BBM465
Assigment-4
Oğuzhan Çetinkaya-21946003
Arda Gün Güneş-21946184

Problem Definition:

Phishing is a cybercrime in which a target or targets are contacted by email, telephone or text message by someone posing as a legitimate institution to lure individuals into providing sensitive data such as personally identifiable information, banking and credit card details, and passwords.

In our assignment, we aimed to create a threat-intelligence system for phishing detection, focusing on HTML content analysis. Our objective was to deepen our understanding of text and machine learning while emphasizing the critical role of semantics in tackling cybersecurity challenges. Using multi-lingual and mono-lingual sentence transformers, along with Google Translate, we generated 768-dimensional feature vectors for each HTML file. The next step involved building machine learning classifiers trained on these embedding vectors. This research-driven task aimed to enhance our grasp of essential concepts and improve implementation techniques in the cybersecurity domain.

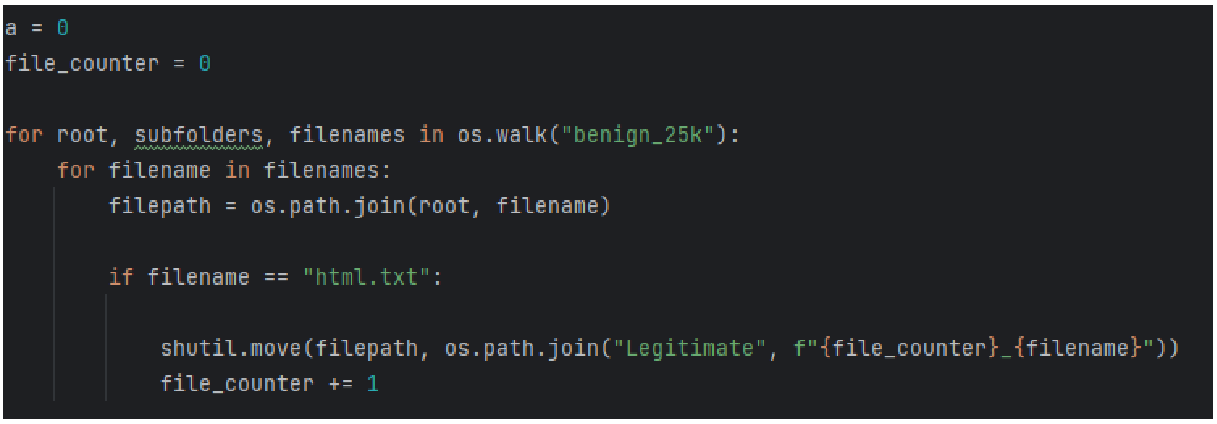
Some Important Notes to Run the Program:

The server.py can be run without a problem. However, if individual scripts are going to be tested with Legitimate folder and Phishing folder. The locations of them should be one directory above of the program: **../Legitimate** **../Phishing**

data_prepare.py:

The data_prepare.py is responsible for organizing the HTML files into two separate folders, one for legitimate web pages and one for phishing web pages.

- It defines two folder paths, one for legitimate pages (**path_legitimate**) and one for phishing pages (**path_phishing**).
- It checks if these folders exist. If not, it creates them.



