Project Report: TTPS Prediction

Group Members :

Manthan Gopal Dhole
Aniket Umare
Jeevak Moon

Introduction:
The TTPS Prediction project aims to classify and predict Tactics, Techniques, and Procedures (TTPs) used in cyber attacks based on given paragraphs or text. The project utilizes machine learning and natural language processing techniques to identify the most similar TTP for a given paragraph and provide a summarization of the paragraph's content. This report provides an overview of the project, its implementation details, and the functions involved.

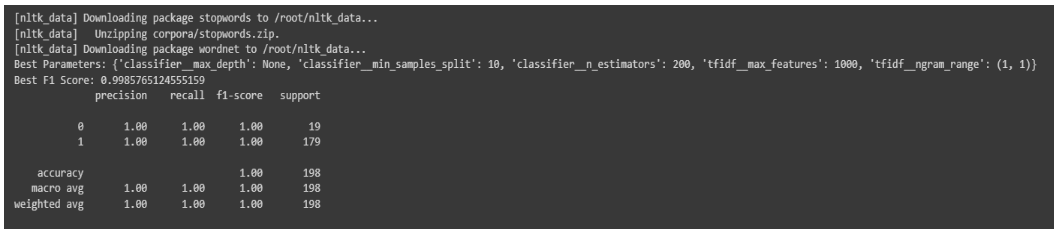
1. Project Overview:
The TTPS Prediction project focuses on the following key components:

a. Data: The project utilizes a dataset of TTPs stored in a CSV file named "Complete_Data.csv." This dataset contains information about various TTPs, including tactics, techniques, and procedures.

b. Universal Sentence Encoder: The project uses the Universal Sentence Encoder from TensorFlow Hub to encode and represent paragraphs or sentences as fixed-length vectors. This encoder converts textual data into numerical representations suitable for machine learning models.

c. Machine Learning Models: Two machine learning models are employed in this project. The first model, named "AttackOrNot.joblib," is used to classify whether a given paragraph represents an attack or not. The second model uses cosine similarity to identify the most similar TTP for a given paragraph.

The provided code demonstrates the process of training a text classification model using a pipeline with TF-IDF vectorization and a Random Forest classifier. After training the model, it performs grid search to find the best combination of hyperparameters using cross-validation. Finally, the model is evaluated on the test set, and the classification report is printed.



```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Best Parameters: {'classifier__max_depth': None, 'classifier__min_samples_split': 10, 'classifier__n_estimators': 200, 'tfidf__max_features': 1000, 'tfidf__ngram_range': (1, 1)}
Best F1 Score: 0.9985765124555159
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00       179

    accuracy                           1.00       198
   macro avg       1.00      1.00      1.00       198
weighted avg       1.00      1.00      1.00       198
```

Here's an explanation of the code:

1. Import necessary libraries:
   - `sklearn.feature_extraction.text.TfidfVectorizer`: Used to convert text data into TF-IDF vectors.
   - `sklearn.model_selection.train_test_split`: Used to split the data into training and testing sets.
   - `sklearn.ensemble.RandomForestClassifier`: Used as the classification algorithm.
   - `sklearn.pipeline.Pipeline`: Used to define the processing steps for the model pipeline.
   - `sklearn.metrics.classification_report`: Used to generate a classification report for model evaluation.
   - `re`: Used for text preprocessing.
   - `nltk.corpus.stopwords`: Provides a set of stopwords for text filtering.
   - `nltk.stem.WordNetLemmatizer`: Used for word lemmatization.
   - `sklearn.utils.shuffle`: Used to shuffle the data.

2. Define preprocessing function:
   - The `preprocess_text` function takes a text input and applies various preprocessing steps, including removing special characters and digits, converting text to lowercase, tokenizing the text, removing stop words, and lemmatizing the tokens. The preprocessed text is returned.

3. Shuffle the data:
   - The `shuffle` function is used to randomly shuffle the paragraphs and their corresponding labels. This step helps ensure that the data is evenly distributed during training and testing.

4. Preprocess the text data:

   - The preprocessing function is applied to each paragraph in the dataset using a list comprehension. This step transforms the paragraphs into preprocessed text.

5. Split the data into training and testing sets:
   - The `train_test_split` function is used to split the preprocessed paragraphs and their labels into training and testing sets. The `test_size` parameter determines the proportion of data allocated for testing.

