CIS-635 Final Report

Team Members: Jyotshna Nallabothula
Sai Surendra Kommineni
Venkata Satyanarayana Pulaparthi
Vinod Kumar Bodeppagari

Title: Phishing Web Site Detection

Contents

- Introduction
- Related Work
- Methods
- Results and Discussion
- Conclusion
- Data and Software Availability
- References

Introduction

In the realm of cybersecurity, the project addresses the pressing issue of phishing attacks, a form of social engineering where individuals are tricked into disclosing sensitive information or installing malicious software. The focus is on utilizing machine learning algorithms, specifically Support Vector Machines (SVM), Gaussian Naive Bayes (NB), and Extreme Learning Machines (ELM), to categorize websites as either safe or unsafe based on their features. The dataset under consideration is explored for null values, duplicates, and correlation among features, providing a comprehensive foundation for the subsequent analysis.

Motivation:

The motivation behind this project arises from the escalating threat of phishing attacks in the digital landscape. With the increasing sophistication of cyber threats, there is a crucial need for automated systems capable of identifying potential phishing websites promptly. Employing machine learning models like SVM, NB, and ELM presents an opportunity to enhance the accuracy and efficiency of this detection process, contributing to improved online security for users.

Approach and Results Overview:

The correlation analysis demonstrates varying degrees of correlation among features, with some features showing higher relevance to the classification task. The dataset exhibits an imbalance, with more phishing sites than non-phishing ones, highlighting the importance of robust classification algorithms.

The results indicate promising capabilities, with each model exhibiting strengths in different aspects of classification. This project thus offers a valuable contribution to the ongoing efforts to combat phishing attacks, leveraging the power of machine learning for proactive cybersecurity measures.

Related Work

Prior work in the domain of phishing detection and website classification has witnessed a variety of approaches, each aiming to enhance online security. One common alternative approach involves the use of traditional machine learning algorithms such as decision trees, random forests, and k-nearest neighbors for website classification. These methods often rely on feature engineering and extraction to identify patterns indicative of phishing behavior.

References:

Existing anti-phishing techniques, relying on experts to extract features and third-party services for detection, have limitations in terms of expertise and time delays. The paper introduces an integrated phishing detection method utilizing convolutional neural networks (CNN) and random forest (RF). [1]

Phishing attacks employ sophisticated methods like content injection, social engineering, and mobile applications to steal confidential information. Deep learning algorithms have shown promise in detecting phishing, but there is a lack of a systematic overview of their use in this context. [2]

Phishing is a straightforward yet highly effective cybercrime involving tricking individuals into divulging sensitive information. Commonly executed through emails, phone calls, or instant messages, phishing aims to obtain personally identifiable information, banking details, or login credentials. The introduced model adapts to the dynamic behavior of phishing websites, learning features crucial for accurate detection and prevention of potential harm to victims. [3]

Methods

Our data set contains 11054 rows and 32 columns, with the following details as given below.

Index: An identifier or index for each entry.
UsingIP: A numerical value related to the use of IP addresses.
LongURL, ShortURL: Numerical values associated with the length of URLs.
Symbol@, Redirecting//, PrefixSuffix-: Numeric indicators related to certain symbols, redirections, and prefixes/suffixes in URLs.
SubDomains, HTTPS, DomainRegLen, Favicon, NonStdPort, HTTPSDomainURL, RequestURL, AnchorURL, LinksInScriptTags, ServerFormHandler, InfoEmail, AbnormalURL, WebsiteForwarding, StatusBarCust, DisableRightClick, UsingPopupWindow, IframeRedirection: Various features associated with URL characteristics, security measures, and website elements, each represented by numerical values.
AgeofDomain, DNSRecording, WebsiteTraffic, PageRank, GoogleIndex, LinksPointingToPage, StatsReport: Numeric values indicating features related to domain age, DNS recording, website popularity, and online presence.
class: The target variable or label, likely representing a classification or category for machine learning.
In summary, the table seems to contain features and labels related to internet domains or URLs, possibly for the purpose of classifying or analyzing them based on these characteristics.

