Author: Katrina Mizuo
Mentors: Joann Chen and Zhou Li

Malicious URL Detection Using Machine Learning

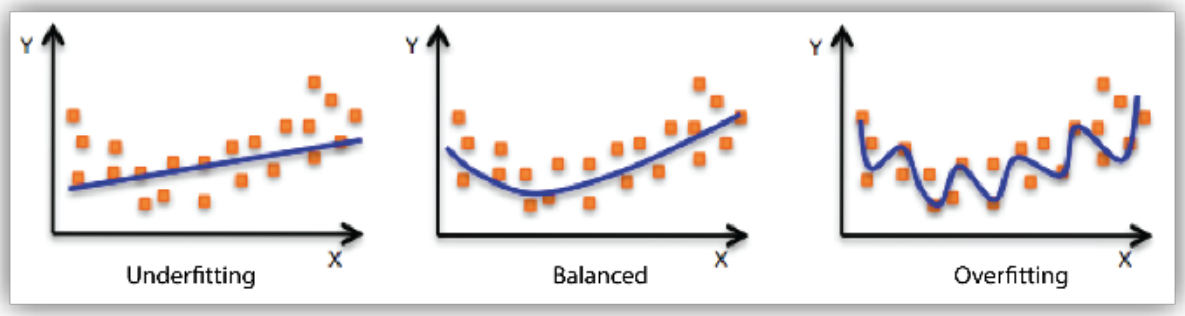
Motivation

Much of what we do in our world today takes place online. The Internet is home to innumerable websites, referred to by URLs (Uniform Resource Locators), with the possibility of some being malicious. Users may click on or visit malicious URLs, unknowingly downloading malware or viruses onto their computer. While it would be useful to blacklist malicious URLs, attackers can sometimes mask their website as a harmless one or create new websites to attack. Instead, machine learning can be applied to this issue to recognize malicious URLs based on characteristics, and make sure new malicious URLs are blacklisted.

Background

When testing data, results can sometimes contain false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN). False positives are when benign URLs are incorrectly categorized as malicious, and false negatives are when malicious URLs are incorrectly categorized as benign. On the other hand, true positives and true negatives are correctly categorized malicious and benign URLs, respectively.

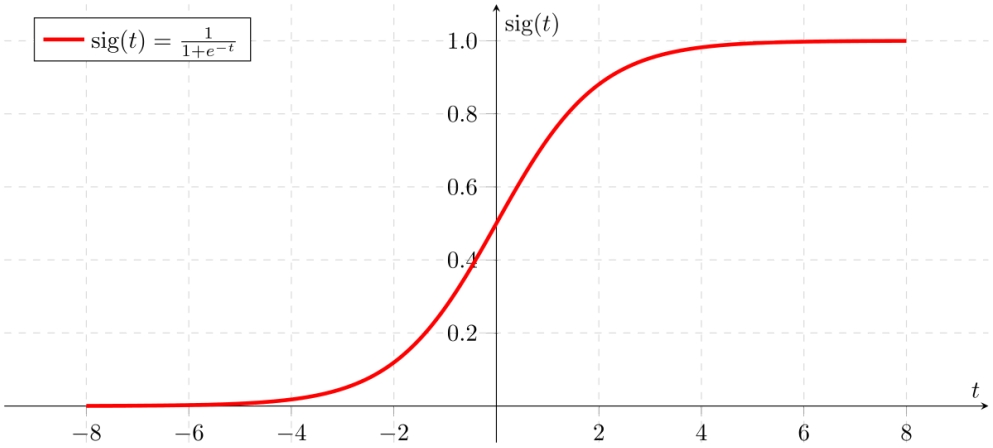
For the machine-learning model to function correctly, it has to use a balanced dataset. With an unbalanced dataset, there will be more of one type of data than the other, and the model's accuracy may be thrown off. For example, the first dataset has 156,937 URLs altogether, but only 56,937 are malicious while 100,000 are benign. Therefore, the dataset is unbalanced, and it would be important to balance it for training and testing. There are also problems that arise when training the machine-learning model. Underfitting can occur when the model is unable to detect patterns in the training data, and overfitting can occur when the model memorizes the training data, but fails to perform well with testing data (see graphs below).



In order to avoid these issues and improve accuracy, two balanced datasets - one for training and one for testing - each with approximately 50% malicious URLs and 50% benign URLs would be optimal. To obtain these datasets, one large dataset will be split, with 80% becoming the training set and the remaining 20% becoming the testing set. Multiple machine-learning models will also be tested to see which performs best with the same datasets.

Logistic Regression

Logistic regression is one of the best models to use when the dependent variable is binary. A logistic function, also known as a sigmoid function, is often used by the model to predict the value of an output (dependent variable), given an input (independent variables). As with the graph below, the function can be restricted so that the output can only be certain values.



In this case, the dependent variable is whether a given URL is malicious (1) or benign (0), and the independent variables are the characteristics of the URL. As logistic regression is easier to implement than other machine-learning models and very efficient to train, it would be an optimal choice.

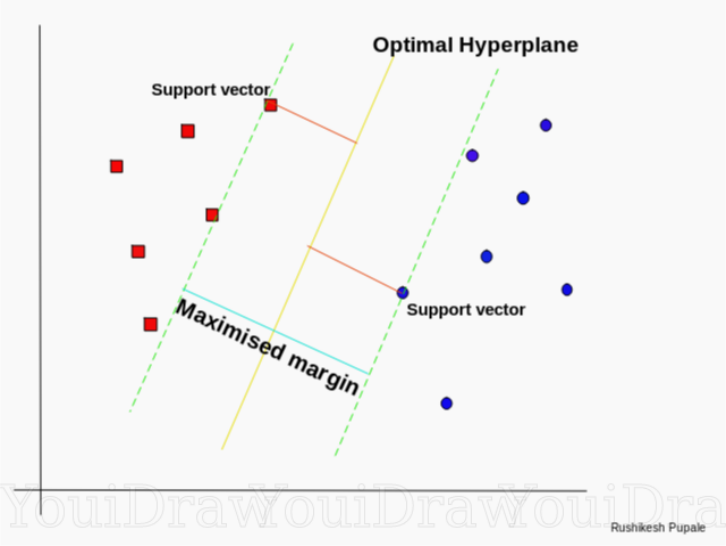
Python implementing the Logistic Regression model:

```
#Importing modules for Logistic Regression
import seaborn as sns
import statsmodels.api as sm
from pyspark.ml.classification import LogisticRegression, OneVsRest

#Instantiating, training, and testing the model
log_reg = LogisticRegression()
logModel = log_reg.fit(trainingData)
predictions = logModel.transform(testData)
predictions.show(10)

#Calculating and displaying the accuracy of the model
```

```
accuracy = predictions.filter(predictions.label ==
predictions.prediction).count() / float(predictions.count())
print("Accuracy : ",accuracy)

#Calculating the amount of true positives, true negatives, false positives,
and false negatives from the model
df = predictions.select('prediction', 'label')

tp = df[(df.label == 1) & (df.prediction == 1)].count() #True positive
tn = df[(df.label == 0) & (df.prediction == 0)].count() #True negative
fp = df[(df.label == 0) & (df.prediction == 1)].count() #False positive
fn = df[(df.label == 1) & (df.prediction == 0)].count() #False negative

#Calculating precision, recall, and F1 score
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f = 2 * (precision * recall)/(precision + recall)
print("Precision: ", precision)
print("Recall : ", recall)
print("F Score : ",f)
```

Support Vector Machine

Support vector machine (SVM) is a model best used to classify data into various categories. It works by finding a hyperplane in an N-dimensional space (N being the number of features) that classifies the data. The hyperplane is a decision boundary bounded by data points, or support vectors, from each group that helps the model differentiate the data. A larger margin for the hyperplane allows for less error when determining which side certain data points are on.



With the current dataset, URLs are classified as malicious or benign, each with their own characteristics. For determining whether a URL is malicious based on its characteristics, SVM would be a strong model.

Python implementing the Support Vector Machine model:

```
#Importing Support Vector Machine
from pyspark.ml.classification import LinearSVC

#Instantiating, training, and testing the model
lsvc = LinearSVC()
lsvcModel = lsvc.fit(trainingData)
svcpredict = lsvcModel.transform(testData)
svcpredict.show(10)

#Calculating and displaying the accuracy of the model
accuracy = svcpredict.filter(svcpredict.label ==
svcpredict.prediction).count() / float(svcpredict.count())
print("Accuracy : ",accuracy)

#Calculating the amount of true positives, true negatives, false positives,
and false negatives from the model
df = svcpredict.select('prediction', 'label')

tp = df[(df.label == 1) & (df.prediction == 1)].count() #True positive
tn = df[(df.label == 0) & (df.prediction == 0)].count() #True negative
fp = df[(df.label == 0) & (df.prediction == 1)].count() #False positive
fn = df[(df.label == 1) & (df.prediction == 0)].count() #False negative

#Calculating precision, recall, and F1 score
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f = 2 * (precision * recall)/(precision + recall)
print("Precision: ", precision)
print("Recall : ", recall)
print("F Score : ",f)
```

Results

The machine-learning models were given training data to find patterns, then attempted to predict the "labels" of URLs in the testing data. As shown in the equations on the right, accuracy is the fraction of how many URLs,

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

whether malicious or benign, were labeled correctly by the models out of all the testing data. Precision is how many malicious URLs were labeled correctly out of all the URLs labeled as malicious, and recall is how many malicious URLs were labeled correctly out of all URLs that were truly malicious. F1, also known as the F score, is a representation of a model's accuracy based on precision and recall. Since the models are being tested to detect malicious URLs, their precision, recall, and F score values would be more representative of their effectiveness than their accuracy.

Github Dataset Results

```
Accuracy :  0.9918903176665621
Precision:  0.9964371604168523
Recall :  0.987552966101695
F Score :  0.9919751718022612
```

Logistic Regression

```
Accuracy :  0.9929208297862807
Precision:  0.9977718360071302
Recall :  0.9882591807909604
F Score :  0.9929927266276387
```

Support Vector Machine

In the results from the Github dataset, SVM's accuracy, precision, recall, and F score were all higher than those of logistic regression. However, each value of logistic regression and SVM only had a difference less than or equal to 0.11%. Although SVM did perform better with this dataset, it cannot be concluded that it will always be more effective than logistic regression.

Kaggle Dataset Results

```
Accuracy :  0.9926899273477442
Precision:  0.9977708426214891
Recall :  0.9878177966101694
F Score :  0.9927693740850817
```

Logistic Regression

```
Accuracy :  0.992645080276258
Precision:  0.9969729344729344
Recall :  0.988524011299435
F Score :  0.9927304964539007
```

Support Vector Machine

In the results from the Kaggle dataset, logistic regression's accuracy, precision, and F score were only slightly higher than those of SVM. However, SVM's recall was higher. While precision is indicative of how many URLs were found in a dataset, recall expresses how many of the malicious URLs were found and classified correctly. This indicates that with this particular dataset, logistic regression performed better at identifying benign URLs, but SVM performed better at identifying malicious URLs.

References

[1] Underfitting versus overfitting graphs. https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

[2] Logistic regression and the sigmoid activation function. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[3] Support vector machine diagrams and explanation. https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[4] URL dataset from Github. https://github.com/rlilojr/Detecting-Malicious-URL-Machine-Learning/blob/master/dataset.csv

[5] URL dataset from Kaggle. https://www.kaggle.com/teseract/urldataset/data