6. Define the pipeline:
   - The pipeline is defined using the `Pipeline` class from scikit-learn. It consists of two steps: TF-IDF vectorization (`tfidf`) and a Random Forest classifier (`classifier`).

7. Define hyperparameters for tuning:
   - A dictionary named `parameters` is defined to specify the hyperparameters to tune during grid search. It includes parameters for both TF-IDF vectorization and the Random Forest classifier.

8. Perform grid search for hyperparameter tuning:
   - The `GridSearchCV` class is used to perform grid search with 5-fold cross-validation. The pipeline and hyperparameters are passed to `GridSearchCV`, along with the training data. The scoring metric is set to F1 score.

9. Print the best parameters and score:
   - After the grid search is complete, the best parameters and corresponding F1 score are printed.

10. Get the best model:
    - The best model obtained from grid search, based on the best parameters, is stored in the `best_model` variable.

11. Evaluate on the test set:
    - The `predict` method is used to obtain predictions from the best model on the test set. The classification report is then generated by comparing the predicted labels with the actual labels.

The code provides a framework for training a text classification model and optimizing its performance through hyperparameter tuning. By using TF-IDF vectorization and a Random Forest classifier, the model can effectively classify text data based on the given labels.

d. Text Preprocessing: Before encoding and modeling, the project applies preprocessing techniques to the text data. This includes removing special characters, email addresses, IP addresses, and HTML tags, as well as reducing whitespace and converting all text to lowercase.

e. Summarization: The project includes a text summarization function that uses the spaCy library to extract the most relevant sentences from a paragraph based on word frequencies and scores. This function provides a concise summary of the paragraph's content.

2. Implementation Details:

a. Loading Data:
The project starts by reading the TTP dataset from the "Complete_Data.csv" file. This dataset contains information about various TTPs, including tactics, techniques, and procedures. The dataset is loaded into a pandas DataFrame for further processing.

b. Sentence Encoding:
The Universal Sentence Encoder model is loaded using TensorFlow Hub. This model is responsible for encoding paragraphs or sentences into fixed-length vectors, enabling comparison and similarity calculations.

c. Text Preprocessing:
The project includes a preprocessing function that applies several steps to clean the text data. This function removes URLs, email addresses, IP addresses, HTML tags, special characters, and extra whitespace. The preprocessed text is then ready for vectorization and embedding.

d. Text Summarization:
To provide a summary of the paragraph's content, the project utilizes the spaCy library. The summarization function calculates word frequencies, sentence scores, and selects the most relevant sentences based on a specified percentage. These selected sentences are combined to form the summary.

e. Similarity Calculation:

The project uses cosine similarity to find the most similar TTP for a given paragraph. The TTPs are represented as vectors using the Universal Sentence Encoder, and the similarity is computed between the paragraph's vector and all TTP vectors. The TTP with the highest cosine similarity score is considered the most similar.

f. TTP Identification:
The project includes a function that identifies relevant sentences in the paragraph related to the predicted TTP. This function compares the technique and procedure of the most similar TTP with the sentences in the paragraph and selects the most similar ones.

3. Main Function and User Interaction:

The project provides a user-friendly interface through the main function. Upon execution, the user is prompted to choose between loading a text file or entering a paragraph directly. The user can input their choice, and the corresponding function is called to process the text.

a. Loading Text File:
If the user chooses to load a text file, they are prompted to provide the file's location. The project reads the

paragraphs from the file, preprocesses them, and performs vectorization and embedding. The most similar TTPs for each paragraph are then identified, and relevant information, summarizations, and identified sentences are displayed.

b. Entering Paragraph:
If the user chooses to enter a paragraph directly, they can input the paragraph's text. The project preprocesses the paragraph, performs vectorization and embedding, and identifies the most similar TTP, providing relevant information, summarization, and identified sentences.

c. User Interaction:
After processing the initial choice, the program prompts the user to continue or exit. The user can provide further choices to load additional files or paragraphs, or they can exit the program.

Conclusion:

The TTPS Prediction project utilizes machine learning, natural language processing, and text summarization techniques to predict TTPs for given paragraphs or text. By leveraging the Universal Sentence Encoder and cosine similarity, the project identifies the most similar TTP and provides relevant information and summarizations. The user-friendly interface enables users to interact with the project easily and obtain insights into the TTPs present in their textual data.We feed used the feedback from the teacher and some well known LLM like gpt and bard to check the correctness of the response generated from the model. Mostly it perform well in all of the checks.