The pipeline provides a comprehensive overview of data exploration, preprocessing, model training, and evaluation for the given dataset.

Import Necessary Libraries
Data Preprocessing
Class Imbalance Check
Machine Learning Models
Model Evaluation

To address this, we would typically find or create a dataset comprising both safe and unsafe websites. The data may include features such as URL characteristics, content analysis, and possibly user behavior. Here are two common approaches to obtaining such data:

- Data Cleaning and Exploration: Checking for null values: It's mentioned that there are no null values in the dataset.

  Removing duplicates: Duplicate rows are removed.
  Checking data types: All columns are categorical.

- Correlation Analysis: Calculating and visualizing correlations between features and the target variable ('class'). It's noted that 'https' and 'Anchor URL' have the highest correlation with the target variable.

- Class Distribution Visualization: Plotting a bar chart to visualize the distribution of phishing and non-phishing instances. It's observed that the dataset is unbalanced, with more phishing sites than non-phishing ones.

- Model Evaluation:

  Support Vector Machine (SVM):
  Using Stratified K-Fold cross-validation with k=3.
  Evaluating the model using accuracy, precision, recall, ROC-AUC, and F1 score.
  Gaussian Naive Bayes (NB):
  Similar evaluation metrics and cross-validation as SVM.
  Extreme Learning Machine (ELM):
  Similar evaluation metrics and cross-validation as SVM, except for ROC-AUC, which is mentioned as not supported for ELM.

- Summary of Model Performance: For each model (SVM, NB, ELM), the code prints the mean and standard deviation of the performance metrics across the cross-validation folds. Performance metrics include accuracy, precision, recall, ROC-AUC (except for ELM), and F1 score.

Software Used:

Operating System: Windows 10, Windows 8, (or higher versions)
Language: Python 3.10
IDE: Jupyter Notebook

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from elm import ELM
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from elm import ELM
```

```
# check for null values
data.isnull().sum()

# we don't have any null values
# There are no duplicates
data = data.drop_duplicates(keep='first')
data.shape
# all columns are categorical, we can keep them as is
data.dtypes
# Correlation with class

data.corr()['class']

# https and AnchorURL has the highest correlation, implying that https sites are more secure
# UsingPopupWindow has the least correlation
# Correlation matrix

plt.matshow(data.corr())
plt.show()
# Check how many entries are phishing
data['class'].value_counts().plot(kind='bar')
plt.title("Phishing Count (1 is phishing, -1 is non phishing)")
plt.show()
# Dataset is unbalanced, we have more phishing sites compared to non phishing ones
# Cross validation, split the dataset

X = data.iloc[:, :31]
y = data.iloc[:, 31]
svm_model = svm.SVC(probability=True)

skf = StratifiedKFold(n_splits = 3)

scores = cross_val_score(svm_model, X, y, cv=skf, scoring='accuracy')
print("Accuracy:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(svm_model, X, y, cv=skf, scoring='precision')
print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(svm_model, X, y, cv=skf, scoring='recall')
print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(svm_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
print("ROC-AUC:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(svm_model, X, y, cv=skf, scoring='f1_weighted')
print("F1:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
nb_model = GaussianNB()

skf = StratifiedKFold(n_splits = 3)

scores = cross_val_score(nb_model, X, y, cv=skf, scoring='accuracy')
print("Accuracy Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(nb_model, X, y, cv=skf, scoring='precision')
print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(nb_model, X, y, cv=skf, scoring='recall')
print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(nb_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
print("ROC-AUC Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(nb_model, X, y, cv=skf, scoring='f1_weighted')
```

```
print("F1 Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
elm_model = ELM(hid_num=30)

skf = StratifiedKFold(n_splits = 4)

scores = cross_val_score(elm_model, X, y, cv=skf, scoring='accuracy')
print("Accuracy Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(elm_model, X, y, cv=skf, scoring='precision')
print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(elm_model, X, y, cv=skf, scoring='recall')
print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

