INTRODUCTION

The goal of this assignment is to develop a system to distinguish legitimate and phishing websites using machine learning and natural language processing (NLP) concepts. There were several main steps to train the model. These steps can be explained as follows:

1. **Data Preparation**: We have gigabytes of data including screenshots, html files and some other data from a website. Firstly, we must extract meaningful and useful data.
2. **Feature Extraction**: In this part, we will parse extracted HTML data with Trafilatura and we benefit from pre-trained NLP model to produce vector representations. We will use both of mono-language model and multi-language model.
3. **Learning Phase**: Finally, by using the 768-dimensional feature vector data of each HTML file, which we obtain previous part, we train a model with these ML algorithms: **XG-Boost, CatBoost**.
4. **Deployment Phase**: In addition, there is a basic website to test model with different sample websites.

Note: Intel(R) Core(TM) i5-6300HQ CPU @2.30GHz as CPU, and **NVIDIA GeForce GTX 960M** as GPU used for project

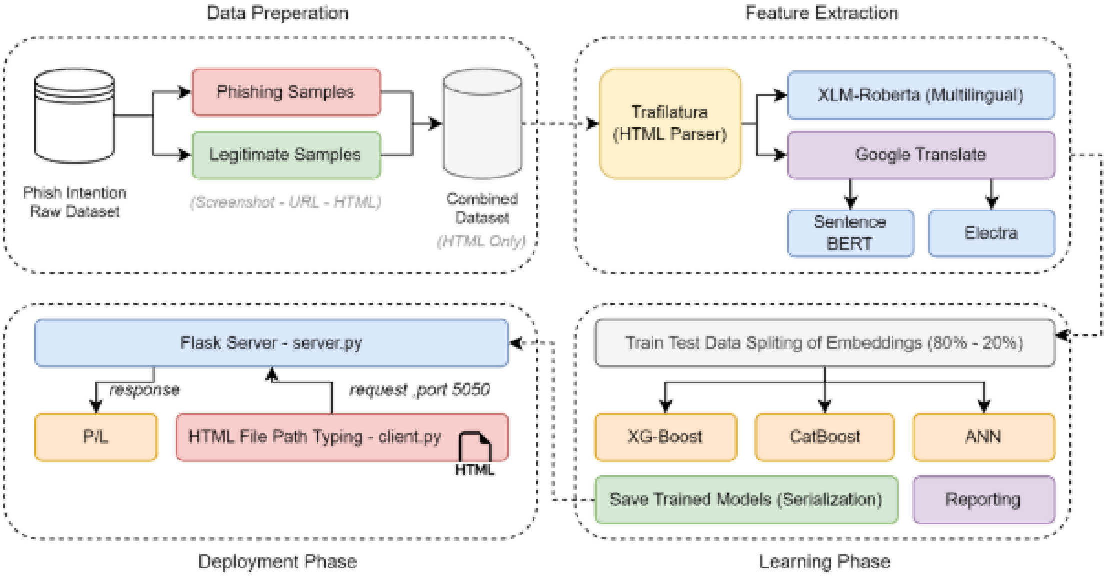


Figure 1: System Workflow

Data Preparation and Cleaning

We have a data set of around 45 GB containing nearly 57,945 website data. The data is divided into three parts:

- **legitimate websites**: 25400 sample
- **phishing websites:** 29496 sample
- **deceptive legitimacy**: 3049 sample

The authentic websites that frequently link to social media platforms like YouTube and Twitter or that enable sign-in and registration using Facebook and Google are examples of misleading legitimacy. The important point is that because these websites feature logos of wellknown businesses and brands, visual detection algorithms frequently mistakenly identify them as phishing. In this project, we will use misleading examples together with legitimate examples.

