University of Windsor

Project Report

Detect Malicious & Benign Websites using Machine Learning

COMP 4750-01 Winter 2019

Project Supervisor: Dr. Saeed Samet

Authors: Vipul Malhotra, Mayank Semwal

ABSTRACT

Malicious websites are of great concern to users and organisations nowadays because they rely on web-pages, including online shopping, obtaining information etc. and it threatens the privacy and security of users, organisation, governments as there is a problem in analysing every website one by one and to index each URL in a blacklist. Even if we cannot observe a part of malicious data, we can always observe compromised websites. Since vulnerable websites are discovered by search engines, compromised websites have similar traits. Therefore, we built a classifier by leveraging not only malicious websites but also compromised websites. Unfortunately, there are lack of datasets which can identify the difference between malicious and benign websites on the basis of their characteristics. To achieve this in our project, we have gathered datasets from multiple resources and categorized them with attributes which define the type of the website. In this project, we address the detection of malicious websites based on their features gathered from multiple sources and use machine learning classification techniques to test and train for future predictions.

1). INTRODUCTION

Drive-by [1] download attacks are the primary method for malware distribution through the Web. When a user accesses a URL that is a starting point of attacks, the user is redirected to an exploit URL via multiple redirections. At the exploit URL, vulnerabilities in browsers and their plugins are exploited, and the user is finally infected with malware. This infected user suffers from damage, such as sensitive data leakage and illegal money transfer, and is integrated into distributed denial-of-service attacks. To expose more users to threats of drive-by downloads, attackers compromise benign websites and inject redirection code to malicious websites such as redirection and exploit URLs. Attackers compromise benign websites and create malicious websites automatically. Since websites built with the old version of content management systems (CMSs) are vulnerable, attackers automatically discover them with search engines, that's the reason, in this project we propose the use of machine Learning approaches to detect the websites and train the model to predict future attacks.

In this project, we use machine learning because vulnerable websites are automatically discovered with search engines by attackers; compromised websites have similar traits. Therefore, we built a classifier by leveraging not only malicious websites but also benign websites. More precisely, we use classification algorithms such as Logistic Regression, Random Forest Classifier and Support Vector Machine (SVM). To perform classification, we integrated two large datasets one consists of predictor variable as URL and response variable as type furthermore in the second dataset we gathered information about websites such as URL, URL_LENGTH, SERVER, CHAR_SET, Domain location, etc. and their response variable as type which indicate whether it is malicious or benign. As a result of evaluating our system with crawling data of around 345000 URLs combining both datasets, we supposed that they were malicious according to their information, in a next step to verify the records in the dataset we used another security tool, VirusTotal. Hence, providing security and privacy of the user information and system is one of the main concerns of our project. In our project, we use supervised learning methods of machine learning. We learn about the fundamental differences between the three different techniques of supervised learning, i.e. Logistic Regression, Random Forest Classifier and Support Vector Machine (SVM).

The primary goal of supervised learning is to train a model from labeled training data (Figure 1) that allows us to make predictions about future data. Here, the term *supervised* refers to a classification of the desired result [4].

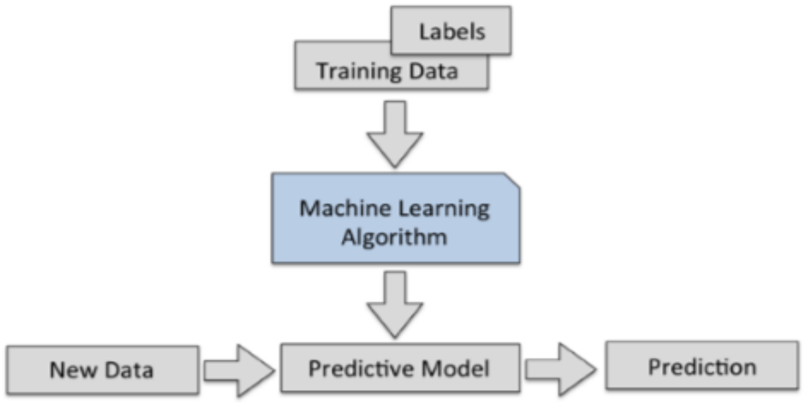


Figure 1: Machine Learning Model

1.1) Data Description:

Data consists of evaluating classification models to predict malicious and benign websites through their application layer and network characteristics. The data is collected by a process that included different sources of benign and malicious URL, all of them are verified and used in a low interactive client honeypot to get their network traffic, furthermore, we use some other tools to get more information, such as the server's country with Whois details of the website [5].

URL: It is the anonymous identification of the URL analyzed in the study.

URL_LENGTH: It is the number of characters in the URL.

NUMBER_SPECIAL_CHARACTERS: it is the number of unique characters identified in the URL.

CHARSET: It is a categorical value, and its meaning is the character encoding standard.

SERVER: It is a categorical value, and its meaning is the operative system of the server.

CONTENT_LENGTH: it represents the content size of the HTTP header.

WHOIS_COUNTRY: it is a categorical variable; its values are the countries we got from the server response (specifically, our script used the API of Whois).

WHOIS_STATEPRO: it is a categorical variable; its values are the states we got from the server response (specifically, our script used the API of Whois).

WHOIS_REGDATE: Whois provides the server registration date, so, this variable has date values with format DD/MM/YYY HH:MM

WHOIS_UPDATED_DATE: Through the Whois, we got the last update date from the server analyzed.

TCP_CONVERSATION_EXCHANGE: This variable is the number of TCP packets exchanged between the server and our honeypot client.

DIST_REMOTE_TCP_PORT: it is the number of the ports detected and different to TCP

REMOTE_IPS: this variable has the total number of IPs connected to the honeypot

APP_BYTES: this is the number of bytes transferred.

SOURCE_APP_PACKETS: packets sent from the honeypot to the server.

REMOTE_APP_PACKETS: packets received from the server.

APP_PACKETS: this is the total number of IP packets generated during the communication between the honeypot and the server.

DNS_QUERY_TIMES: this is the number of DNS packets generated during the communication between the honeypot and the server.

TYPE: this is a categorical variable, its values represent the type of web page analyzed, specifically, one is for malicious websites, and zero is for benign websites.

2). IMPLEMENTATION

In this section, we will discuss the roadmap for building our project using the machine learning. (Figure 2), below shows the workflow diagram for predictive modeling [6].