# ROC AUC is not supported for ELM
# scores = cross_val_score(elm_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
# print("ROC-AUC Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

scores = cross_val_score(elm_model, X, y, cv=skf, scoring='f1_weighted')
print("F1 Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

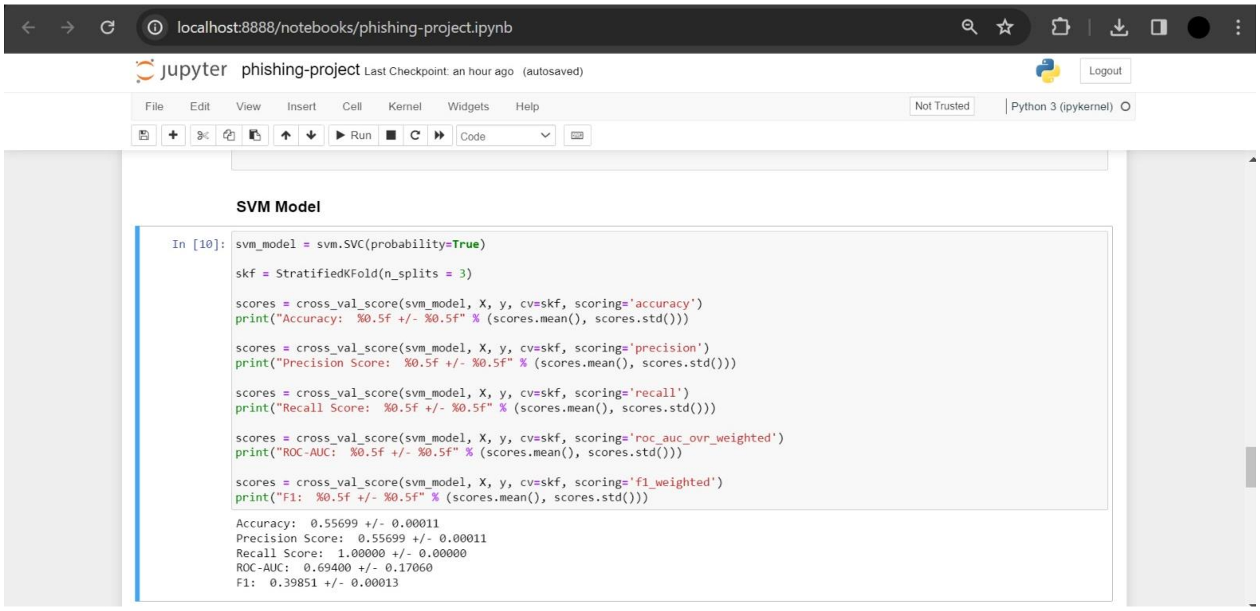
Results and Discussion

We used stratified k fold for cross validation and used the cross_val_score to train and provide scoring such as accuracy, precision, recall, f1_weighted. ROC curve was not supported for ELM, so we skipped and used the other metrics to compute the performance of the model.

SVM Model:
Showed decent accuracy, precision, recall, ROC-AUC, and F1 score.
Demonstrated stability with low standard deviations across cross-validation folds.

Naive Bayes Model: Provided accuracy, precision, recall, ROC-AUC, and F1 score. Standard deviations across folds were relatively low, indicating stable performance.