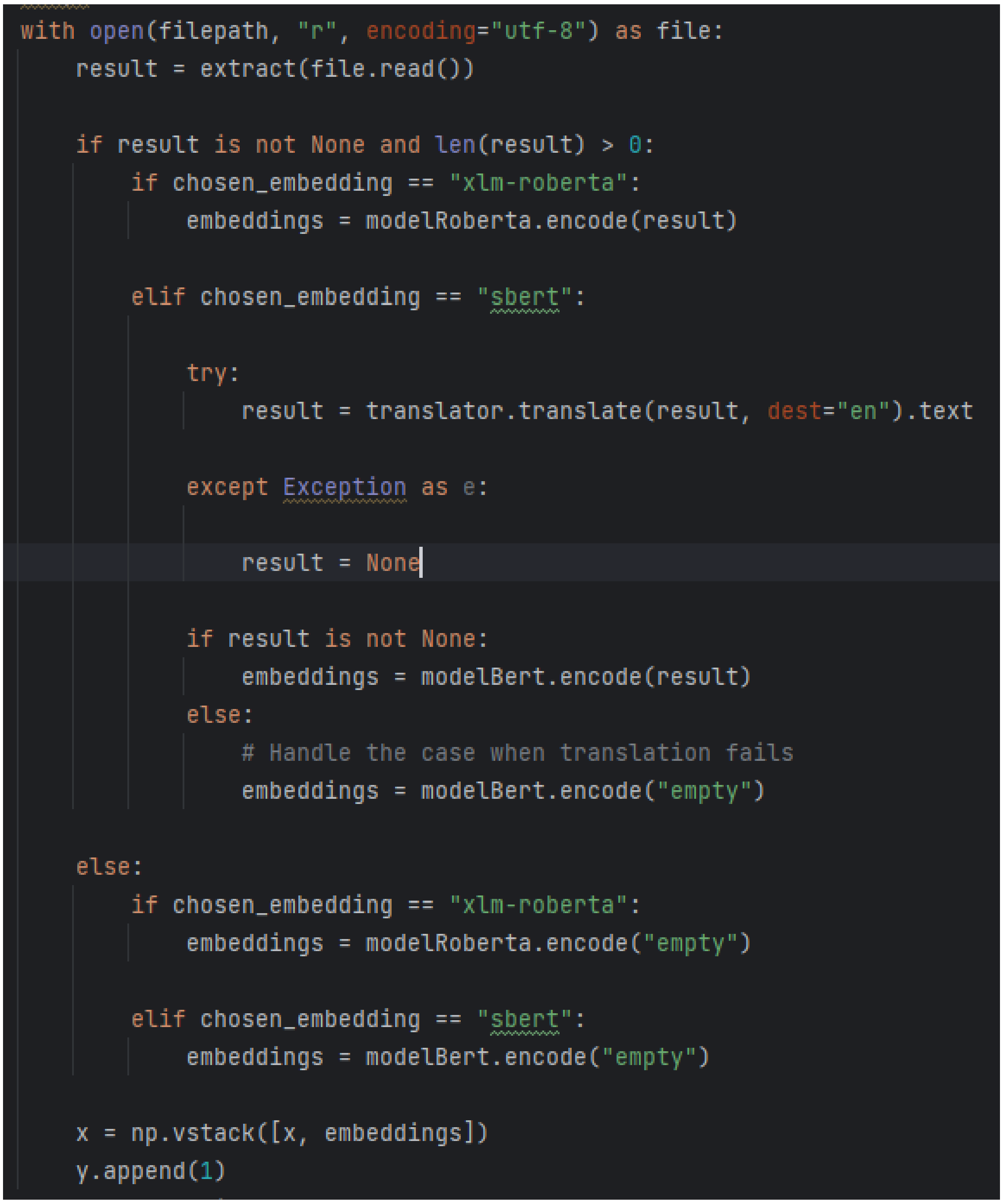
```python
a = 0
file_counter = 0

for root, subfolders, filenames in os.walk("benign_25k"):
    for filename in filenames:
        filepath = os.path.join(root, filename)

        if filename == "html.txt":

            shutil.move(filepath, os.path.join("Legitimate", f"{file_counter}_{filename}"))
            file_counter += 1
```

- It iterates through the files in the "benign_25k" directory.
- If a file is named "html.txt," it moves that file to the "Legitimate" folder with a new name that includes a file counter.
- The same process applies for the “phishing_sample30k” directory.

prepare_embedding.py:

The prepare_embedding.py is responsible for processing HTML files, extracting text, translating it if necessary, and then generating embeddings using Sentence Transformers.