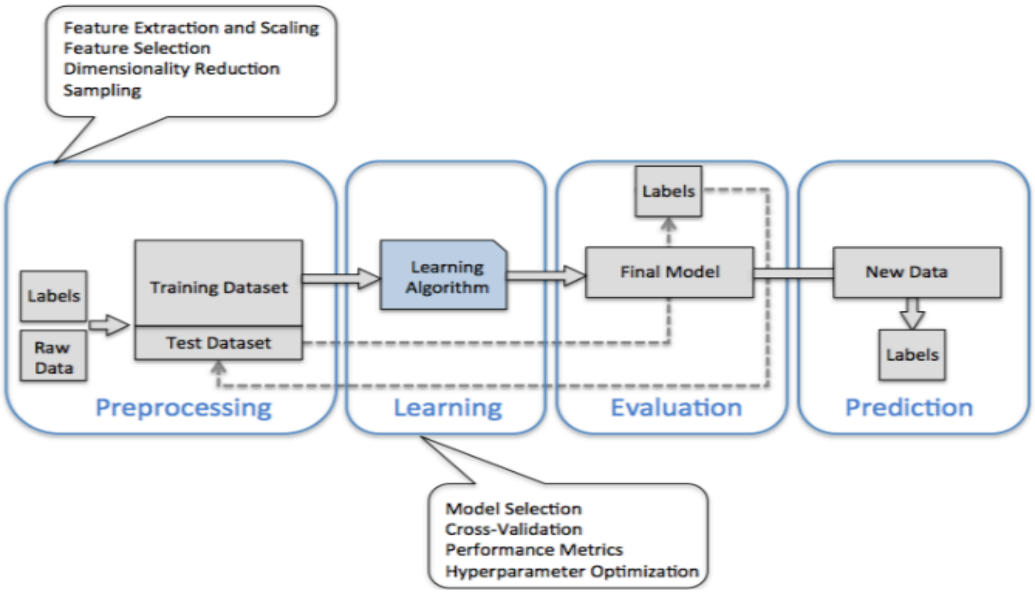


Figure 2: Workflow diagram

- **Preprocessing:** Some of the selected features may be highly correlated and therefore redundant to a certain degree. To keep the raw data in the form and shape for optimal performance of the learning algorithm, dimensionality reduction techniques in Random Forest Classifier, SVM and Logistic regression for achieving better accuracy and efficiency. Its advantage is less storage space and can run faster [6]. We use the training set to train and optimize our machine learning model, while we keep the test set until the very end to evaluate the final model.

- **Learning Algorithm:** In this, we use three algorithms to train the data and check the accuracy and compare them to find the optimal solution [6].
  - Model Selection: Select a model which will give more efficient results while implementation of the algorithm, training complex dataset and predicting the desired output.
  - Cross-Validation: Validate test and trained data with the predicted result to cross-validate the accuracy and performance.
  - Performance Metrics: Compare classification as mentioned earlier techniques to check their predicted results to find out which technique is more efficient.
  - Hyperparameter Optimization: A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

- **Evaluation:** Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the selected model will work in the future.

- **Prediction:** Machine learning is a way of identifying patterns in data and using them to make predictions or decisions automatically.

3). MODEL TESTING USING MENTIONED CLASSIFICATION TECHNIQUES

3.1) Logistic Regression: In spite of its name logistic regression is a model for classification, not regression. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more interval or ratio-level independent variables. Logistic Regression used when the dependent variable(target) is categorical [3]**.** Input values (x) are combined linearly using weights or coefficient values (Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

y = e^ (b0 + b1*x) / (1 + e^ (b0 + b1*x)). ------ *(1)*

Where *y* is the predicted output, *b0* is the bias or intercept term and *b1* is the coefficient for the single input value (x). Each column in your input data has an associated *b* coefficient (a constant real value) that must be learned from your training data [6].

Algorithm for logistic regression:

1. Input: Set of (input, output) training pair samples; call the input sample features x1, x2 to xn, and the output result y. Typically, there can be lots of input features xi.
2. Let p(x) be a linear function of x. Every increment of a component of x would add or subtract so much to the probability. The conceptual problem here is that p must be between 0 and 1, and linear functions are unbounded.

*y = b0 + b1*x.* ----------*(2)*

3. Find **odds ratio** which is odds in the favour of the particular event.

p/(1-p), where p stands for the probability of the positive event --------(3)

4. Now define the logit function to calculate the logarithm of the odds ratio

$$logit(p)=log\ p/(1-p).$$ -------- *(4)*

5. Logit function takes input values in the range 0 to 1 and transforms them to values over the entire real number range, which express a linear relationship between feature values and the log-odds:

$$logit(p(y=1|\boldsymbol{x})) = w_0x_0+w_1x_1+...+w_mx_m = \sum_{i=0}^{m} wixi = wTx$$ -------*(5)*

Here, p (y =1| x) is the conditional probability that a particular sample belongs to class A given its features x.

6. Now to predict the probability in order to classify the class, use logistic function/sigmoid function:

$$\varphi(z)= 1 / (1+e^{-z}) \qquad \text{--------}(6)$$

7. Output: set of weights w (or $\mathbf{w}_i$), one for each feature, whose linear combination predicts the value of y.

3.2) Support Vector Machine (SVM): The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N is number of features) that distinctly classifies the data points **Figure** 3 .

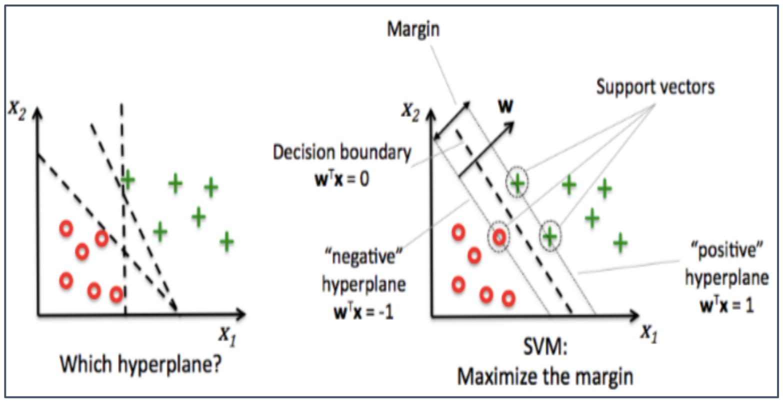


Figure 3:Support Vector Diagram

To separate the two classes of data points, there are many possible hyperplanes that can be selected. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points falling on either side of the hyperplane can be attributed to different classes.

Learning of hyperplane, it is done by using linear algebra i.e. kernel, to linearly separate each variable. For high dimensional data other kernels are used and are of four types: Linear, Polynomial, RBF (Radial Basis function), Sigmoid. SVM minimizes the misclassification errors by maximizing the margin. We have used RBF which non-linearly separate the variables. For distance metric, squared Euclidean distance is used.

