Building a Smarter AI-Powered Spam Classifier: Designing a Web Application to Classify Spam Messages Using TF-IDF, Multinomial Naive Bayes, and Other NLTK Libraries with Iterative Improvement to Enhance Accuracy, Precision, Recall, and F1-score.

CONTEXT:

1.MACHINE LEARNING ALGORITHM
2.TRAINING MODEL
3.PERFORMANCE EVALUATION

1.MACHINE LEARNING ALGORITHM:

1. The "Machine Learning Model" part of the code focuses on building and using the spam classification model.
2. It imports necessary libraries for machine learning, such as Multinomial Naive Bayes, TF-IDF vectorization, and performance metrics.
3. The code defines a class called **SpamClassifierModel**, which will hold the model and related functions.
4. The constructor initializes the model and loads the data from a CSV file.
5. The **refreshModel** method performs text preprocessing, TF-IDF vectorization, and model training using Multinomial Naive Bayes.
6. It also computes and stores various performance metrics like accuracy, precision, recall, and F1 score.
7. The **predict** method accepts new text, preprocesses it, and predicts whether it's spam or not using the trained model.
8. If it predicts spam, it calls the **iteratePerformance** method to update the model's knowledge.
9. The **iteratePerformance** method appends the new data to the dataset and refreshes the model for continuous learning.
10. The model can be used for real-time spam classification with performance updates.

CODE:

```
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
class SpamClassifierModel():
   def __init__(self,):
      self.data = pd.read_csv(r"./src/data/spam.csv",sep="\t",names=["label", "message"])
      self.refreshModel()
   def refreshModel(self,):
      self.porterStemmer = PorterStemmer()
      corpus = []
      for i in range(0, len(self.data)):
         review = re.sub('[^a-zA-Z]', ' ', self.data['message'][i]).lower().split()
         review = [self.porterStemmer.stem(word) for word in review if not word in
stopwords.words('english')]
         review = ' '.join(review)
         corpus.append(review)

      self.TfidfVectorization = TfidfVectorizer(max_features=2500)
      X = self.TfidfVectorization.fit_transform(corpus).toarray()
      y=pd.get_dummies(self.data['label']).iloc[:,1].values
   def predict(self,new_text):
      new_review = re.sub('[^a-zA-Z]', ' ', new_text).lower().split()
      new_review = [self.porterStemmer.stem(word) for word in new_review if not word in
stopwords.words('english')]
      new_review = ' '.join(new_review)

      new_X = self.TfidfVectorization.transform([new_review]).toarray()

      prediction = self.spam_detect_model.predict(new_X)
      if prediction[0]:
         self.iteratePerformance(new_text,"spam")
      else:
         self.iteratePerformance(new_text,"ham")

      return prediction
   def iteratePerformance(self,new_message,prediction):
      self.data.loc[len(self.data)]={"label":prediction,"message":new_message}
      self.refreshModel()
      print({"label":prediction,"message":new_message})
      print(self.data)
```

2.TRAINING MODEL:

1. The "Training Model" part focuses on data preparation and initial model training.
2. It imports libraries for data handling, such as Pandas, regular expressions, and NLTK for text processing.
3. The class **SpamClassifierModel** is defined, and the constructor loads a dataset from a CSV file.

4. The **refreshModel** method processes the text data, removing non-alphabetical characters and stemming words.
5. It creates a TF-IDF vectorization of the text and splits the data into training and testing sets.
6. The Multinomial Naive Bayes model is trained on the training data.
7. Performance metrics (accuracy, precision, recall, and F1 score) are computed and stored.
8. This part is responsible for initializing the model with an initial dataset and training it for the first time.
9. It's a one-time process that sets up the model for future real-time use in the "Machine Learning Model" part.
10. Once this part is executed, the model is ready to make predictions and learn from new data in the "Machine Learning Model" section.

CODE:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
self.spam_detect_model = MultinomialNB().fit(X_train, y_train)
 y_predict=self.spam_detect_model.predict(X_test)
```

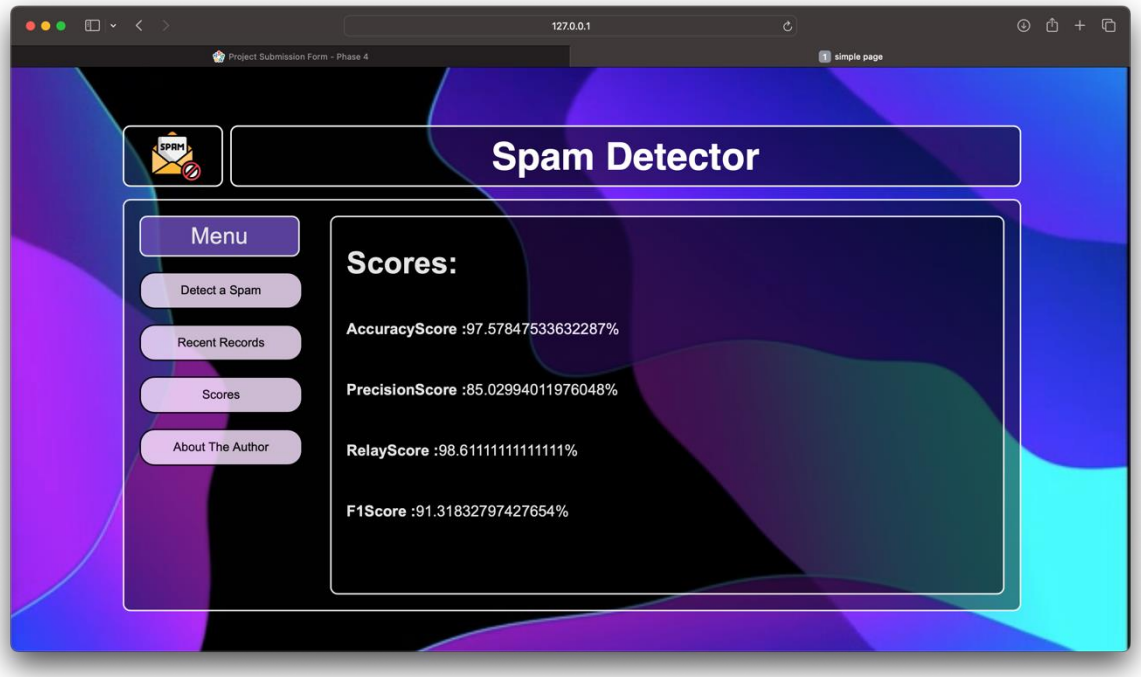
3.PERFORMANCE EVALUATION:

1. **Accuracy Score:** Accuracy measures the overall correctness of the model's predictions. It's the ratio of correctly classified instances to the total number of instances in the test dataset. In the code, **accuracy_score** is used to calculate accuracy.
2. **Precision Score:** Precision measures the accuracy of positive predictions made by the model. It's the ratio of true positives (correctly predicted spam) to all instances predicted as spam. High precision indicates a low rate of false positive spam predictions.
3. **Recall (Sensitivity) Score:** Recall measures the model's ability to identify all positive instances in the dataset. It's the ratio of true positives to all actual positive instances. High recall suggests that the model rarely misses spam messages.
4. **F1 Score:** The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall. F1 score is especially useful when there is an imbalance between the number of spam and non-spam messages.

CODE:

```
self.currectAccuracyScore = accuracy_score(y_predict, y_test)
self.currentPrecisionScore = precision_score(y_predict, y_test)
self.currentRelayScore = recall_score(y_predict, y_test)
self.currentf1Score = f1_score(y_predict, y_test)
```

DEMO:
BEFORE EVALUATION:



AFTER EVALUATION: