Case Study Five

Allen Hoskins, Brittany Lewandowski

October 31, 2022

1 INTRODUCTION

Firewall data provides important information pertinent to system security. Specifically, it provides an overview of network traffic indicating activity that has been blocked from entering the network. This kind of data is important as it can be monitored to prevent cyber security attacks and monitored to ensure that networks are safe and secure.

To help ensure that networks are secure, machine learning can be leveraged to analyze firewall data. The subject of this case study is on building two classification models, (SVM and SGD), to identify whether specific firewall data will be blocked or allowed into a network. Our hope is that this research can be extrapolated to new firewall data sets, to identify and block malicious network attacks.

2 METHODS

***DATA UNDERSTANDING:***

Data used in this case study was stored in a CSV file called “log2,” and was obtained by Dr. Slater. The data set contains firewall data with columns including: “source\_port”, “action”, “packets”, and “bytes.” Although we received this data set from Dr. Slater, it can alternatively be accessed from the UCI Machine Learning Repository [here.](https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data) Exhibit 1.0 below highlights additional details about our data set’s shape and characteristics.

|  |  |  |
| --- | --- | --- |
| Total Number of Features | Total Number of Records | Response Variable |
| 12 | 65532 | “Action” |

Exhibit 1.0 – Firewall data set characteristics

***DATA PREPROCESSING:***

The first step we took in pre-processing was loading our data into a data frame, “log\_df” and viewing its general characteristics. From running the command “log\_df.info(),” we saw that our data set had no null values and contained columns with object and integer data types. Seeing that our data set contained categorical variables, we determined that one-hot encoding would be required before model building.

After reviewing our data set, we took a closer look at the records of our features to validate that each column’s data type was accurately identified. Upon doing so, we noticed that our “Port” features were misrepresented. Although our “Port” features had numeric records, these records were representative of classes, not numeric values. Since Python had originally classified these columns as integer data types, we converted them to object data types to reflect their true values. Features that were converted in this process included: “source\_port,” “destination\_port,” “nat\_source\_port,” and “nat\_destination\_port.”

Once we curated our full data set, the last step we took in pre-processing was separating our numeric and categorical features into two data frames: “num\_df,” and “cat\_df.” This separation was done in preparation for one-hot encoding. This spilt was done so that we did not one hot encode our numerical features.

***EXPLORATORY DATA ANALYSIS:***

Upon loading and cleaning our data, we began performing our Exploratory Data Analysis (EDA). First, we viewed the distribution of the variables in our data set (Exhibit 1.1) and found that the distribution of our data was non-normal. Given that we are dealing with firewall data, this was not surprising to us, and we decided to proceed with our analysis without performing any additional transformations to our data.

Chart

Description automatically generatedChart, histogram

Description automatically generatedChart, bar chart

Description automatically generatedChart, histogram

Description automatically generated

Exhibit 1.1 – Distribution of features in our log2.csv

Next, we wanted to assess if multicollinearity existed amongst variables in our data. To do this, we built a correlation plot, (Exhibit 1.2) and saw that while several columns had correlation coefficients > .95, no columns had a correlation coefficient equal to one. Given that removing columns can result in data loss, we decided not to remove any columns.

Graphical user interface

Description automatically generated

Exhibit 1.2 – Correlation Plot of Numeric Columns

3 MODEL PREPARATION

***RE-CODING CATEGORICAL COLUMNS:***

Seeing that our four “Port” categorical variables each contained 25 classes, we recognized that one-hot encoding would not be practical from a processing perspective. As an alternative, we identified records in these variables that had infrequent values, and assigned them to a new port, (999999). Once this was complete, we printed out the results using the following calculation “cat\_df[col].value\_counts()/len(cat\_df)\*100” to view the new counts for our categorical variables. This output was used to identify thresholds to scale down the number of classes that were in these variables. Any records that did not meet their defined threshold, were assigned to the class “Other.” Exhibit 1.3 below highlights the thresholds that were selected for each variable.

|  |  |  |
| --- | --- | --- |
| Feature Name | Selected Threshold | Explanation |
| source\_port | .00021 | ~ 88% of data falls into the top 4 classes |
| nat\_source\_port | .00004 | ~99% of data falls into the top 2 classes |
| destination\_port | .004 | ~73% of data falls into the top 5 classes |
| nat\_destination\_port | .1 | ~95% of data falls into the top 5 classes |

Exhibit 1.3 – Thresholds set for categorical columns to simplify OHE processing

***ONE HOT ENCODING:***

Once our categorical variables were re-coded, we utilized Pandas’ get dummies function to one-hot encode these variables. After this finished, we joined our one-hot encoded data to our numeric columns and obtained our full data set (“log\_final”). Before modeling, we viewed shape of our new data set and our final data frame had 65,532 records and 139 columns.

3 MODEL BUILDING

Since the subject of this case study was predicting the class of an internet connection request, machine learning was selected as an appropriate method. Additionally, given that our response variable, “Action” was categorical, classification algorithms were identified as the best models to run on our data. Given that our data set was large and that SVM and SGD(VowPalWabbit) are both powerful classifiers on high dimensional data, they were the algorithms that we used to build our models.

For reproducibility, we began modeling by setting our seed as 0. Next, we separated our feature columns from our response and used an 80/20 train/test split to get our training and test data. Finally, to ensure that our data was on the same scale, we used a standard scaler to scale our data.

The first model we fit was SVM. To ensure that we fit the best SVM model, we fit models with linear and polynomial kernels with the following hyperparameters (Exhibit 1.4 & 1.5).

SVM Hyperparameters Fit With A Linear Kernel:

|  |  |
| --- | --- |
| Parameter | Values Tested For Parameter |
| C | 0.3,0.4,0.5,0.6,0.7 |
| cache\_size | 2000 |

Exhibit 1.4– SVM Hypers. With Linear Kernel

SVM Hyperparameters Fit with A Polynomial Kernel:

|  |  |
| --- | --- |
| Parameter | Values Tested For Parameter |
| C | 0.3,0.4,0.5,0.6,0.7 |
| cache\_size | 2000 |

Exhibit 1.5– SVM Hypers. With Polynomial Kernel

The process for fitting our VowPalWabbit model was similar to our SVM model. First, we re-coded our response variable values from: “allow,” “action,” “drop,” and “reset-both” to “1”, “2”, “3”, “4” as VowPalWabbit needs numerical response variables. Next, we separated our feature columns from our response, and used an 80/20 train test split to get our training and test data. Finally, to convert our data frame object to the correct VowPalWabbit format, we used the function “DFtoVW” from the VowPalWabbit module to get our final train and test data objects.

For building our VowPalWabbit model, we tested a variety of regularization values. Exhibit 1.6 shows the full details for the hyperparameters we tested in our model.

VPW Hyperparameters:

|  |  |
| --- | --- |
| Parameter | Values Tested For Parameter |
| L2 | 1, .1, .001, .0001, .00001, .0000001 |
| P | 10 |
| loss\_function | Hinge, linear |
| oaa | 4 |
| learning\_rate | 0.5 |

Exhibit 1.6– VPW Hypers

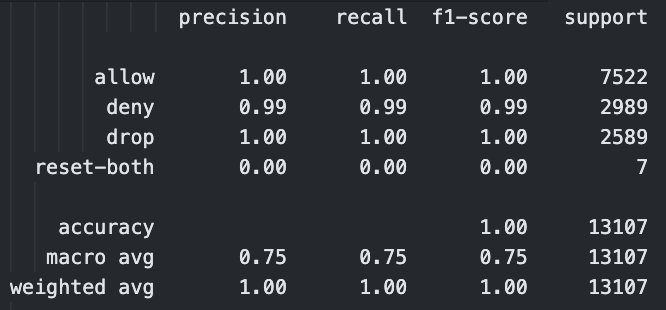
5 RESULTS

Exhibits 1.7-1.9 below highlight how our SVM and VPW models performed. The results of our SVM models indicate that using a linear kernel is more effective at classifying our response. This is evident when comparing classification reports of our best linear and polynomial kernel models (Exhibits 2.0 & 2.1).