Algorithm for SVM:

1. **Input**: Set of (input, output) training pair samples; call the input sample features x1, x2 to xn, and the output result y. Typically, there can be lots of input features xi.
2. Begin by training two or more SVMs based on tree kernel function, namely, svm1 and svm2, on x1 and x2 to xn, respectively from n number of features

   svm1.train and x1 = M1 model

   svm2.train and x2 = M2 model

   classify the relations on training sets by using two or more trained SVM models (M1 model and M2 model), and then generate different classification result on the same set of input features x1 and x2.
3. The fit method of SVC class will call to train the algorithm on the training dataset, which is passed as a parameter to the fit method.
4. To maximize the margin of hyperplane in the fit process of step 3, to maximize below are the four steps:
   1. Find *positive* and *negative* hyperplanes that are parallel to the decision boundary, which can be expressed.

      $$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \qquad (1)$$

      $$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \qquad (2)$$

   2. Then we subtract those two linear equations (**1**) and (**2**) from each other to find the hyperplane.

      $$\Rightarrow \boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right) = 2$$

   3. Now further to maximize margin hyperplane for SVM trained with samples known as support vectors.

      $$\|\boldsymbol{w}\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

   4. So Finally, we arrive at the following equation:

      $$\frac{\boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

5. Finally, now once we have trained the algorithm, the next step is to make predictions on the test data using predict method.

3.3) Random Forest Classifier: Random Forest creates multiple decision trees and makes it random. The forest (multiple decision trees) it builds is an ensemble of decision trees. Random forest builds multiple decision trees and merges them to get a more accurate and stable prediction [2]. A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset. It uses averaging to improve the predictive accuracy and control over-fitting [2]. One big advantage of the random forest is that it can be used for both classification and regression problems. We will talk about the random forest in classification since classification is sometimes considered the building block of machine learning. Random Forest is also considered as a very handy and easy to use the algorithm, because it's default hyperparameters often produce a good prediction result [6]. The number of hyper-parameters is also not that high and they are straightforward to understand.

Steps performed by random forest*:*

Input*:* Set of (input, output) training pair samples; call the input sample features x1, x2 to xn, and the output result y. Typically, there can be lots of input features xi.

1. Randomly select "k" features from total "m" features of dataset.
Where k < m

2. Among the "k" features, calculate the node "d" using the best split point.

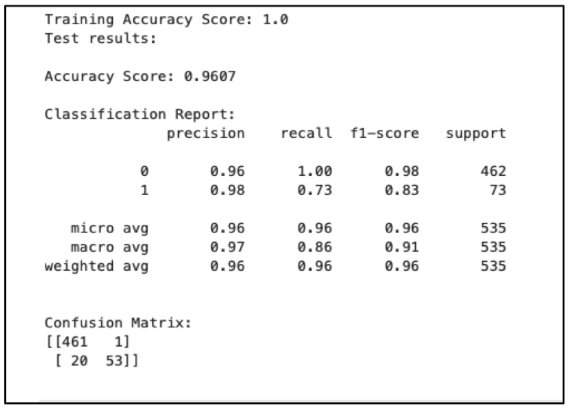
3. Split the node into daughter nodes using the best split.

4. Repeat 1 to 3 steps until "l" number of nodes has been reached.

5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

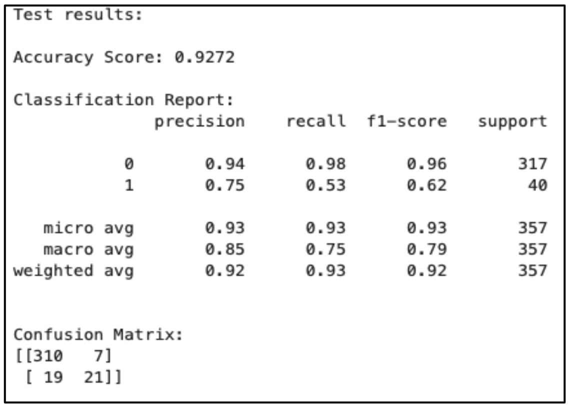
Output*: In average it takes all the predictions, which cancels out the biases, and attains performance by selecting the best feature from decisions instead of most important feature*

4). PERFORMANCE EVALUATION: Models appear to perform similarly across the datasets with performance more influenced by choice of dataset rather than model selection. Hence, we have selected three classification algorithms and compared each one of them based on of their accuracy with the corresponding dataset. To measure the performance of classification model, confusion matrix is used. It is described below with the formulas to measure accuracy, precision, recall and f1-score in Figure 7.



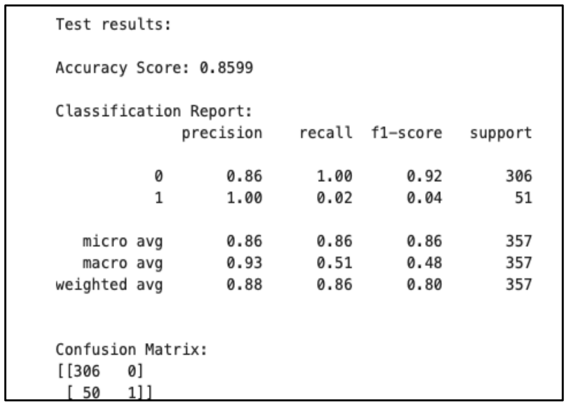
```
Training Accuracy Score: 1.0
Test results:

Accuracy Score: 0.9607

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       462
           1       0.98      0.73      0.83        73

   micro avg       0.96      0.96      0.96       535
   macro avg       0.97      0.86      0.91       535
weighted avg       0.96      0.96      0.96       535


Confusion Matrix:
[[461   1]
 [ 20  53]]
```

Figure 4: Random Forest Test Results



```
Test results:

Accuracy Score: 0.9272

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       317
           1       0.75      0.53      0.62        40

   micro avg       0.93      0.93      0.93       357
   macro avg       0.85      0.75      0.79       357
weighted avg       0.92      0.93      0.92       357


Confusion Matrix:
[[310   7]
 [ 19  21]]
```

Figure 5: Logistic Regression Test Results



```
Test results:

Accuracy Score: 0.8599

Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       306
           1       1.00      0.02      0.04        51

   micro avg       0.86      0.86      0.86       357
   macro avg       0.93      0.51      0.48       357
weighted avg       0.88      0.86      0.80       357


Confusion Matrix:
[[306   0]
 [ 50   1]]
```

Figure 6: SVM Test Results

Results obtained from the model display that Random forest classifier performed and predicted better with increased accuracy results in term of training and testing. Random forest achieved 96% in predicting and detecting the malicious and benign websites based on their features, whereas SVM is the least favorable in predicting the results.

Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It is a table with 4 different combinations of predicted and actual values in the case for a binary classifier.

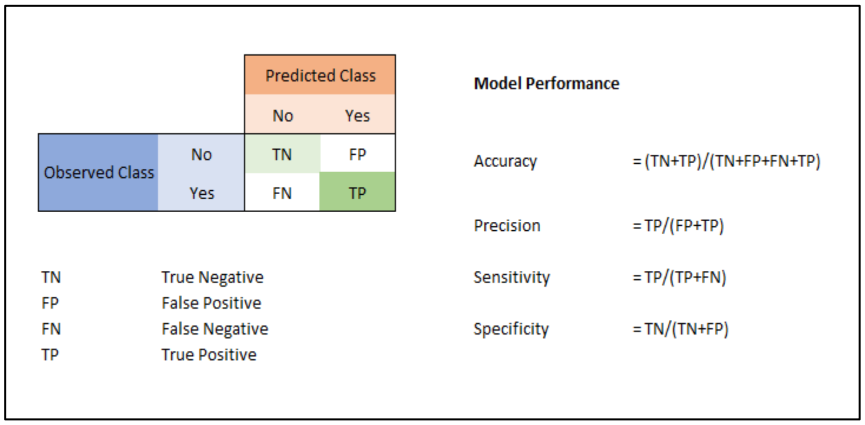
- A **true positive** is an outcome where the model *correctly* predicts the *positive* class. Similarly, a **true negative** is an outcome where the model *correctly* predicts the *negative* class.

  Accuracy: Accuracy is the fraction of predictions that the model got right.

  Precision: Out of all observed classes, how much predicted correctly.

  Recall: Out of all the positive classes, how much predicted correctly.

  F-1 Score: To combine precision and recall into a single metric called the F1 score.



		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

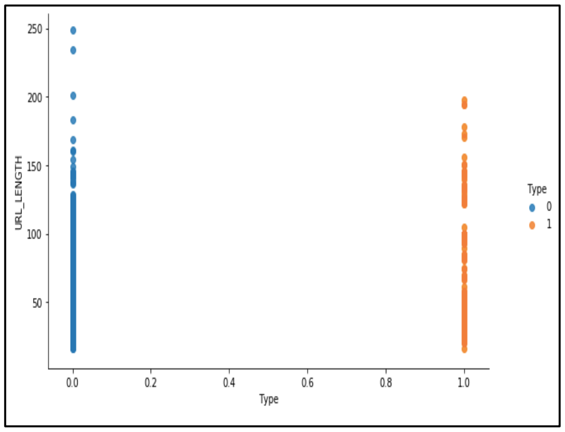
TN	True Negative
FP	False Positive
FN	False Negative
TP	True Positive

Model Performance

Accuracy	=(TN+TP)/(TN+FP+FN+TP)
Precision	=TP/(FP+TP)
Sensitivity	=TP/(TP+FN)
Specificity	=TN/(TN+FP)

Figure 7: Confusion Matrix

5). VISUALIZATION:



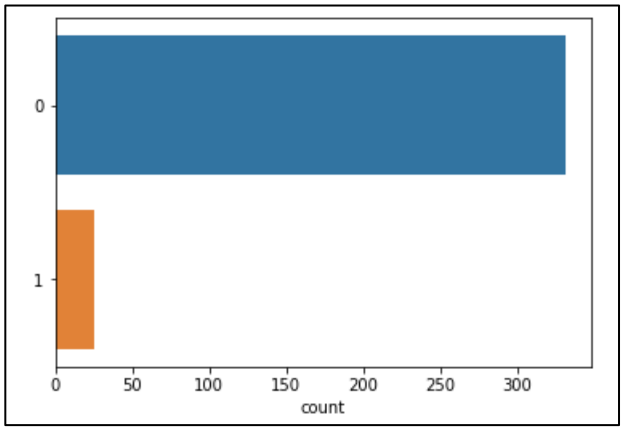


Figure 8: Initial count in Logistic Regression Figure 9: Predicted count in Logistic Regression

In Figure *8*, the graph displays the count of malicious and benign websites in the raw data for a single feature (here URL_LENGTH feature) in the dataset. Similarly, we can check for N number of features. We can see the difference between malicious and benign is almost equal so to detect better in Figure 9 the model combined all the features and predicted based on that, Figure 9 is the predicted results and it can distinguish the difference significantly to achieve accuracy.