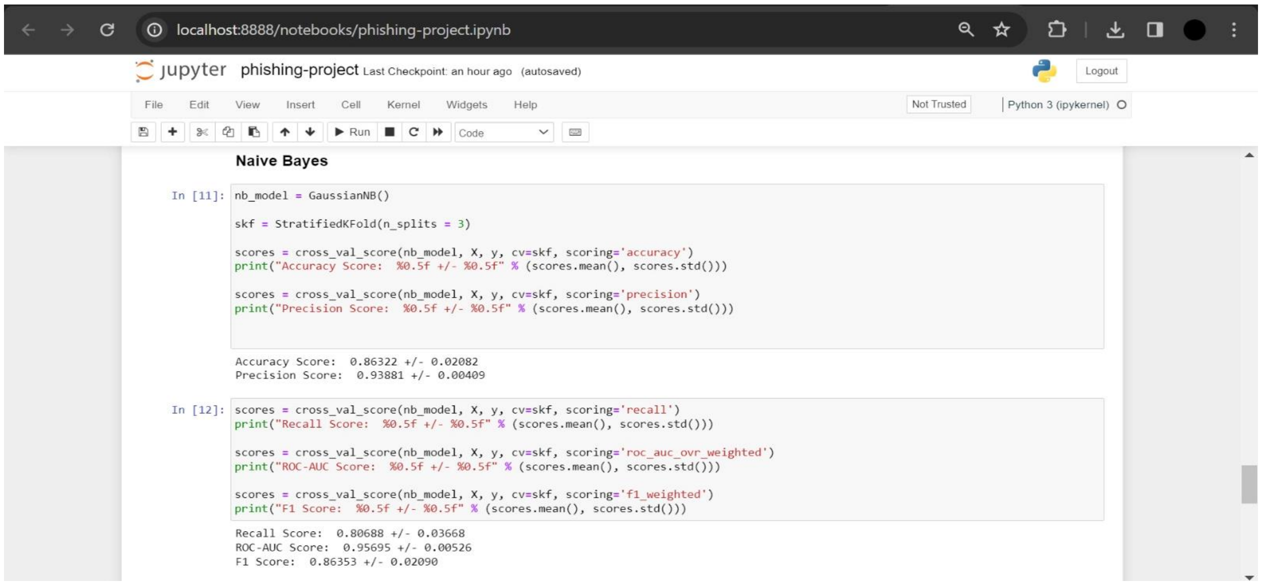
ELM Model: Showed accuracy, precision, recall, and F1 score.
ROC-AUC was not applicable for ELM.


localhost:8888/notebooks/phishing-project.ipynb
jupyter phishing-project Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

SVM Model

```
In [10]: svm_model = svm.SVC(probability=True)

         skf = StratifiedKFold(n_splits = 3)

         scores = cross_val_score(svm_model, X, y, cv=skf, scoring='accuracy')
         print("Accuracy:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(svm_model, X, y, cv=skf, scoring='precision')
         print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(svm_model, X, y, cv=skf, scoring='recall')
         print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(svm_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
         print("ROC-AUC:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(svm_model, X, y, cv=skf, scoring='f1_weighted')
         print("F1:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