SVM Linear Kernel Model Results:

|  |  |
| --- | --- |
| Model & Parameters | Exhibit |
| Model 1: C=0.3 | 1.7.1 |
| Model 2: C=0.4 | 1.7.2 |
| Model 3: C=0.5 | 1.7.3 |
| Model 4: C=0.6 | 1.7.4 |
| Model 5: C=0.7 | 1.7.5 |

Exhibit 1.7– SVM Linear Kernel Model Results

 A screenshot of a computer

Description automatically generated with low confidence

Exhibit 1.7.1 Exhibit 1.7.2

A screenshot of a computer

Description automatically generated with low confidence Calendar

Description automatically generated

Exhibit 1.7.3 Exhibit 1.7.4

Calendar

Description automatically generated

Exhibit 1.7.5

SVM Polynomial Kernel Model Results:

|  |  |
| --- | --- |
| Model & Parameters | Exhibit |
| Model 1: C=0.3 | 1.8.1 |
| Model 2: C=0.4 | 1.8.2 |
| Model 3: C=0.5 | 1.8.3 |
| Model 4: C=0.6 | 1.8.4 |
| Model 5: C=0.7 | 1.8.5 |

Exhibit 1.8– SVM Polynomial Kernel Model Results

Calendar

Description automatically generated Calendar

Description automatically generated

Exhibit 1.8.1 Exhibit 1.8.2

A screenshot of a computer

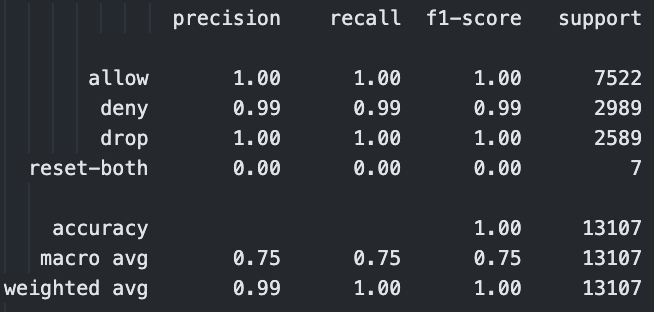
Description automatically generated with low confidence 

Exhibit 1.8.3 Exhibit 1.8.4

Calendar

Description automatically generated

Exhibit 1.8.5

VPW Model Results:

|  |  |
| --- | --- |
| Model & Parameters | Exhibit |
| Model 1: L2=1 | 1.9.1 |
| Model 2: L2=.1 | 1.9.2 |
| Model 3: L2=.001 | 1.9.3 |
| Model 4: L2=.0001 | 1.9.4 |
| Model 5: L2=.00001 | 1.9.5 |
| Model 6: L2=.000001 | 1.9.6 |

Exhibit 1.9– VPW Model Results

A screenshot of a computer

Description automatically generated with low confidence Calendar

Description automatically generated

Exhibit 1.9.1 Exhibit 1.9.2

Calendar

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence

Exhibit 1.9.3 Exhibit 1.9.4

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

Exhibit 1.9.5 Exhibit 1.9.6

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Exhibit 2.0– Best SVM Linear Kernel Classification Report

Exhibit 2.1– Best SVM Polynomial Kernel Classification Report

Graphical user interface, application, Teams

Description automatically generated

Exhibit 2.2– Best VPW Classification Report

When comparing our best SVM model to our best VPW model, we saw that VPW outperformed SVM. This conclusion was made by comparing Exhibits 2.0 and 2.2 as well as the classification reports shown above. While VPW was not fine-tuned utilizing all its hyperparameters, it was able to successfully predict the fourth “action” category while SVM was unable to do so.

6 CONCLUSION

This case study showed us how we can work with and model larger data sets. Specifically, it reminded us how we can scale down classes of categorical variables by re-coding them with thresholds and it demonstrated how using higher powered classification algorithms such as VPW can help to quickly classify large data sets.

Both the SVM and VPW algorithms, produced great accuracy for the first 3 “action” responses, however both struggled to predict the fourth, which appeared very infrequent in the data set. This shows that models are not perfect and the training data split is critical in predicting all expected outcomes.

One lesson learned in this case study was that although SVM is a good classification algorithm, complexity is introduced when trying to fit it to larger data sets. Running VPW in conjunction with SVM here showed us that we can get a better model with algorithms that require less processing power.

In addition to utilizing different classification algorithms, the results of the VPW model further show that while exponentially decreasing the l2 regularization, there is a threshold where additional parameter tuning does not increase the accuracy or F1 score of the model.

If we had additional time to allot to this case study, we would focus on tuning other hyperparameters in VPW. The VPW documentation had a great level of hyperparameters that we could tune, however the focus on this study was on tuning our l2 regularization hyperparameter.