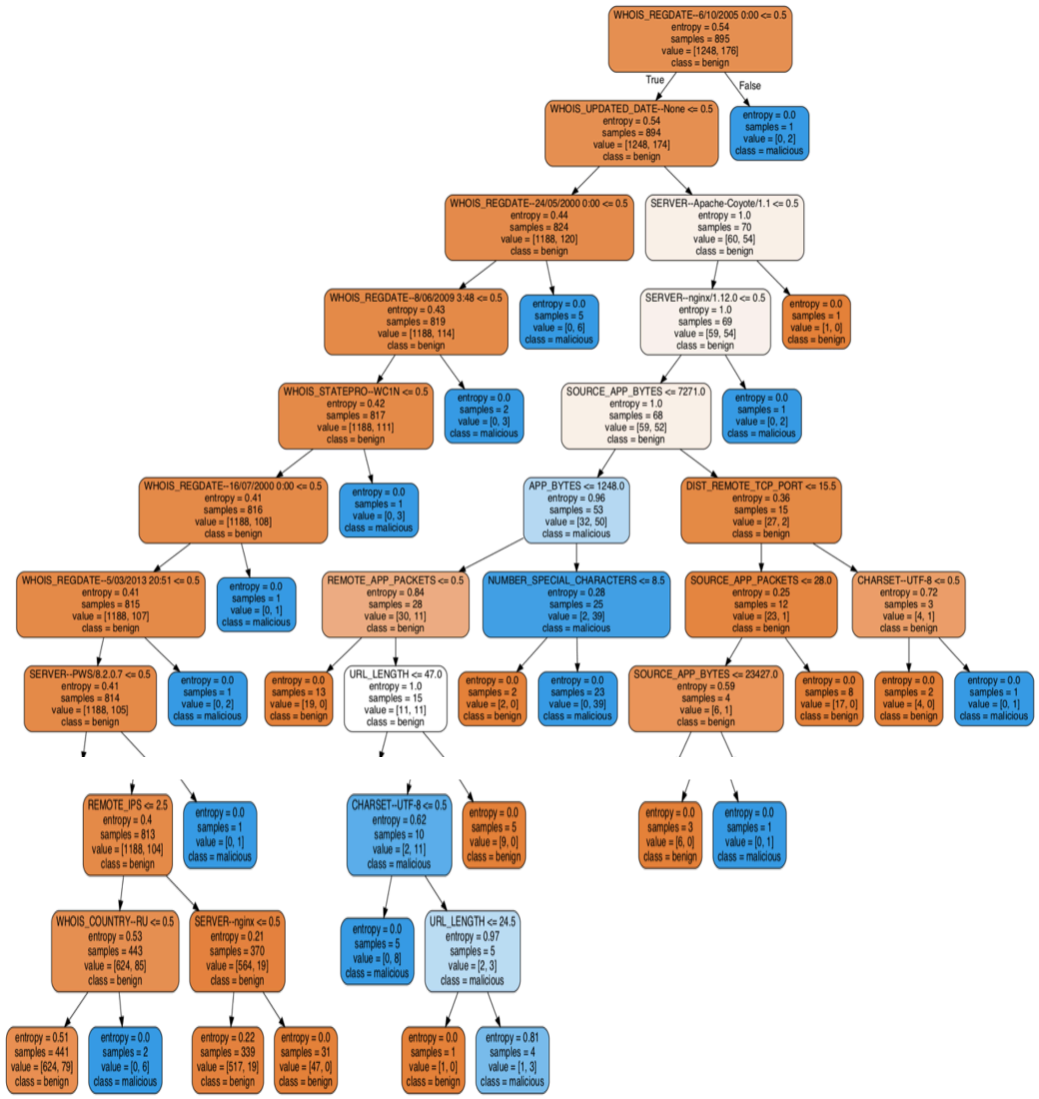

WHOIS_REGDATE--6/10/2005 0:00 <= 0.5 entropy = 0.54 samples = 895 value = [1248, 176] class = benign
True
False
WHOIS_UPDATED_DATE--None <= 0.5 entropy = 0.54 samples = 894 value = [1248, 174] class = benign
entropy = 0.0 samples = 1 value = [0, 2] class = malicious
WHOIS_REGDATE--24/05/2000 0:00 <= 0.5 entropy = 0.44 samples = 824 value = [1188, 120] class = benign
SERVER--Apache-Coyote/1.1 <= 0.5 entropy = 1.0 samples = 70 value = [60, 54] class = benign
WHOIS_REGDATE--8/06/2009 3:48 <= 0.5 entropy = 0.43 samples = 819 value = [1188, 114] class = benign
entropy = 0.0 samples = 5 value = [0, 6] class = malicious
SERVER--nginx/1.12.0 <= 0.5 entropy = 1.0 samples = 69 value = [59, 54] class = benign
entropy = 0.0 samples = 1 value = [1, 0] class = benign
WHOIS_STATEPRO--WC1N <= 0.5 entropy = 0.42 samples = 817 value = [1188, 111] class = benign
entropy = 0.0 samples = 2 value = [0, 3] class = malicious
SOURCE_APP_BYTES <= 7271.0 entropy = 1.0 samples = 68 value = [59, 52] class = benign
entropy = 0.0 samples = 1 value = [0, 2] class = malicious
WHOIS_REGDATE--16/07/2000 0:00 <= 0.5 entropy = 0.41 samples = 816 value = [1188, 108] class = benign
entropy = 0.0 samples = 1 value = [0, 3] class = malicious
APP_BYTES <= 1248.0 entropy = 0.96 samples = 53 value = [32, 50] class = malicious
DIST_REMOTE_TCP_PORT <= 15.5 entropy = 0.36 samples = 15 value = [27, 2] class = benign
WHOIS_REGDATE--5/03/2013 20:51 <= 0.5 entropy = 0.41 samples = 815 value = [1188, 107] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = malicious
REMOTE_APP_PACKETS <= 0.5 entropy = 0.84 samples = 28 value = [30, 11] class = benign
NUMBER_SPECIAL_CHARACTERS <= 8.5 entropy = 0.28 samples = 25 value = [2, 39] class = malicious
SOURCE_APP_PACKETS <= 28.0 entropy = 0.25 samples = 12 value = [23, 1] class = benign
CHARSET--UTF-8 <= 0.5 entropy = 0.72 samples = 3 value = [4, 1] class = benign
SERVER--PWS/8.2.0.7 <= 0.5 entropy = 0.41 samples = 814 value = [1188, 105] class = benign
entropy = 0.0 samples = 1 value = [0, 2] class = malicious
entropy = 0.0 samples = 13 value = [19, 0] class = benign
URL_LENGTH <= 47.0 entropy = 1.0 samples = 15 value = [11, 11] class = benign
entropy = 0.0 samples = 2 value = [2, 0] class = benign
entropy = 0.0 samples = 23 value = [0, 39] class = malicious
SOURCE_APP_BYTES <= 23427.0 entropy = 0.59 samples = 4 value = [6, 1] class = benign
entropy = 0.0 samples = 8 value = [17, 0] class = benign
entropy = 0.0 samples = 2 value = [4, 0] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = malicious
REMOTE_IPS <= 2.5 entropy = 0.4 samples = 813 value = [1188, 104] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = malicious
CHARSET--UTF-8 <= 0.5 entropy = 0.62 samples = 10 value = [2, 11] class = malicious
entropy = 0.0 samples = 5 value = [9, 0] class = benign
entropy = 0.0 samples = 3 value = [6, 0] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = malicious
WHOIS_COUNTRY--RU <= 0.5 entropy = 0.53 samples = 443 value = [624, 85] class = benign
SERVER--nginx <= 0.5 entropy = 0.21 samples = 370 value = [564, 19] class = benign
entropy = 0.0 samples = 5 value = [0, 8] class = malicious
URL_LENGTH <= 24.5 entropy = 0.97 samples = 5 value = [2, 3] class = malicious
entropy = 0.51 samples = 441 value = [624, 79] class = benign
entropy = 0.0 samples = 2 value = [0, 6] class = malicious
entropy = 0.22 samples = 339 value = [517, 19] class = benign
entropy = 0.0 samples = 31 value = [47, 0] class = benign
entropy = 0.0 samples = 1 value = [1, 0] class = benign
entropy = 0.81 samples = 4 value = [1, 3] class = malicious

Figure 10:Random Forest Predicted Result

Figure 10 represents the predicted results by random forest classifier. Random Forest algorithm randomly selects observations and features to build several decision trees and then averages the results based on the measurement of the relative importance (Figure *11*) of each feature on the prediction.

Figure *11* is the correlation heat map, which is features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features. In Figure *12* the predicted correlation between malicious and benign websites.