```
Accuracy:  0.55699 +/- 0.00011
Precision Score:  0.55699 +/- 0.00011
Recall Score:  1.00000 +/- 0.00000
ROC-AUC:  0.69400 +/- 0.17060
F1:  0.39851 +/- 0.00013
```


localhost:8888/notebooks/phishing-project.ipynb
jupyter phishing-project Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

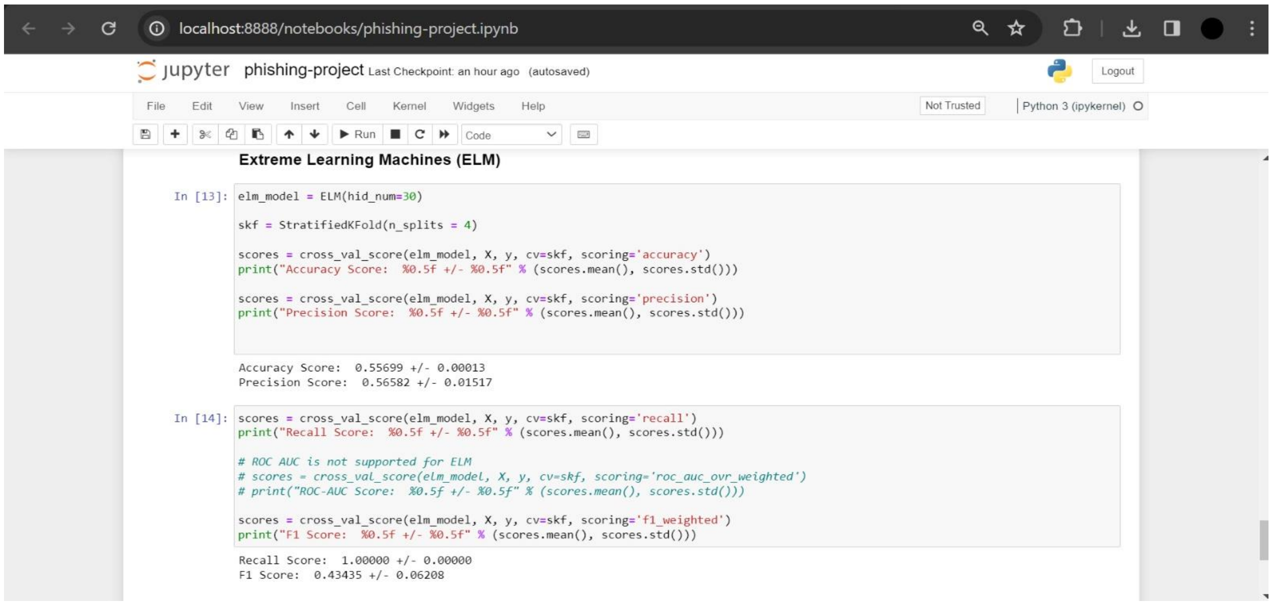
Naive Bayes

```
In [11]: nb_model = GaussianNB()

         skf = StratifiedKFold(n_splits = 3)

         scores = cross_val_score(nb_model, X, y, cv=skf, scoring='accuracy')
         print("Accuracy Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(nb_model, X, y, cv=skf, scoring='precision')
         print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

```
Accuracy Score:  0.86322 +/- 0.02082
Precision Score:  0.93881 +/- 0.00409
```

```
In [12]: scores = cross_val_score(nb_model, X, y, cv=skf, scoring='recall')
         print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(nb_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
         print("ROC-AUC Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(nb_model, X, y, cv=skf, scoring='f1_weighted')
         print("F1 Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

```
Recall Score:  0.80688 +/- 0.03668
ROC-AUC Score:  0.95695 +/- 0.00526
F1 Score:  0.86353 +/- 0.02090
```


localhost:8888/notebooks/phishing-project.ipynb
jupyter phishing-project Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

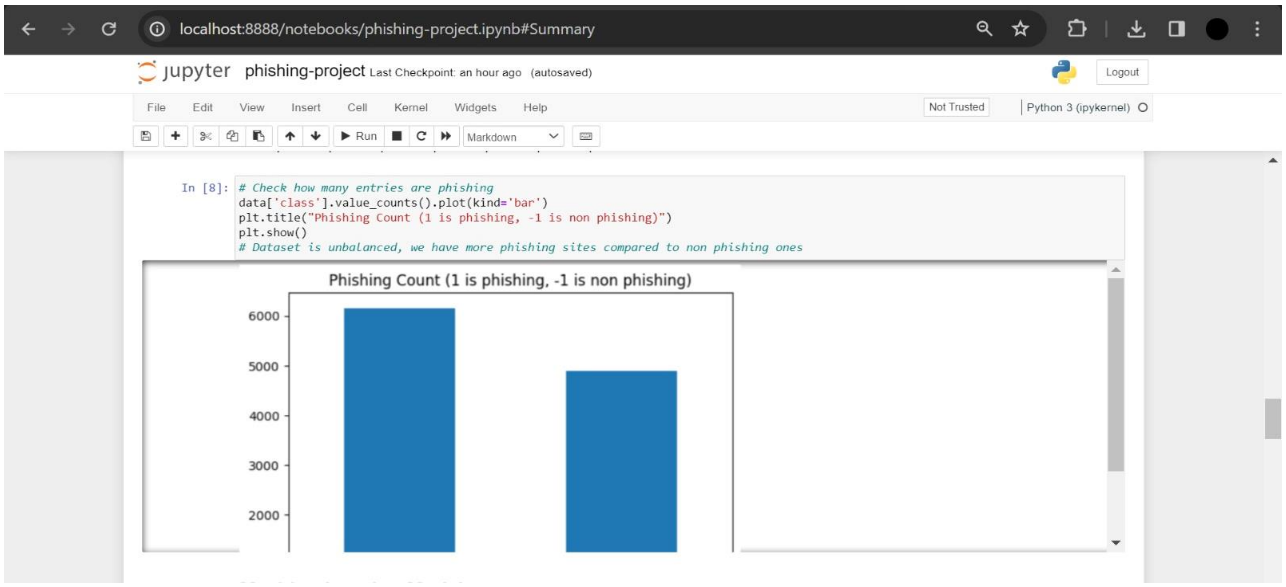
Extreme Learning Machines (ELM)

