OOPs implementation of the TTPs prediction model

All the project related model data : https://drive.google.com/drive/folders/1-6iMy8K-tFxtaFSxxmpnhhjDfAgUQUzp?usp=sharing

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import numpy as np
import joblib
import re
from bs4 import BeautifulSoup
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from sklearn.metrics.pairwise import cosine_similarity
import tensorflow_hub as hub
from heapq import nlargest
from string import punctuation
```

```
class TTPS_prediction:

    def __init__(self):
        self.ttps_Data = pd.read_csv("/content/drive/MyDrive/CyberSecurity/Task_3_TTPS/Complete_Data.csv")
        self.sentenceEncoder_model = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
        self.nlp = spacy.load('en_core_web_sm')
        self.attackOrNot_Model = joblib.load("/content/drive/MyDrive/CyberSecurity/Task_3_TTPS/AttackOrNot.joblib")
        self.ttp_embeddings = joblib.load("/content/drive/MyDrive/CyberSecurity/Task_3_TTPS/ttp_embeddings.pkl")
        self.vectorizer = TfidfVectorizer()

    def read_paragraphs_from_file(self,file_path):
        try:
          with open(file_path, 'r',encoding="UTF-8") as file:
              content = file.read()
              paragraphs = content.split('\n\n')
              paragraphs = [paragraph.strip() for paragraph in paragraphs if paragraph.strip() and len(paragraph.split())<=400]
          self.vectorizationAndEmbedding(paragraphs)
        except :
          print("XXXXXXXXXXXXXX File is not present at give path XXXXXXXXXXX")



    def preprocess_text(self, text):
        # Apply any preprocessing steps (e.g., removing special characters, lowercasing)
        string = re.sub(r'http\S+|www\S+', '', text)

        # Replace email addresses with 'email'
        email_regex = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b')
        string = email_regex.sub('email', string)

        # Replace IP addresses with 'ip'
        ip_regex = re.compile(r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b')
        string = ip_regex.sub('ip', string)

        # Remove newline characters
        string = string.replace('\n', '')
        soup = BeautifulSoup(string, "html.parser")
        string = soup.get_text()

        # Remove special characters
        string = re.sub(r"[^a-zA-Z0-9\s]", "", string)

        # Remove extra whitespaces
        string = re.sub(r"\s+", " ", string)

        preprocessed_text = string.strip()  # Remove leading/trailing whitespaces
        return preprocessed_text

    def summarize(self, text, per):
        doc = self.nlp(text)
        tokens = [token.text for token in doc]
        word_frequencies = {}
        for word in doc:
            if word.text.lower() not in list(STOP_WORDS):
```

```
            if word.text.lower() not in list(STOP_WORDS):
                if word.text.lower() not in punctuation:
                    if word.text not in word_frequencies.keys():
                        word_frequencies[word.text] = 1
                    else:
                        word_frequencies[word.text] += 1
        max_frequency = max(word_frequencies.values())
        for word in word_frequencies.keys():
            word_frequencies[word] = word_frequencies[word] / max_frequency
        sentence_tokens = [sent for sent in doc.sents]
        sentence_scores = {}
        for sent in sentence_tokens:
            for word in sent:
                if word.text.lower() in word_frequencies.keys():
                    if sent not in sentence_scores.keys():
                        sentence_scores[sent] = word_frequencies[word.text.lower()]
                    else:
                        sentence_scores[sent] += word_frequencies[word.text.lower()]
        select_length = int(len(sentence_tokens) * per)
        summary = nlargest(select_length, sentence_scores, key=sentence_scores.get)
        final_summary = [word.text for word in summary]
        summary = ''.join(final_summary)
        return summary

    def most_similar_string(self, sentence, string_list):
        """
        Returns the most similar string in the string list to the given sentence.

        Args:
          sentence: The sentence to compare.
          string_list: A list of strings.

        Returns:
          The most similar string in the string list.
        """

        tfidf_matrix = self.vectorizer.transform(string_list + [sentence])

        sentence_embeddings = self.sentenceEncoder_model([sentence])[0]
        string_embeddings = self.sentenceEncoder_model(string_list)

        cosine_similarities = cosine_similarity([sentence_embeddings], string_embeddings)[0]

        most_similar_index = np.argmax(cosine_similarities)
        return string_list[most_similar_index]

    def vectorizationAndEmbedding(self, paragraphs):
        preprocessed_paragraphs = [self.preprocess_text(paragraph) for paragraph in paragraphs]

        # Fit the vectorizer on the preprocessed paragraphs
        self.vectorizer.fit(preprocessed_paragraphs)

        # Transform the preprocessed paragraphs into vectors

        paragraph_vectors = self.vectorizer.transform(preprocessed_paragraphs)
        paragraph_embeddings = self.sentenceEncoder_model(preprocessed_paragraphs)

        most_similar_ttps = []
        most_similar_indexes = []
        for i in range(len(paragraphs)):
            paragraph_embedding = paragraph_embeddings[i]

            # Compute the cosine similarity between the paragraph embedding and all TTP embeddings
            similarities = cosine_similarity([paragraph_embedding], self.ttp_embeddings)