- We used trafilatura for HTML extraction, Sentence Transformers for generating embeddings, numpy for array manipulation, os for file operations, time for measuring processing time, torch for GPU support, pickle for data serialization, sys for accessing command line arguments, and Translator from googletrans for translation
- We labelled legitimate samples as 1 and phishing samples as 0:
- The function **def fill_legitimate_embeddings** takes two parameters x and y, which are numpy arrays for feature vectors and labels, respectively.
- It iterates over the files in the "../Legitimate" directory using os.walk.
- For each file, it opens and reads the content using with open.
- The text content is extracted using the extract function.
- Depending on the chosen embedding method (chosen_embedding), it encodes the text using either XLM-Roberta or SBERT.
- If translation is required for SBERT, it attempts to translate the text to English using Google Translate.
- The resulting embeddings are stacked vertically into the array x, and the label '1' (for legitimate) is appended to the array y.



```
with open(filepath, "r", encoding="utf-8") as file:
    result = extract(file.read())

    if result is not None and len(result) > 0:
        if chosen_embedding == "xlm-roberta":
            embeddings = modelRoberta.encode(result)

        elif chosen_embedding == "sbert":

            try:
                result = translator.translate(result, dest="en").text

            except Exception as e:

                result = None

            if result is not None:
                embeddings = modelBert.encode(result)
            else:
                # Handle the case when translation fails
                embeddings = modelBert.encode("empty")

    else:
        if chosen_embedding == "xlm-roberta":
            embeddings = modelRoberta.encode("empty")

        elif chosen_embedding == "sbert":
            embeddings = modelBert.encode("empty")

    x = np.vstack([x, embeddings])
    y.append(1)
```

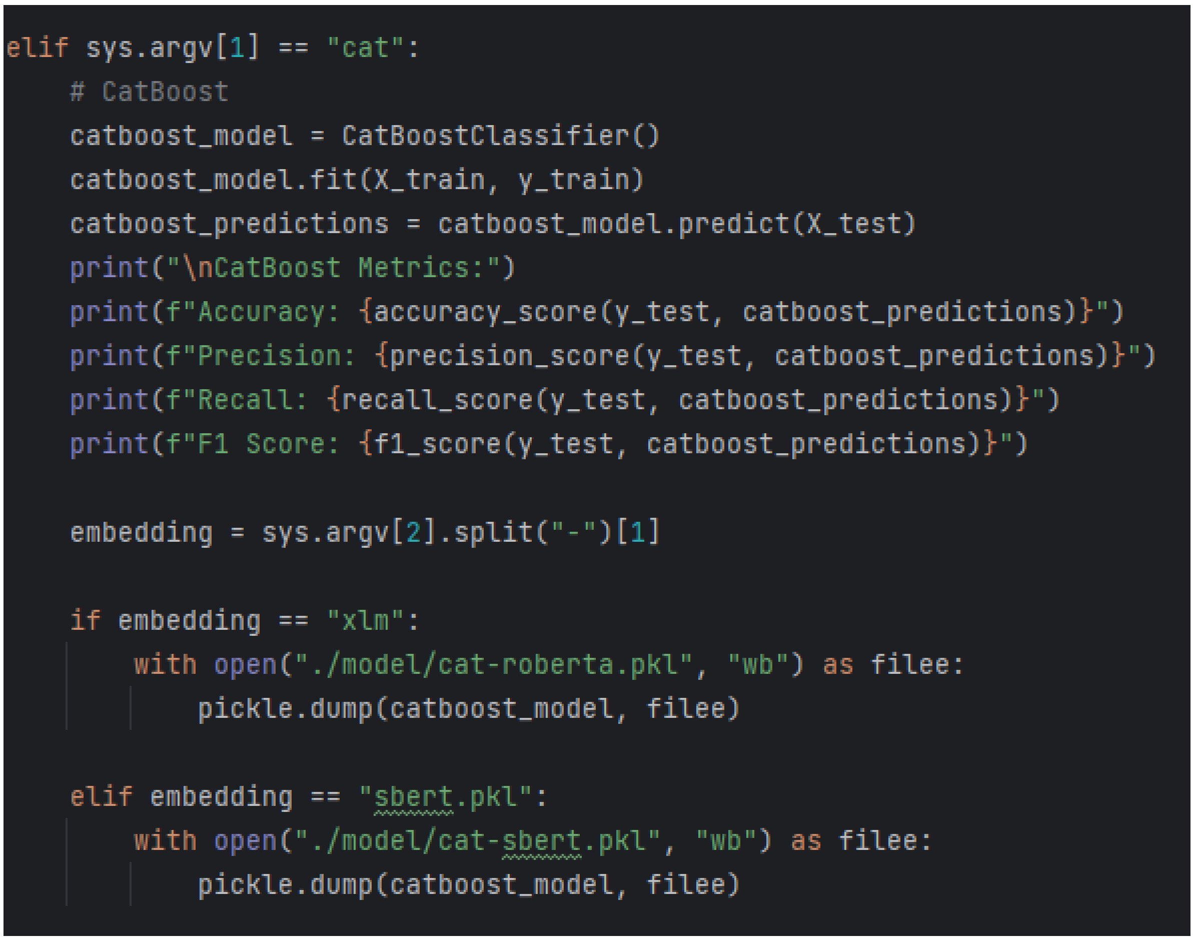
- The function prints the elapsed time every 1000 iterations for monitoring.
- The final x and y arrays are returned.
- The same process is applied to **def fill_phishing_embeddings** with the exception of class label 0.