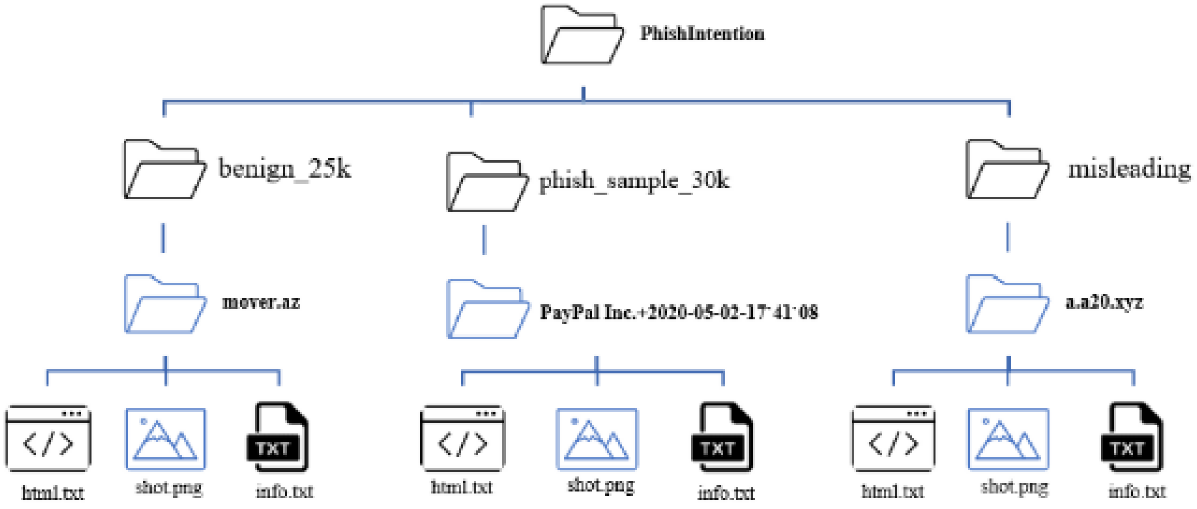
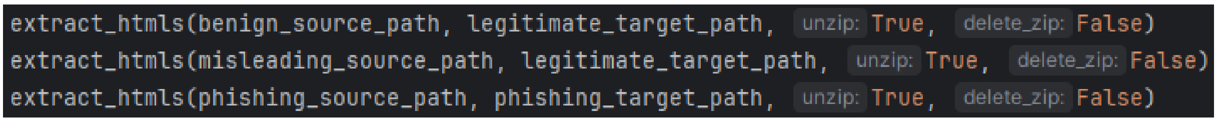


Figure 2: The folder hierarchy of the PhishIntention dataset

Before extracting HTML files, we need to unzip the files if dataset have not been extracted from zip. So, we add a two more options to extractor: unzip and delete_zip. If unzip is true, the dataset is unzipped, and if delete_zip is true, the huge zip file will be removed after extraction operation.



```
extract_htmls(benign_source_path, legitimate_target_path, unzip: True, delete_zip: False)
extract_htmls(misleading_source_path, legitimate_target_path, unzip: True, delete_zip: False)
extract_htmls(phishing_source_path, phishing_target_path, unzip: True, delete_zip: False)
```

After uncompressing, we find all HTML files and save them to the new path. During these operations, we change all non-alphanumeric characters with "_", because these characters sometimes cause **OS Error**. We will give detail in Problems Occured part.

