Hi, this is **Renuka Gadde**

For now, let me discuss my assignment for the midterm that is,

**Cloud-based PE Malware Detection API**

We need to finish three tasks for this project.

**Task 1**:

Deploying the model on AWS SageMaker means getting the model we trained in Lab 5.4 ready to run on AWS SageMaker cloud service.

**Task 2**:

Developing a Python client involves creating a program that takes an executable file, extracts relevant features from it, sends this information to the SageMaker endpoint, and receives the classification results.

**Task 3**:

After developing the Python client, we test it with one malware PE file and one benign PE file from the test dataset created during Lab 5.4. We demonstrate this testing process in our demo video.

Now, I'll describe how I used AWS Sage Maker to implement this project.

Initially, we need to set up a single notebook instance in Amazon Sage Maker.

**Creating a static malware detection file description**

After completing the instance setup, we download all the required libraries to run the trained model. We extract four data files from this code, starting with the first one.

1. The first file is a list of the most common n-grams.
2. The other three files are the trained model files.

The other two files are **imports\_featurizer** and a section named **[section\_name]**. These four files are used for deployment and on the client side.

**Deployment code :**

We have developed an **inference.py** script that SageMaker uses to process the input provided to the API.Top of Form

**model\_fn(model\_dir)** :

This function deserializes the fitted model. It loads the model from the specified directory (model\_dir) using joblib and returns the loaded model.

**input\_fn(request\_body, request\_content\_type):**

This function processes the input data sent to the model. If the content type of the request is JSON (application/json), it parses the JSON request body and returns the input data. Otherwise, it raises a ValueError indicating that the model only supports JSON input.

**predict\_fn(input\_data, model):**

This function makes predictions using the input data and the loaded model. It first processes the input data using the process input function, then uses the model to make predictions based on the processed data. The predictions are returned.

**process\_input(input\_data):**

This function preprocesses the input data before passing it to the model for prediction. It extracts features from the input data, transforms text features using featurizers loaded from files (imports\_featurizer.pkl and section\_names\_featurizer.pkl), and concatenates all features into a single sparse matrix. The processed data is returned.

**output\_fn(prediction, content\_type):**

This function formats the prediction results before sending it back as the response. It takes the prediction result (which is typically a numerical value), converts it to an integer, creates a JSON response containing the output, and returns it.

Here, we're going to discuss the prerequisite libraries.

This code initializes connections to AWS SageMaker services, S3, and defines an IAM role for accessing resources. It requires the ARN number from the created notebook instance. The code also retrieves the Amazon SageMaker URI for a specific version of the scikit-learn framework image, specifying the framework, version, Python version, and instance type.

This code Imports the previously zipped model from S3 Bucket.

This code creates a SageMaker model with specified configurations including the model's name, image URI, model artifacts URL, and execution role.

This code sets up an endpoint configuration for deploying our SageMaker model. It prepares the environment where our model will be hosted and made accessible. We give it a name and specify details like which model to use, what type of computing instance to run it on, and how many instances to start with. This configuration ensures that our model is deployed properly and ready to serve predictions.

**Creating Endpoint on Sage maker :**

This code continuously checks the status of the endpoint creation process until it's completed. It retrieves the status of the endpoint and prints it, waiting for 15 seconds between each check. Once the endpoint status changes from "Creating", it stops and prints the final status along with other endpoint details.

After the endpoint is successfully created, we are checking here the end point is working or not.

This code sends input data to a SageMaker endpoint named 'sklearn-local-ep2024-04-04- 22-38-27', expecting a JSON response, and prints the result.

**Client file explanation:**

**Now, let us configure the PE client. For the client, we will utilize the following libraries:**

After installing the necessary libraries, we will retrieve an IP address from Google Colab. This IP address will serve as our endpoint for invoking the script on the client side. We will be processing executable files (exe) as input and sending them to the API. This client-side script will be the backbone of our web application.

To streamline the process, we are leveraging a pre-trained data set for efficiency. This ensures that we do not need to train the model each time we use it, saving time and resources.

Next, let us outline the Python script for invocation. With this script, we can interact with the API, passing our processed exe files for classification.

Lastly, we'll launch a web application with Streamlit that offers an interactive interface so that people can easily engage with the model.