model_build.py:

The model_build.py script is responsible for building and training machine learning models using the embeddings generated from HTML content.

- **Command-line Arguments**:
- It expects two command-line arguments (sys.argv[1] and sys.argv[2]), which are provided when the script is executed in the terminal.
- sys.argv[1]: Specifies the type of machine learning model to train ("xgb" for XGBoost or "cat" for CatBoost).
- sys.argv[2]: Specifies the path to a pickle file containing embedings, including feature vectors (X) and labels (Y).
- **Data Loading**:
- It opens the specified pickle file using the second command-line argument (sys.argv[2]).
- The data loaded includes feature vectors (X) and labels (Y), which were prepared in a previous step.
- **Data Splitting**:
- The loaded data is split into training and testing sets using the train_test_split function from scikit-learn.
- 80% of the data is used for training (X_train, y_train), and 20% is used for testing (X_test, y_test).
- **Model Training**:
- Depending on the value of the first command-line argument (sys.argv[1]):

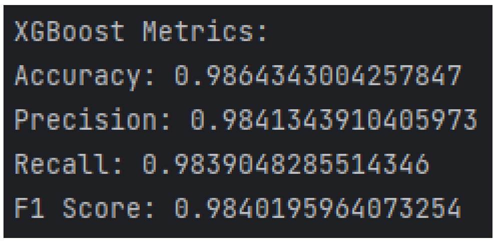
- If it is "xgb," an XGBoost model is created (xgb_model) and trained on the training data.
- If it is "cat," a CatBoost model is created (catboost_model) and trained on the training data.
- It chooses a specified embedding according to the sys.argv[2].
- It then prints metrics such as accuracy, precision, recall, and F1 score for the trained model on the testing set.



```python
elif sys.argv[1] == "cat":
    # CatBoost
    catboost_model = CatBoostClassifier()
    catboost_model.fit(X_train, y_train)
    catboost_predictions = catboost_model.predict(X_test)
    print("\nCatBoost Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, catboost_predictions)}")
    print(f"Precision: {precision_score(y_test, catboost_predictions)}")
    print(f"Recall: {recall_score(y_test, catboost_predictions)}")
    print(f"F1 Score: {f1_score(y_test, catboost_predictions)}")

    embedding = sys.argv[2].split("-")[1]

    if embedding == "xlm":
        with open("./model/cat-roberta.pkl", "wb") as filee:
            pickle.dump(catboost_model, filee)

    elif embedding == "sbert.pkl":
        with open("./model/cat-sbert.pkl", "wb") as filee:
            pickle.dump(catboost_model, filee)
```

