IBM Naan Mudhalvan

Phase 2 – Project Submission

Topic:

“Building a Smarter AI-Powered Spam Classifier”

Designing an SMS spam classifier involves several steps. Here's a detailed overview of the process:

Problem Definition:

- **Define the problem:** Creating a machine learning model to classify SMS messages into spam or non-spam (ham).
- **Understand the data:** Gather a dataset containing labeled SMS messages, distinguishing between spam and non-spam.

1. Data Preprocessing:

a. Data Cleaning:

- **Remove duplicates:** Eliminate identical messages to avoid bias.
- **Handle missing data:** Address any missing or null values in the dataset.

b. Text Preprocessing:

- **Tokenization:** Split messages into individual words (tokens).
- **Lowercasing**: Convert all text to lowercase to ensure uniformity.
- **Removing special characters and numbers:** Clean the text to retain only words.

- **Stopword Removal:** Eliminate common words (e.g., “and”, “the”) that don’t carry significant meaning.
- **Stemming or Lemmatization:** Reduce words to their root form to enhance feature extraction.

Code:

```
Import pandas as pd
#Load data (assuming you have a DataFrame named ‘data’)
 data = pd.read_csv(“sms_spam_dataset.csv”)

# Data Cleaning
Data.drop_duplicates(inplace=True)
Data.dropna(subset=[‘message’], inplace=True)

# Text Preprocessing
Import re
From nltk.tokenize import word_tokenize
From nltk.corpus import stopwords
From nltk.stem import PorterStemmer
Stop_words = set(stopwords.words(‘english’))
Ps = PorterStemmer()
def clean_text(text):
   text = re.sub(r'\W', ' ', text)
   tokens = word_tokenize(text)
   tokens = [ps.stem(word.lower()) for word in tokens if
word.isalpha()]
   tokens = [word for word in tokens if word not in
stop_words]
   return ' '.join(tokens)
data[‘clean_message’] = data[‘message’].apply(clean_text)
```

2. Feature Extraction:

- **Bag of Words (BoW):**Convert text data into numerical vectors, counting the frequency of words in each message.
- **TF-IDF (Term Frequency-Inverse Document Frequency):** Assign weights to words based on their importance in individual messages and the entire dataset.

Code:

```
from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer

# Bag of Words (BoW)
count_vectorizer = CountVectorizer()
X_bow =
count_vectorizer.fit_transform(data['clean_message'])

# TF-IDF
tfidf_vectorizer = TfidfVectorizer()
X_tfidf =
tfidf_vectorizer.fit_transform(data['clean_message'])
```

3. Model Selection:

- Choose a suitable machine learning algorithm (e.g., Naïve Bayes, Support Vector Machines, or Neural Networks) for text classification.
- Split the data into training and testing sets to evaluate the model's performance accurately.

4. Model Training:

- Train the selected model using the preprocessed and feature-extracted data.
- Optimize hyperparameters through techniques like cross-validation to enhance the model's accuracy.

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_tfidf,
data['label'], test_size=0.2, random_state=42)

# Choose and train the model
model = MultinomialNB()
model.fit(X_train, y_train)
```

5. Model Evaluation:

- Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score.
- Adjust the model and retrain if necessary to achieve the desired performance.

Code:

```
# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print detailed metrics
print(classification_report(y_test, y_pred))
```

6. Deployment:

- Deploy the trained model into a production environment, making it accessible for classifying new SMS messages.
- Set up an interface (like a web or mobile app) for users to input SMS messages and receive classification results.

Code:

```
pip install Flask
  from flask import Flask, request, jsonify
  import pickle
app = Flask(__name__)

  # Load the trained model and vectorizer
  with open('model.pkl', 'rb') as model_file:
     model = pickle.load(model_file)

  with open('tfidf_vectorizer.pkl', 'rb') as vectorizer_file:
     tfidf_vectorizer = pickle.load(vectorizer_file)

  @app.route('/predict', methods=['POST'])
  def predict():
     try:
        # Get the SMS message from the request
        data = request.get_json(force=True)
        message = data['message']
```

```
        # Preprocess the message using the TF-IDF
vectorizer
        processed_message = clean_text(message)
        tfidf_message =
tfidf_vectorizer.transform([processed_message])

        # Make a prediction using the pre-trained model
        prediction = model.predict(tfidf_message)

        # Return the prediction as JSON response
        return jsonify({'prediction': str(prediction[0])})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(port=5000)
```

3. **Run the Flask App:**
   ```
   python app.py
   ```

7. Monitoring and Maintenance:

- Implement regular monitoring to ensure the model's accuracy over time.
- Update the model periodically with new data to adapt to evolving spam patterns.

Code:

1.Scheduled Monitoring Tasks

```
import schedule
import time
def monitor_performance():
    # Code to monitor model performance (metrics calculation and logging)
    pass

# Schedule monitoring task to run every day at a specific time
schedule.every().day.at("12:00").do(monitor_performance)

# Keep the program running to execute scheduled tasks
while True:
    schedule.run_pending()
    time.sleep(1)
```

2. Handling Model Drift and Retraining (Assuming a Retraining Function)

```
def check_model_drift():
    # Code to detect model drift and trigger retraining if necessary
    # ...
    if drift_detected:
        retrain_model()

# Schedule model drift check task to run every week
schedule.every().week.do(check_model_drift)
```

8. Feedback Loop:

- Gather user feedback on the classified messages to further improve the model.
- Iteratively refine the model based on user input and emerging spam tactics.

Code:

3. User Feedback Processing (Using a Flask API Endpoint):

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/feedback', methods=['POST'])
def process_feedback():
    feedback_data = request.get_json()
    # Code to process user feedback and update the model
    # ...
    return jsonify({'message': 'Feedback received and
processed successfully!'})

if __name__ == '__main__':
    app.run(port=5000)
```