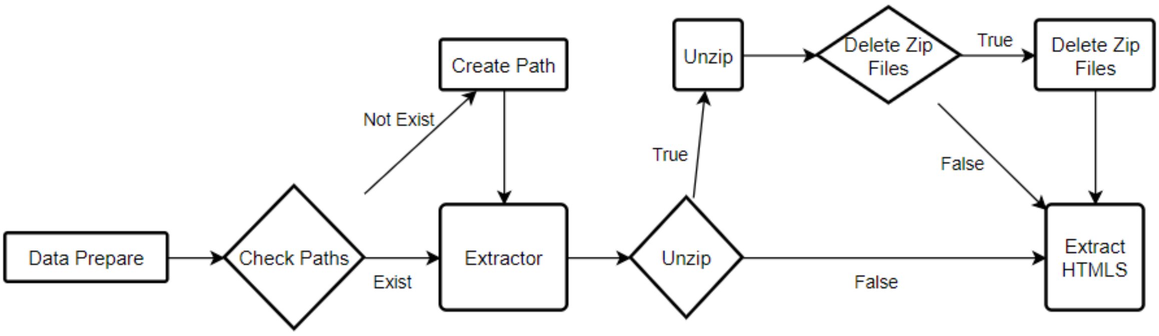


Figure 3: Data Prepare Flow Chart

Feature Extraction

On this step, we generated embeddings with using three algorithm. **Xlm-roberta, Sbert** and **Electra**. An embedding is simply vector representation of any text with 768 element. Xlm-roberta can generate embedding independent from its language. But for other transformers, that's not the case. Sbert and Electra can generate embedding from only English text. That's why when generating embedding, if the transformer is xlm-roberta, the text hasn't send to translator api.

Before translation operation for mono-language models, we parse HTML files to parse website contents. The content of HTML files is usually spread into different divisions (in different DIV, TD, and TR tags). We should focus on site contents instead of CSS, or tags. We use **Trafilatura** parser for this step.

Trafilatura: It is Python package and command-line tool which seamlessly downloads, parses, and scrapes web page data: it can extract metadata, main body text and comments while preserving part of the text formatting and page structure. The output can be converted to different formats.

To make the translation, googletrans library used with version 4.0.0-rc1. But after translating some content, the translator stopped translating text. The main reason of this error is Error 429 (Too Many Requests). That's why embedding results of xlm-roberta will be used for machine learning since it don't require translate to english and it has high accuracy score.

After translation, generated 768 dimensional vectors which are embeddings. To distinguish legitimate and phising embeddings, there were added 769th attribute to embeddings. For legitimate, this value is 0, for phishing, th value is 1. The reason to do this is value to be able to save embedding informations to one file.

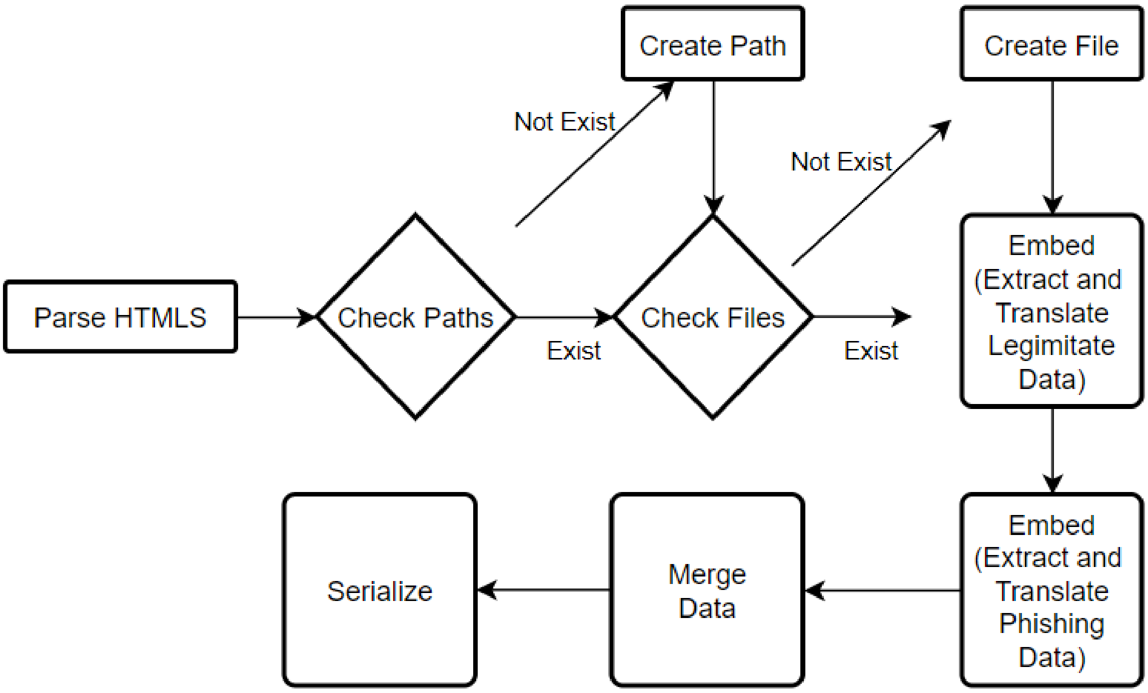


Figure 4: Feature Extension Flow Chart

Learning Phase

To check the correctness of model, there exists some formules.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$Precision = \frac{TP}{TP + FP} \times 100\%$$