            # Find the index of the most similar TTP
            most_similar_index = np.argmax(similarities)

            # Retrieve the most similar TTP
            most_similar_ttp = self.ttps_Data.iloc[most_similar_index]
            most_similar_ttps.append(most_similar_ttp)
            most_similar_indexes.append(most_similar_index)

        # Print the most similar TTPs for each paragraph
        for i in range(len(paragraphs)):
            if self.attackOrNot_Model.predict([paragraphs[i]])[0] == 0:
                continue

            print("\nParagraph ", i, " =========\n", paragraphs[i])
            print("Tactic    :", self.ttps_Data.iloc[most_similar_indexes[i]][1])
            print("Technique :", self.ttps_Data.iloc[most_similar_indexes[i]][2])
            print("Procedure :", self.ttps_Data.iloc[most_similar_indexes[i]][3])
            print("Summarization --->", self.summarize(paragraphs[i], 0.3))
```

```
            sentences = paragraphs[i].split('.')
            technique = self.ttps_Data.iloc[most_similar_indexes[i]][2]
            procedure = self.ttps_Data.iloc[most_similar_indexes[i]][3]
            print("Sentences for TTPs identification :")
            most_similar_technique=self.most_similar_string(technique, sentences)
            if(most_similar_technique!=""):
              print("=========>>", most_similar_technique)

            print("=========>>", self.most_similar_string(procedure, sentences))
            print("-------------------------------------------------")

    def Main(self):
      print("Enter your choice ")
      print("1:To load .txt file")
      print("2:To load paragraph")
      print("Press other key to exit")
      choice=int(input("Input choice: "))
      while(choice==1 or choice==2):
        if(choice==1):
          location=input("Enter the location of file : ")
          self.read_paragraphs_from_file(location)
        else:
          string=str(input("Enter the paragraph : "))
          self.vectorizationAndEmbedding([string])

        choice=int(input("Enter the choice :"))
```

```
x = TTPS_prediction()
```

```
x.Main()
```

```
Enter your choice
1:To load .txt file
2:To load paragraph
Press other key to exit
Input choice: 2
Enter the paragraph :  SQL injection, the procedure might involve scanning the target website for vulnerabilities, writing a SQ

Paragraph  0  =========
  SQL injection, the procedure might involve scanning the target website for vulnerabilities, writing a SQL query that includes
Tactic    : T1505.001
Technique : SQL Stored Procedures
Procedure : Adversaries may abuse SQL stored procedures to establish persistent access to systems. SQL Stored Procedures are co
Summarization --->
Sentences for TTPs identification :
=========>>  SQL injection, the procedure might involve scanning the target website for vulnerabilities, writing a SQL query th
=========>>  SQL injection, the procedure might involve scanning the target website for vulnerabilities, writing a SQL query th
-------------------------------------------------
Enter the choice :1
Enter the location of file : /content/drive/MyDrive/CyberSecurity/Task_3_TTPS/threat.txt
<ipython-input-4-d869224d2768>:37: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want
  soup = BeautifulSoup(string, "html.parser")

Paragraph  1  =========
 For the purposes of reporting, IBM considers Latin America to include Mexico, Central America and South America.Incidents in L
while BEC and email thread hijacking tied for third place at 11% each. Extortion and data theft were the most commonly seen imp
Tactic    : T1499
Technique : Endpoint Denial of Service
Procedure : Adversaries may perform Endpoint Denial of Service (DoS) attacks to degrade or block the availability of services t
Summarization ---> Incidents in Latin America bucked global trends, returning retail-wholesale as the most-attacked industry at
Sentences for TTPs identification :
=========>>  Top initial access vectors included external remote services at 30% and exploitation of public-facing applications
=========>>  Ransomware outstripped other attacks in Latin America, accounting for 32% of cases to which X-Force responded
-------------------------------------------------

Paragraph  3  =========
 Distributed Denial of Service Attack against an Industry Sector
A hacktivist group targets a select set of companies for a large-scale distributed denial of service (DDoS)
attack. The group uses a distributed botnet that is loosely coordinated and controlled by members of the group. Byanalyzing tra
services of content distribution providers to deploy DDoS-resistant web architectures.
Tactic    : T1498
Technique : Network Denial of Service
Procedure : Adversaries may perform Network Denial of Service (DoS) attacks to degrade or block the availability of targeted re
Summarization ---> Distributed Denial of Service Attack against an Industry Sector
A hacktivist group targets a select set of companies for a large-scale distributed denial of service (DDoS)
attack.Using network traffic collected by the ISPs, law enforcement agencies can identify the command and control servers, seiz
Sentences for TTPs identification :
```

```
=========>> Distributed Denial of Service Attack against an Industry Sector
A hacktivist group targets a select set of companies for a large-scale distributed denial of service (DDoS)
attack
=========>>  Byanalyzing traffic generated by the botnet, one of the companies targeted in the attack is able to determine that
------------------------------------------------

Paragraph  4  =========
 Financial Conference Phishing Attack:
A cyber crime group makes use of a publicly available conference attendee list to target specific
individuals with a wave of phishing emails. The group is able to identify attendees who are members of
the target organization’s corporate accounting team (i.e., individuals who may have the authority to
```