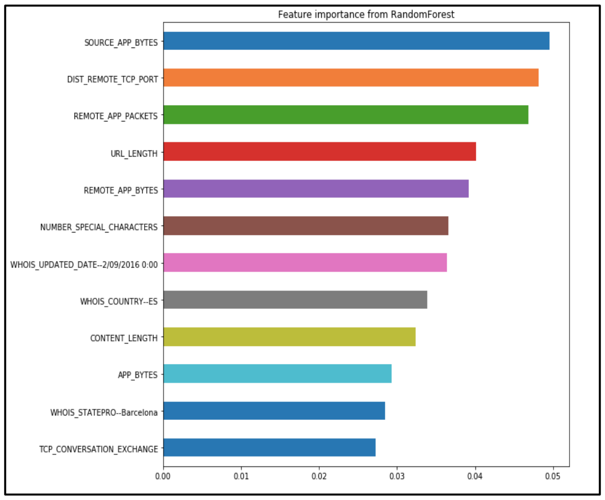


Figure 11:Feature Importance in Random Forest

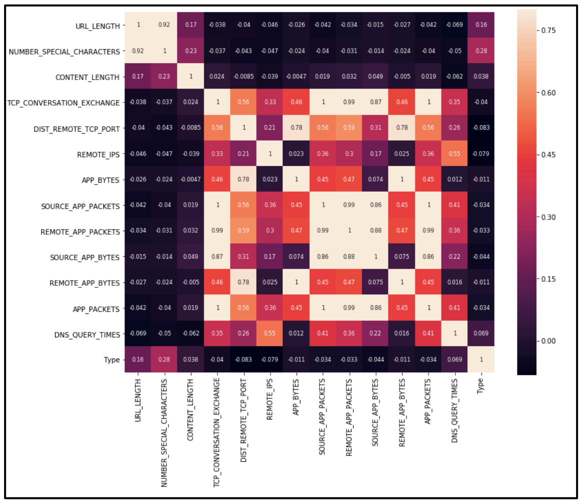


Figure 12:SVM Correlation heat map

6). CONCLUSION: Studied Supervised Learning and used models to classify benign and malicious websites. The experimental results show that the Random Forest performed better than other classification techniques used. RFC works best for unbalanced data because it does not consist of skewness (E.g., Mean). Unequal instances for different classes. The overfitting problem will never come when we use the random forest algorithm in any classification problem. Logistic Regression can be used, if speed is the criteria, whereas in the case of categorical data RF will perform better. However, Better use of misclassified data to update the weights and classify them again to predict based on of more data, because more data leads to better prediction. Finally, models appear to perform similarly across the datasets with performance more influenced by choice of dataset rather than model selection.

7). FUTURE WORK: Need to be implemented for all other classification techniques in Machine Learning and Deep learning algorithms to compare results from all the techniques.

In future, Important task will be to assemble feature extraction from website at runtime and perform dynamic detection from the trained dataset. Further, Evolutionary computation will be best to learn from dynamic features of websites and predict better results of the everchanging internet.

8). REFERENCES

[1]. Toshiki Shibahara ; Yuta Takata ; Mitsuaki Akiyama ; Takeshi Yagi ; Takeshi Yada, “Detecting Malicious Websites by Integrating Malicious, Benign, and Compromised Redirection Subgraph Similarities”, “2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), pp. 655 – 664, 2017”.

[2]. Niklas Donges ; “The Random Forest Algorithm”, https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd”.

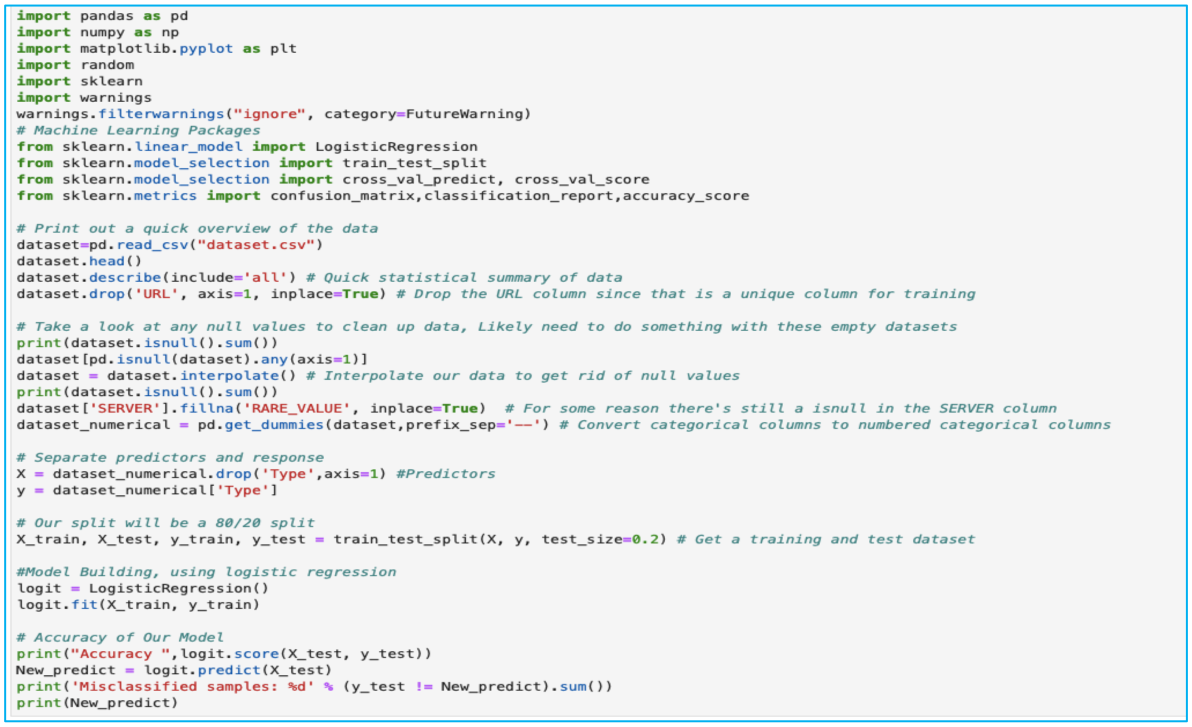
[3]. Jason Brownlee ; Understand Machine Learning Algorithms, https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

[4]. Badreesh Shetty; Supervised Machine Learning: Classification, https://towardsdatascience.com/supervised-machine-learning-classification-5e685fe18a6d

[5]. Christian Urcuqui; https://www.kaggle.com/xwolf12/malicious-and-benign-websites/home.