$$Recall = \frac{TP}{TP + FN} \times 100\%$$

$$F1\ Score = \frac{2\ \times Precision\ \times Recall}{Precision + Recall} \times 100\%$$

Performance metrics for XGBoost machine learning algorithm:

- Xlm-roberta
  - Accuracy: 98.28%
  - Precision: 98.26%
  - Recall: 98.80%
  - F1 score: 98.53%
  - Confusion matrix:

Act.\ Pred.	0	1
0	4034	101
1	69	5695

- Sbert
  - Accuracy: 98.15%
  - Precision: 98.13%
  - Recall: 98.81%
  - F1 score: 98.47%
  - Confusion matrix:

Act.\ Pred.	0	1
0	2910	85
1	54	4468

- Electra
  - Accuracy: 97.63%
  - Precision: 97.74%
  - Recall: 98.35%
  - F1 score: 98.05%
  - Confusion matrix:

Act.\ Pred.	0	1
0	2829	102
1	74	4421

Performance metrics for CatBoost machine learning algorithm:

- Xlm-roberta
  - Accuracy: 97.65%
  - Precision: 97.74%
  - Recall: 98.23%
  - F1 score: 97.98%
  - Confusion matrix:

Act.\ Pred.	0	1
0	4004	131
1	102	5662

- Sbert
    - Accuracy: 97.98%
    - Precision: 97.81%
    - Recall: 98.85%
    - F1 score: 98.33%
    - Confusion matrix:

Act.\ Pred.	0	1
0	2895	100
1	52	4470

- Electra
    - Accuracy: 96.96%
    - Precision: 96.88%
    - Recall: 98.13%
    - F1 score: 97.50%
    - Confusion matrix:

Act.\ Pred.	0	1
0	2789	142
1	84	4411

As it can be seen from performance metrics above, XGBoost has slightly better performance than catboost. Also when machine learning performed on XGB with CPU, it took 3 minutes. But even though GPU used with cat algorithm, this took 10 minutes with 1000 iteration.

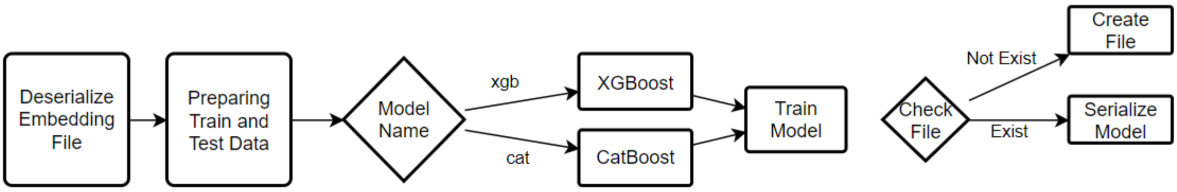


Figure 5: Learning Phase Flow Chart

Deployment Phase

After the models are generated with using xgb and cat, these models are serialized with using pkl files anda re ready for deployment to server.py. There has generated six different model with using xgb and cat boost and different embedding algorithms. Since xlm-roberta_xgb_based_model.pkl gives the best performance metrics, this model will be used on server.py.

Problems Occured While Development

File Naming Problems

Since we processed real world data, there were some data caused problem, even because of their names. Some file names contains this character ” `” which causes no problem on some Windows computer.