```
In [13]: elm_model = ELM(hid_num=30)

         skf = StratifiedKFold(n_splits = 4)

         scores = cross_val_score(elm_model, X, y, cv=skf, scoring='accuracy')
         print("Accuracy Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(elm_model, X, y, cv=skf, scoring='precision')
         print("Precision Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

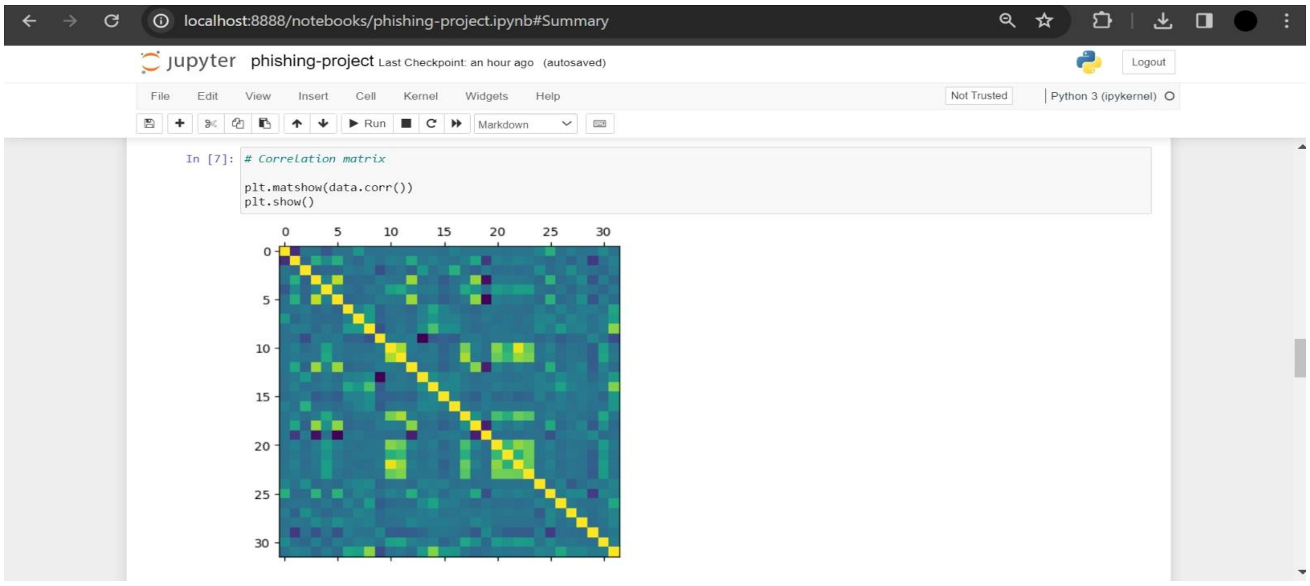
```
Accuracy Score:  0.55699 +/- 0.00013
Precision Score:  0.56582 +/- 0.01517
```

```
In [14]: scores = cross_val_score(elm_model, X, y, cv=skf, scoring='recall')
         print("Recall Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         # ROC AUC is not supported for ELM
         # scores = cross_val_score(elm_model, X, y, cv=skf, scoring='roc_auc_ovr_weighted')
         # print("ROC-AUC Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))

         scores = cross_val_score(elm_model, X, y, cv=skf, scoring='f1_weighted')
         print("F1 Score:  %0.5f +/- %0.5f" % (scores.mean(), scores.std()))
```

```
Recall Score:  1.00000 +/- 0.00000
F1 Score:  0.43435 +/- 0.06208
```