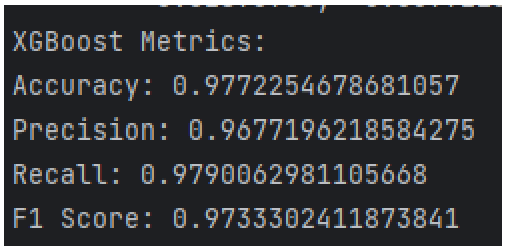
Here are the metrics:

XGB-Roberta:



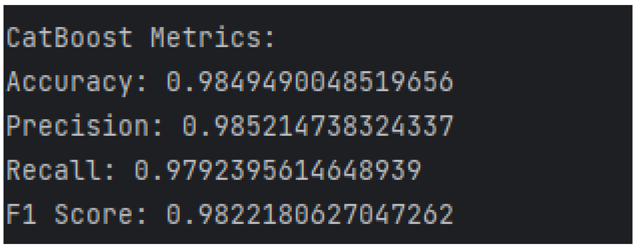
```
XGBoost Metrics:
Accuracy: 0.9864343004257847
Precision: 0.9841343910405973
Recall: 0.9839048285514346
F1 Score: 0.9840195964073254
```

- **XGB-SBERT**:



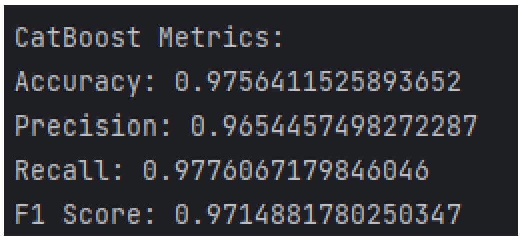
```
XGBoost Metrics:
Accuracy: 0.9772254678681057
Precision: 0.9677196218584275
Recall: 0.9790062981105668
F1 Score: 0.9733302411873841
```

- **CAT-ROBERTA**:



```
CatBoost Metrics:
Accuracy: 0.9849490048519656
Precision: 0.985214738324337
Recall: 0.9792395614648939
F1 Score: 0.9822180627047262
```

- **CAT-SBERT**:



```
CatBoost Metrics:
Accuracy: 0.9756411525893652
Precision: 0.9654457498272287
Recall: 0.9776067179846046
F1 Score: 0.9714881780250347
```

As can be seen from the above results. **XGBoost** model with **xlm-roberta** sentence tranformer gives the best result in terms of **accuracy, recall** and **f1 score**. So we choosed **XGBoost** model with sentence transformer **xlm-roberta.**

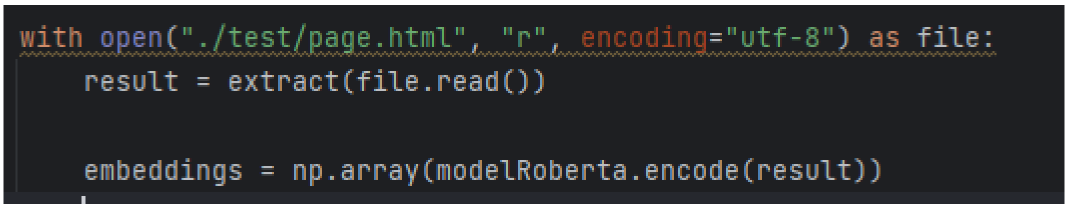
xlm-roberta had a better performance in compared to **sbert** sentence transformer. The reason for that is while **xlm-roberta** can easily detect and translate texts, **sbert** needs a third-party translation library. It depends on the **googletrans**'s performance. Sometimes **googletrans** raises an error due to some **IP problems**. Because of that some of the texts couldn't be translated. And this caused **sbert**'s accuracy to fall down.