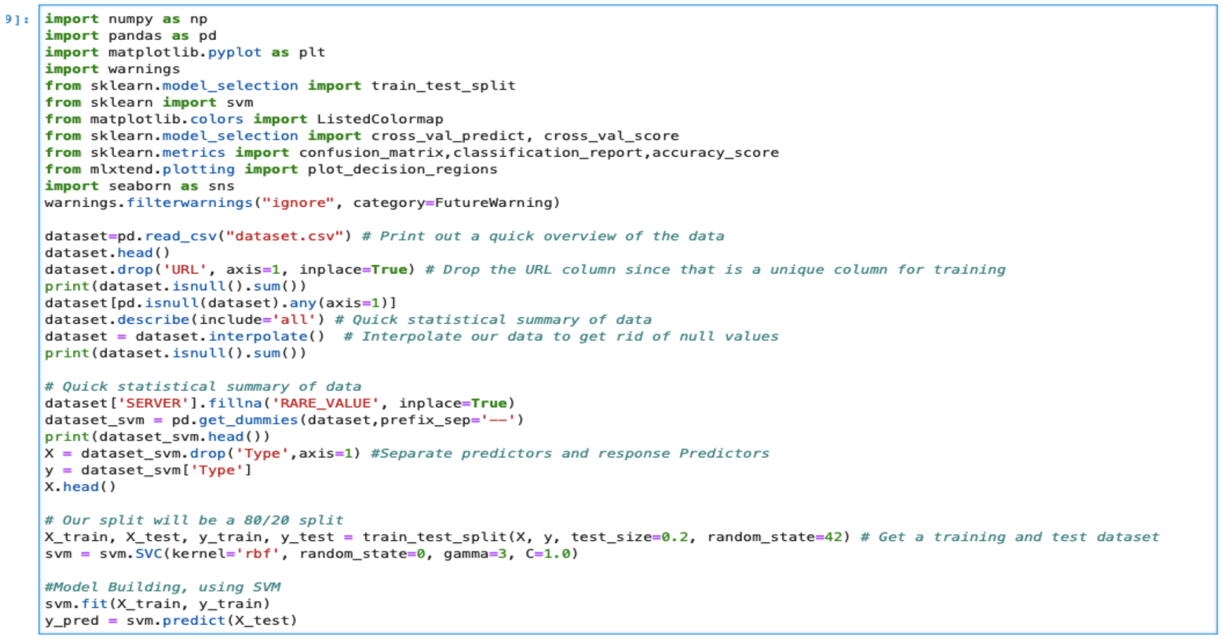
[6]. Sebastian Raschka, Python Machine Learning, 1st Edition.

Appendix



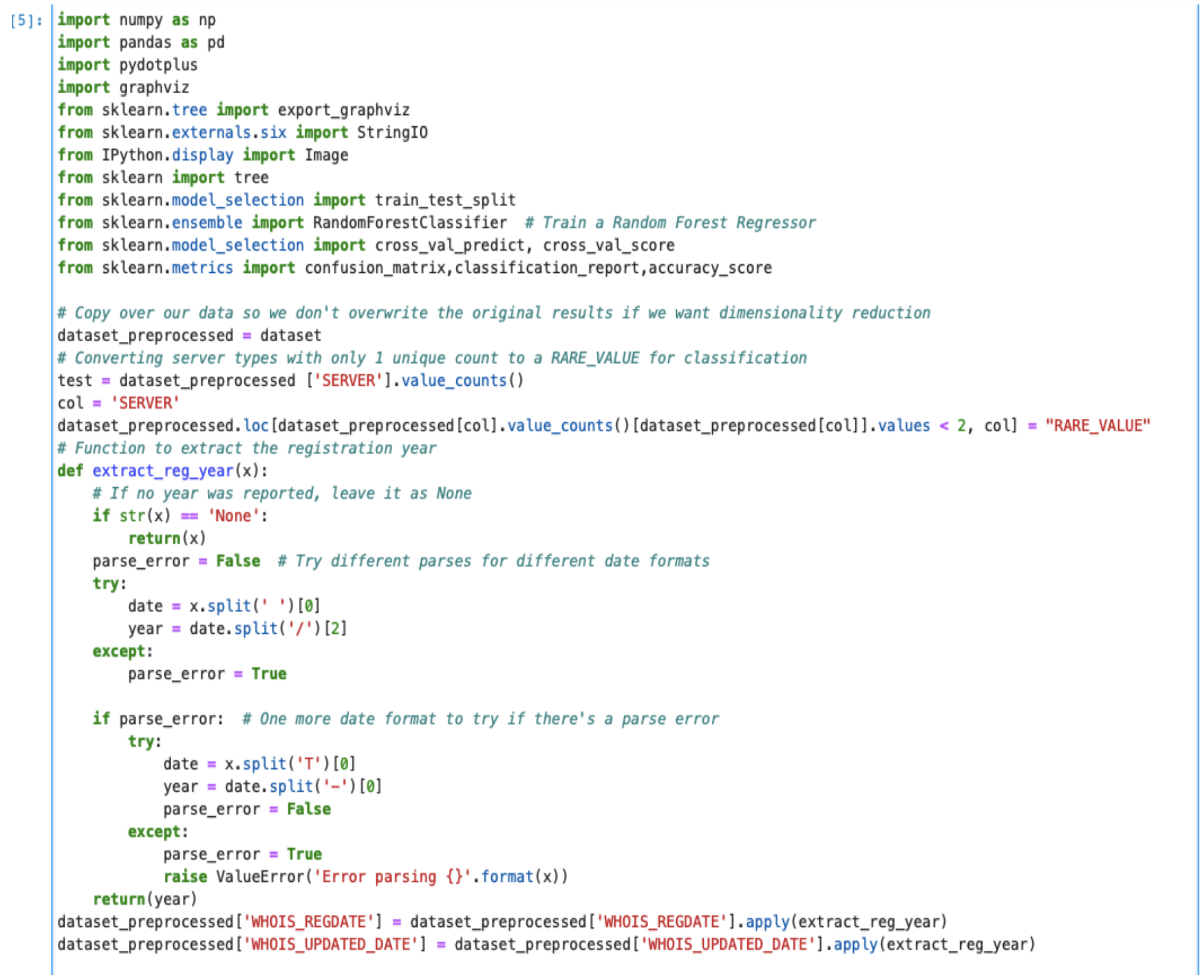
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import sklearn
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
# Machine Learning Packages
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

# Print out a quick overview of the data
dataset=pd.read_csv("dataset.csv")
dataset.head()
dataset.describe(include='all') # Quick statistical summary of data
dataset.drop('URL', axis=1, inplace=True) # Drop the URL column since that is a unique column for training

# Take a look at any null values to clean up data, Likely need to do something with these empty datasets
print(dataset.isnull().sum())
dataset[pd.isnull(dataset).any(axis=1)]
dataset = dataset.interpolate() # Interpolate our data to get rid of null values
print(dataset.isnull().sum())
dataset['SERVER'].fillna('RARE_VALUE', inplace=True)  # For some reason there's still a isnull in the SERVER column
dataset_numerical = pd.get_dummies(dataset,prefix_sep='--') # Convert categorical columns to numbered categorical columns

