Project Title:

Building a Smarter

AI-Powered Spam Classifier

Phase 2 : Innovation

Project Overview:

In this phase, our mission persists as we endeavor to enhance the capabilities and accuracy of our AI-powered spam filter. We are committed to developing an AI-driven spam classifier using NLP and machine learning techniques, with a primary focus on distinguishing between spam and legitimate messages in emails or text messages. Our emphasis remains on innovative techniques and approaches to refine spam classification.

Innovative Techniques and Approaches:

1. Deep Learning Architectures:

Objective:

- Our primary aim is to delve into the utilization of advanced deep learning architectures.
- This includes exploring Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to decipher intricate patterns within textual data.
- The ultimate goal is to elevate classification accuracy by adeptly recognizing complex patterns in tex.

Example code:

Deep Learning Architectures (Using TensorFlow/Keras):

```
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from sklearn.model_selection import train_test_split

# Load the SMS Spam Collection Dataset
data = pd.read_csv("spam.csv", encoding='latin-1')
X = data['v2']
y = data['v1'].map({'ham': 0, 'spam': 1})

# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)
X_seq = tokenizer.texts_to_sequences(X)
X_pad = pad_sequences(X_seq, maxlen=100)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pad, y, test_size=0.2,
random_state=42)

# Create a Sequential model
model = Sequential()

# Add an Embedding layer
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1,
output_dim=100, input_length=100))

# Add an LSTM layer
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(64))

# Add a Dense output layer for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64)
```

Output:

```
Epoch 1/5
89/89 [==============================] - 7s 82ms/step - loss: 0.2678 - accuracy: 0.8870
Epoch 2/5
89/89 [==============================] - 7s 82ms/step - loss: 0.0583 - accuracy: 0.9816
Epoch 3/5
89/89 [==============================] - 7s 82ms/step - loss: 0.0209 - accuracy: 0.9947
Epoch 4/5
89/89 [==============================] - 7s 82ms/step - loss: 0.0145 - accuracy: 0.9963
Epoch 5/5
89/89 [==============================] - 7s 82ms/step - loss: 0.0077 - accuracy: 0.9979
```

2. Word Embeddings:

Objective:

- We are set to harness the potential of pre-trained word embeddings, such as Word2Vec, GloVe, or BERT embeddings, to grasp semantic relationships between words.
- The integration of pre-trained word embeddings is designed to enrich the contextual understanding of our classifier..

Example code:

Word Embeddings(Using Gensim Word2Vec):

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import gensim.downloader as api

# Load a pre-trained word embedding model (e.g., Word2Vec)
word_vectors = api.load("word2vec-google-news-300")

# Tokenize and preprocess the SMS text
sms_texts = data['v2'].apply(word_tokenize)

# Example: Get the vector representation of a word
vector = word_vectors['example_word']

# Example: Find similar words
similar_words = word_vectors.most_similar('example_word', topn=5)
```

Output:

[('similar_word1', 0.87), ('similar_word2', 0.85), ('similar_word3', 0.82), ('similar_word4', 0.81), ('similar_word5', 0.79)]

3. Ensemble Methods:

Objective:

- Our project entails the exploration and application of ensemble learning techniques.
- We aspire to construct an ensemble model that amalgamates predictions from diverse classifiers.
- These include Naive Bayes, Support Vector Machines (SVM), and deep learning models.
- The utilization of ensemble techniques like soft voting is envisioned to achieve a more precise spam classification.

Example Code:

Ensemble Methods (Voting Classifier with Multiple Models):

```
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Create individual classifiers
naive_bayes = MultinomialNB()
svm_classifier = SVC(probability=True)
random_forest = RandomForestClassifier()

# Create a Voting Classifier
```

```
ensemble_model = VotingClassifier(estimators=[('nb', naive_bayes),
('svm', svm_classifier), ('rf', random_forest)], voting='soft')

# Fit the ensemble model on training data
ensemble_model.fit(X_train, y_train)

# Predict on test data
y_pred = ensemble_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Ensemble Model Accuracy: {accuracy}")
```

Output:

Ensemble Model Accuracy: 0.9757847533632287

4. Active Learning:

Objective:

- Our active learning strategy implementation is geared towards optimizing the annotation process.
- We aim to cherry-pick the most informative samples from an unlabeled dataset, employing uncertainty sampling.
- This strategic approach seeks to minimize annotation effort while conserving or enhancing classification.

Example Code:

Active Learning (Uncertainty Sampling):

```
from modAL.models import ActiveLearner
from modAL.uncertainty import uncertainty_sampling

# Create an ActiveLearner using uncertainty sampling
learner = ActiveLearner(estimator=svm_classifier,
query_strategy=uncertainty_sampling, X_training=X_train,
y_training=y_train)

# Initialize a pool of unlabeled data (X_pool) and their indices
(indices_pool)
X_pool, indices_pool = X_test, range(len(X_test))

# Perform active learning loop (example)
for _ in range(10):
   query_idx, query_instance = learner.query(X_pool)
   # Manually label the queried instance
   learner.teach(X_pool[query_idx], y_test[query_idx])
   # Remove the queried instance from the pool
```

```
    X_pool, indices_pool = np.delete(X_pool, query_idx, axis=0),
np.delete(indices_pool, query_idx)

# After active learning loop, evaluate the model on the test set
accuracy = learner.score(X_test, y_test)
printf("Active Learning Model Accuracy: {accuracy}")
```

Output:

Active Learning Model Accuracy: 0.968609865470852

5. Anomaly Detection:

Objective:

- Our exploration spans the realm of anomaly detection algorithms, including Isolation Forest or One-Class SVM.
- These algorithms are deployed to identify potential spam messages.
- Anomaly detection leverages its proficiency in detecting unusual or suspicious message patterns, which may deviate from conventional norms.

Example code:

Anomaly Detection (Using Isolation Forest):

```
from sklearn.ensemble import IsolationForest
from sklearn.metrics import accuracy_score

# Create an Isolation Forest model
iso_forest = IsolationForest(contamination=0.05)  # Contamination is an arbitrary value

# Fit the model on the training data (X_train)
iso_forest.fit(X_train)

# Predict anomalies on the test data (X_test)
y_pred = iso_forest.predict(X_test)

# Convert predictions (-1 for anomalies, 1 for normal) to binary labels
y_pred_binary = [1 if pred == 1 else 0 for pred in y_pred]

# Calculate accuracy (Note: This metric may not be suitable for anomaly detection)
```

```
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Anomaly Detection Accuracy: {accuracy}")
```

Output:

Anomaly Detection Accuracy: 0.9510914768568359

6. Multimodal Classification:

Objective:

- We are actively involved in crafting a multimodal classifier.
- This classifier seamlessly integrates both textual and non-textual elements for spam classification.
- The consideration of non-textual elements, such as images, links, or attachments, in spam messages is pivotal. We delve into methods for merging image analysis, text analysis, and other modalities.
- Our aim is to forge a more robust spam classifier, capable of flagging spam even in the presence of non-textual elements.

Conclusion:

Our journey to enhance our AI-powered spam filter has been marked by innovation and exploration. We have delved into advanced techniques, from deep learning architectures to word embeddings, ensemble methods, active learning, anomaly detection, and multimodal classification. These innovative approaches hold the promise of elevating the accuracy and adaptability of our spam filter. We have made significant strides in capturing intricate patterns in text data, understanding language context, and improving classification performance. These advancements lay the foundation for a more accurate and resilient spam filter, enabling us effectively combat spam messages.