Although **XGBoost** showed a slightly better performance, the difference between **XGBoost** and **Cat** wasn't significant.

server.py:

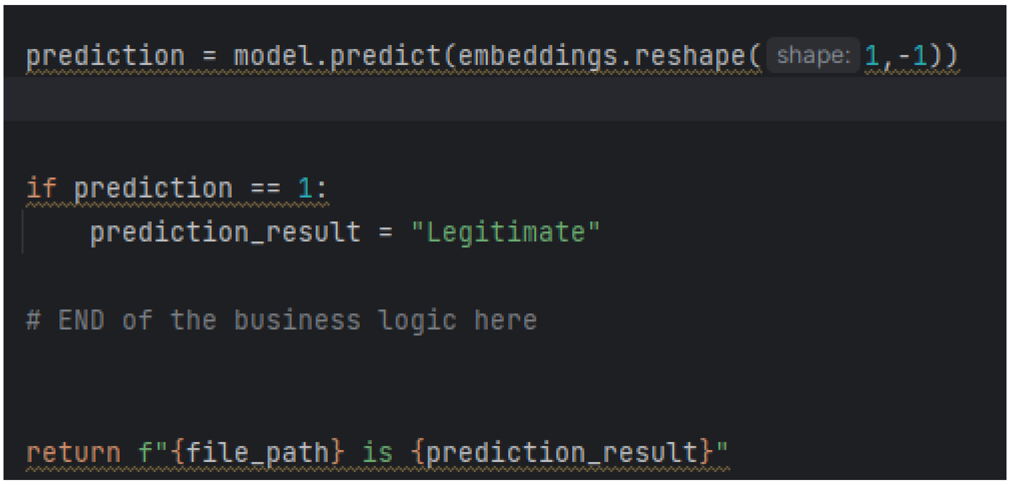
This script is a Flask web application that serves as an interface for making predictions using a pre-trained XGBoost model for phishing detection.

1. **Prediction Logic:**
   - When a file is uploaded through the '/predict' route, it checks if the file is an HTML file.
   - The uploaded HTML file is saved to a temporary location in the 'test/' directory.
   - The content of the HTML file is then extracted using **trafilatura**.
   - The extracted content is passed through the Sentence Transformer model (**modelRoberta**) to obtain embeddings.



```
with open("./test/page.html", "r", encoding="utf-8") as file:
    result = extract(file.read())

    embeddings = np.array(modelRoberta.encode(result))
```