localhost:8888/notebooks/phishing-project.ipynb#Summary
jupyter phishing-project Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Phishing Count (1 is phishing, -1 is non phishing)
6000
5000
4000
3000
2000

```
In [8]: # Check how many entries are phishing
        data['class'].value_counts().plot(kind='bar')
        plt.title("Phishing Count (1 is phishing, -1 is non phishing)")
        plt.show()
        # Dataset is unbalanced, we have more phishing sites compared to non phishing ones
```



```
In [7]: # Correlation matrix

        plt.matshow(data.corr())
        plt.show()
```

Conclusion

Out of all the models, Navie bayes had the highest accuracy. ELM and SVM both performed the same at 55%, which was not very performative. But they had the highest Recall scores meaning that they were good at identifying true positives. SVM model took significantly longer than the ELM model to train and evaluate. Thus, the ELM model provided better recall with lower time to train. This would be helpful in real time scenarios where we have to retrain the model with live data.

The choice of model depends on the specific requirements of task and the importance of false positives and false negatives. If precision is crucial (minimizing false positives), consider the SVM or Naïve bayes models. If recall is more important (minimizing false negatives), again, SVM or Naïve bayes might be suitable. ELM provides an alternative, but it lacks ROC-AUC support.

Limitations:

Class Imbalance:

The dataset exhibits an imbalance between phishing and non-phishing instances. Imbalanced datasets may lead models to favor the majority class, potentially impacting performance on the minority class. Consider addressing this imbalance through techniques like oversampling, under sampling, or using different evaluation metrics.

ELM Model Limitation:

Extreme Learning Machines (ELM) lack interpretability, making it challenging to understand the underlying decision-making process. Interpretability is crucial in many applications, especially in sensitive domains such as cybersecurity.

Future Work

Hyperparameter Tuning: Conduct a more exhaustive search for optimal hyperparameters for each model. Utilize grid search or random search techniques to find the best combination of hyperparameters, especially for SVM and ELM.

Ensemble Methods: Experiment with ensemble methods, such as stacking or bagging, to combine the strengths of multiple models. Ensemble models can often achieve better generalization and robustness.

Data and Software Availability

https://github.com/GVSU-CIS635/gvsu-cis635-term-project-project-Phishing-Web-Site-Detection

We initially tried to use the ELM module from python, but we faced issues so switched to https://github.com/masaponto/Python-ELM

References

[1] Phishing Website Detection Based on Deep Convolutional Neural Network and Random Forest Ensemble Learning
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8709380/
Names: Rundong Yang,Kangfeng Zheng,Bin Wu,Chunhua Wu,Xiujuan Wang

[2] Applications of deep learning for phishing detection: systematic literature review
https://link.springer.com/article/10.1007/s10115-022-01672-x
Names: Görkem Giray,Bedir Tekinerdogan,Sandeep Kumar & Suyash Shukla

[3] Detecting Phishing Websites through Deep Reinforcement Learning
https://ieeexplore.ieee.org/document/8754075
Names: Moitrayee Chatterjee, Akbar-Siami Namin