To open and read the files, we used open(), a builtin function of Python for file IO operations. The functions above worked without problem on Linux computers. But on some Windows computers, trying to read these files throwed this error:

```
OSError: [Errno 22] Invalid argument:
'PhishIntention/phish_sample_30k/Charles Schwab+2020-08-17-
13`04`08/html.txt'
```

To solve this error, we first opened all files with latin-1 encoding instead of utf-8. It seemed to work on first glance, but was returning corrupted results when trying to read to read Arabic, Kiril alphabet etc. That’s why we changed file names with new versions of them don’t contain “`” character with using function below contains regex.

```
def change_directory_names(path):
    files = os.listdir(path)
    for i in files:
        old_path = path + "/" + i
        new_path = path + "/" + re.sub(r'[^0-9a-zA-Z]+', '_',
i)
        if not os.path.exists(new_path):
            os.rename(old_path, new_path)
```

File Reading Problems

After this step, we were ready for file reading operations. But for some files, using open() function raised this exception below:

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in
position 44555: invalid continuation byte
```

That’s why we changed our file opening function with

```
codecs.open(html_file_path, 'r', encoding='utf-8',
errors='ignore')
```

The function ignores invalid bytes rather than raising exception, which solved our problem and enabled us reading files without any problem.

Problems Occured when CUDA GPU Accelereation With Electra

On embedding with electra, we used cuda for GPU acceleration like every other embedding algorithm. When we send every every parsed html content to other embedding functions, no problem occured. But when we tried same approach on electra algorithm, it allocated whole memory of GPU and raised this exception:

```
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to
allocate 150.00 MiB. GPU 0 has total capacity of 4.00 GiB od
whitch 0 bytes is free. Of the allocated memory 10.82 GiB is
allocated by PyTorch ...
```

Right after the exception thrown, the computer locked entirely and even moving the mouse cursor became impossible. That's why we needed to hard reset the computer. After that, we decdided to send html contents as chunks. When 100 html page content per chunk processed, there had no problem occured.

Googletrans Module Problems

When used **Pool()** object from multiprocessing library or builtin **map()** function to send translate requests with an efficient manner, this exception

AttributeError: 'Translator' object has no attribute 'raise_Exception'

occured. But when we used **ThreadPoolExecutor(max_workers=10)** from concurrent.futures library, no problem occured. With this library, we can also arrange the maximum worker amount. Translating 1000 web content with sending request to googletrans api took around 150 second for 4 max_workers. When we increased ths number to 10, the time also decreased to 100 second. After that point, increasing max_worker had no significant impact on executing time.

Then we realized that even though no exception occured after sending 1000 content transalate request to API, only half of the results returned. It seems like exceeding request limit cauesed this problem. That's why we decided to design a caching mechanism. To prevent sending multiple translate requests for same file, we changed the translater function to save translated website contents under Legitimate_Extracted_Texts_Translated and Phishing_Extracted_Texts_Translated folders. That means if a website inside the dataset s translated once, the result will be saved under these folders and when needed to translate again, the request won't be sent again

We also observed that translating with ThreadPoolExecutor is faster than Pool.

Translating 100 content took 34 second for Pool(processes=4) and 13 second for ThreadPoolExecutor(max_workers=10). That's why we used ThreadPoolExecutor to handle translating proces. Because it contains sending request to server and waiting fort he response was most of the job.

RESOURCES

[1] PhishIntention - Experiment-structure. (n.d.). PhishIntention - Experiment-structure. Retrieved January 5, 2024, from https://sites.google.com/view/phishintention/experiment-structure

[2] Trafilatura. (2020, June 2). PyPI. Retrieved January 5, 2024, from https://pypi.org/project/trafilatura/0.5.0/

[3] xlm-roberta-base · Hugging Face. (n.d.). Xlm-roberta-base · Hugging Face. Retrieved January 5, 2024, from https://huggingface.co/xlm-roberta-base

[4] SentenceTransformers Documentation &mdash; Sentence-Transformers documentation. (n.d.). SentenceTransformers Documentation &Mdash; Sentence-Transformers Documentation. Retrieved January 5, 2024, from https://www.sbert.net/

[5] ELECTRA. (n.d.). ELECTRA. Retrieved January 5, 2024, from https://huggingface.co/docs/transformers/model_doc/electra