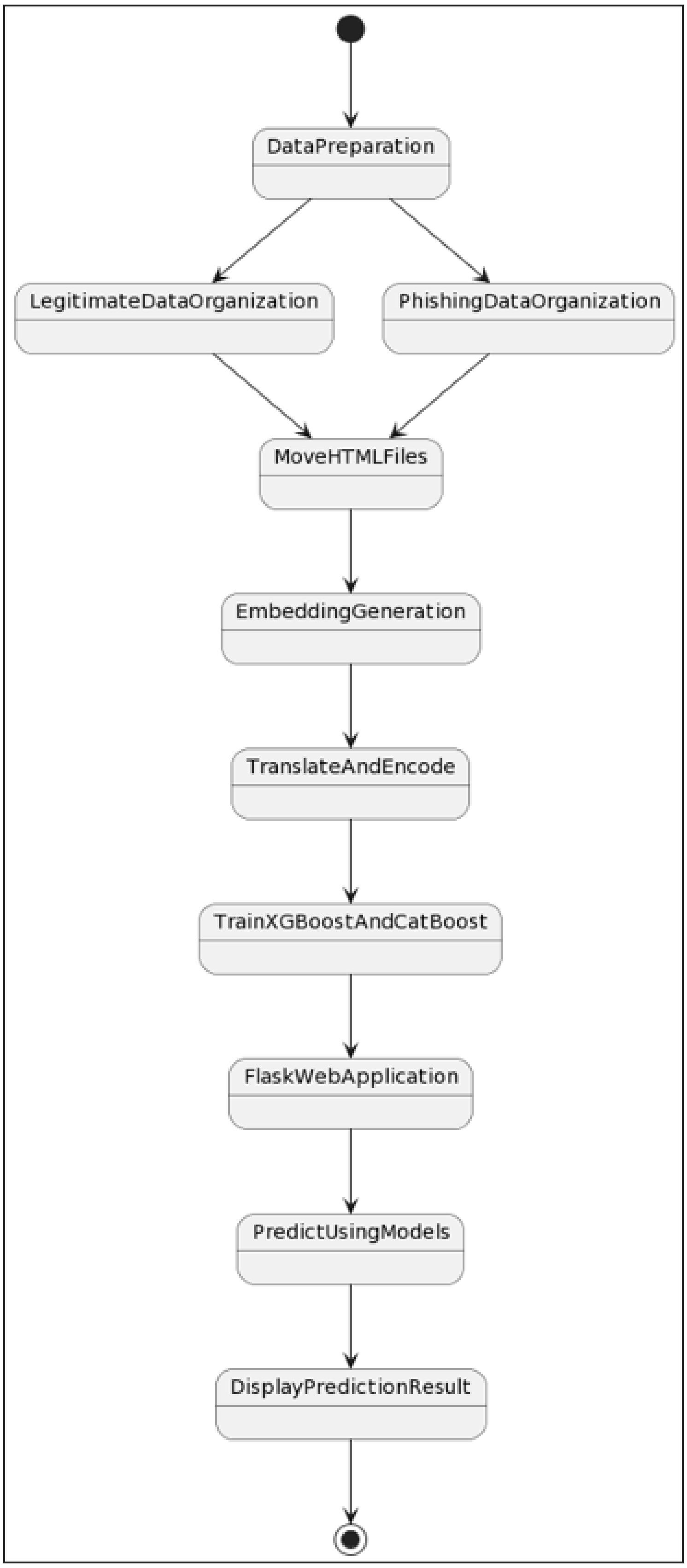
   - The embeddings are then reshaped and used to make a prediction with the pre-trained XGBoost model (**model**).
   - If the prediction is equal to 1, the result is set as "Legitimate"; otherwise, it is set as "Phishing".



```
prediction = model.predict(embeddings.reshape( shape: 1,-1))


if prediction == 1:
    prediction_result = "Legitimate"

# END of the business logic here


return f"{file_path} is {prediction_result}"
```

2. **Response:**
    - The response of the prediction, indicating whether the uploaded HTML file is predicted as "Legitimate" or "Phishing," is returned.

Flow Chart:


DataPreparation
LegitimateDataOrganization
PhishingDataOrganization
MoveHTMLFiles
EmbeddingGeneration
TranslateAndEncode
TrainXGBoostAndCatBoost
FlaskWebApplication
PredictUsingModels
DisplayPredictionResult

Conclusion:

In conclusion, our assignment aimed at developing a threat-intelligence system for phishing detection, concentrating on HTML content analysis. Throughout this endeavor, we navigated the intricate landscape of cybersecurity by leveraging multi-lingual and mono-lingual sentence transformers, along with Google Translate, to generate 768-dimensional feature vectors for each HTML file. Subsequently, machine learning classifiers, including XGBoost and CatBoost, were meticulously trained on these embeddings. This research-driven task not only deepened our understanding of text and machine learning but also underscored the pivotal role of semantics in addressing cybersecurity challenges. The amalgamation of HTML content analysis, advanced embeddings, and machine learning classifiers presents a comprehensive approach to enhancing cyber-threat detection mechanisms. Our journey not only enriched our comprehension of fundamental concepts but also honed our practical skills in implementing effective real-world cybersecurity solutions.