# Separate predictors and response
X = dataset_numerical.drop('Type',axis=1) #Predictors
y = dataset_numerical['Type']

# Our split will be a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # Get a training and test dataset

#Model Building, using logistic regression
logit = LogisticRegression()
logit.fit(X_train, y_train)

# Accuracy of Our Model
print("Accuracy ",logit.score(X_test, y_test))
New_predict = logit.predict(X_test)
print('Misclassified samples: %d' % (y_test != New_predict).sum())
print(New_predict)
```

Code implementation of Logistic Regression

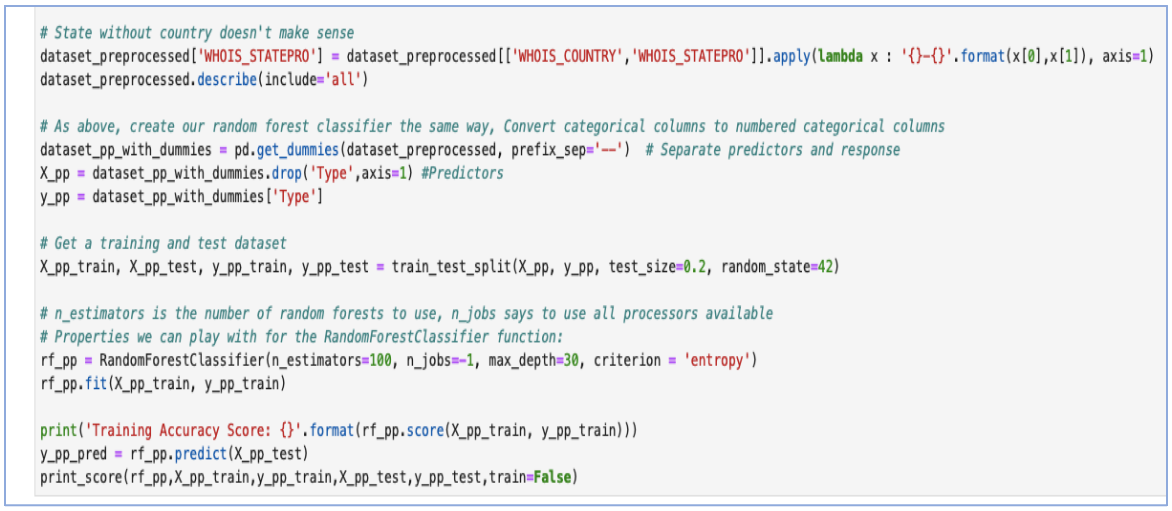


```python
9]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from sklearn import svm
from matplotlib.colors import ListedColormap
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from mlxtend.plotting import plot_decision_regions
import seaborn as sns
warnings.filterwarnings("ignore", category=FutureWarning)

dataset=pd.read_csv("dataset.csv") # Print out a quick overview of the data
dataset.head()
dataset.drop('URL', axis=1, inplace=True) # Drop the URL column since that is a unique column for training
print(dataset.isnull().sum())
dataset[pd.isnull(dataset).any(axis=1)]
dataset.describe(include='all') # Quick statistical summary of data
dataset = dataset.interpolate()  # Interpolate our data to get rid of null values
print(dataset.isnull().sum())

# Quick statistical summary of data
dataset['SERVER'].fillna('RARE_VALUE', inplace=True)
dataset_svm = pd.get_dummies(dataset,prefix_sep='--')
print(dataset_svm.head())
X = dataset_svm.drop('Type',axis=1) #Separate predictors and response Predictors
y = dataset_svm['Type']
X.head()

# Our split will be a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Get a training and test dataset
svm = svm.SVC(kernel='rbf', random_state=0, gamma=3, C=1.0)

#Model Building, using SVM
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
```

Code implementation of Support Vector Machine



```
[5]: import numpy as np
import pandas as pd
import pydotplus
import graphviz
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier  # Train a Random Forest Regressor
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

# Copy over our data so we don't overwrite the original results if we want dimensionality reduction
dataset_preprocessed = dataset
# Converting server types with only 1 unique count to a RARE_VALUE for classification
test = dataset_preprocessed ['SERVER'].value_counts()
col = 'SERVER'
dataset_preprocessed.loc[dataset_preprocessed[col].value_counts()[dataset_preprocessed[col]].values < 2, col] = "RARE_VALUE"
# Function to extract the registration year
def extract_reg_year(x):
    # If no year was reported, leave it as None
    if str(x) == 'None':
        return(x)
    parse_error = False  # Try different parses for different date formats
    try:
        date = x.split(' ')[0]
        year = date.split('/')[2]
    except:
        parse_error = True

    if parse_error:  # One more date format to try if there's a parse error
        try:
            date = x.split('T')[0]
            year = date.split('-')[0]
            parse_error = False
        except:
            parse_error = True
            raise ValueError('Error parsing {}'.format(x))
    return(year)
dataset_preprocessed['WHOIS_REGDATE'] = dataset_preprocessed['WHOIS_REGDATE'].apply(extract_reg_year)
dataset_preprocessed['WHOIS_UPDATED_DATE'] = dataset_preprocessed['WHOIS_UPDATED_DATE'].apply(extract_reg_year)
```



```
# State without country doesn't make sense
dataset_preprocessed['WHOIS_STATEPRO'] = dataset_preprocessed[['WHOIS_COUNTRY','WHOIS_STATEPRO']].apply(lambda x : '{}-{}'.format(x[0],x[1]), axis=1)
dataset_preprocessed.describe(include='all')

# As above, create our random forest classifier the same way, Convert categorical columns to numbered categorical columns
dataset_pp_with_dummies = pd.get_dummies(dataset_preprocessed, prefix_sep='--')  # Separate predictors and response
X_pp = dataset_pp_with_dummies.drop('Type',axis=1) #Predictors
y_pp = dataset_pp_with_dummies['Type']

# Get a training and test dataset
X_pp_train, X_pp_test, y_pp_train, y_pp_test = train_test_split(X_pp, y_pp, test_size=0.2, random_state=42)

# n_estimators is the number of random forests to use, n_jobs says to use all processors available
# Properties we can play with for the RandomForestClassifier function:
rf_pp = RandomForestClassifier(n_estimators=100, n_jobs=-1, max_depth=30, criterion = 'entropy')
rf_pp.fit(X_pp_train, y_pp_train)

print('Training Accuracy Score: {}'.format(rf_pp.score(X_pp_train, y_pp_train)))
y_pp_pred = rf_pp.predict(X_pp_test)
print_score(rf_pp,X_pp_train,y_pp_train,X_pp_test,y_pp_test,train=False)
```

Code implementation of Random Forest Classifier