selection**

Model = MultinomialNB() # Using a Naïve Bayes classifier, you can explore other models as well

**# Step 7: Model Training**

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

**# Step 8: Evaluation**

Y\_pred = model.predict(X\_test)

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

Print(f’Accuracy: {accuracy:.2f}’)

# You can also print a detailed classification report for precision, recall, etc.

Print(classification\_report(y\_test, y\_pred))

**Explanation:**

* We start by importing necessary libraries, including pandas for data handling, scikit-learn for machine learning, and TfidfVectorizer for feature extraction.
* Data Collection (Step 2): We assume that you have a CSV file with ‘text’ (email content) and ‘label’ (spam or not spam) columns.
* Data Preprocessing (Step 3): In this example, we assume that the data is already preprocessed (cleaned and normalized). In practice, you may need to perform additional preprocessing, such as removing special characters or stemming.
* Feature Extraction (Step 4): We use TF-IDF vectorization to convert the text data into numerical features. Adjust the `max\_features` parameter based on your dataset and memory constraints.
* Split Data (Step 5): We split the dataset into training and testing sets to evaluate the model’s performance.
* Model Selection (Step 6): We choose a Naïve Bayes classifier for this example, but you can explore other models like Support Vector Machines or deep learning models.
* Model Training (Step 7): We train the selected model on the training data.
* Evaluation (Step 8): We evaluate the model’s performance on the test data and print accuracy and a classification report for more details.

Remember that building a production-ready spam classifier involves more complexities, such as handling large-scale data, dealing with imbalanced datasets, and ensuring user privacy and security. The code provided here serves as a basic starting point for educational purposes.