Case Study 5 Report:

Firewall Automation

Alexy Morris and Domicia Herring

October 31, 2022

Abstract

The team has been provided historical data from a major cybersecurity company to create an algorithm to accept or deny access to someone trying to gain access to a site. Currently, the company is utilizing an absorbent amount of resources to do this manually and would like to automate it. The historical data details how the company has chosen to accept or deny requests in the past. The model will be optimized to filter incoming requests, classify the request in whether they will be accepted or denied, then automatically accept or deny the request based on the classification given.

1. Introduction

1.1 Business Understanding

Firewalls are an important part of our daily lives.Whether at home or work, on a laptop or phone, firewalls are constantly running in the background analyzing the network traffic to block unwanted pop-ups and malicious attacks on your operating systems. For example, not everyone is allowed into your home. A subset of people, source addresses, are allowed to enter your home, a destination address. If you are not within that subset of people, an alarm is triggered to the owner and your access is denied. But if you are within that subset of people, you are granted access to the home.This is how a firewall functions to protect us.

Due to the high amount of cybersecurity threats operating systems encounter everyday, the accuracy of one's firewall is of high importance. This case study's purpose is to build a classifier that automatically approves or denies a request to enter a site.

1.2 Data Meaning Type

To create an algorithm that predicts the factors for whether an individual is permitted to access a site or not, historical data of past requests collected from UCI Machine Learning Repository will be utilized. The dataset comprised 65,531 records and 12 features is depicted in Figure 1.2a. In Figure 1.2b, the actions that were taken on the requests are described.

Feature Names	Description
source_port	Identifies where the host data is sent from
destination_port	Identifies where the data is sent to
nat_source_port	Translates destination addresses and packets
nat_destination_port	Translates private Ips into public Ips
Action	Actions taken by firewall
Bytes	Number of bytes
bytes_sent	Number of bytes sent
bytes_received	Number of bytes received
Packets	Number of packets
elapsed_time	Time elapsed with process
pkts_sent	Number of packets sent
pkts_received	Number of packets received

Figure 1.2a- Feature Descriptions

Action	Description
Allow	Access approved
Deny	Access Denied
Drop	Notify destination party that device is denied and drop connecton on the source side
Reset-both	Resets both side

Figure 1.2b- Action Descriptions

2. Methods

2.1 Data Preprocessing

The data originally consisted of one dataset with 65,531 records. To ensure the data was easy to handle, the two word column headers were transformed into one word by filling the space with “_”. The IP addresses were transformed from int64 to str. The data was checked for null values and the dataset did not have any. The dataset required little cleaning, therefore the team moved onto the exploratory data analysis.

2.2 Exploratory Data Analysis

Exploring the data, the team saw that it consisted of 37,640 requests allowed, 14,987 requests denied, 12,85 dropped and 54 reset-both as depicted in Figure 2.2a and Figure 2.2b.

```
allow         37640
deny          14987
drop          12851
reset-both       54
Name: Action, dtype: int64
```

Figure 2.2a- Action Breakdown

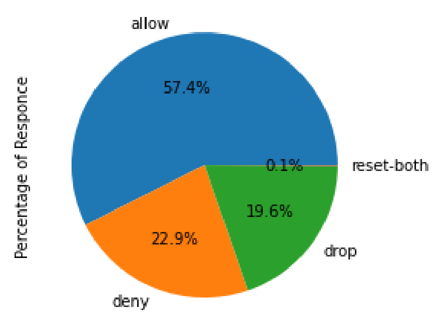


Figure 2.2b- Action Pie Graph

With “reset-both” only making up 0.1% of the dataset, the team decided to drop the class.The result is that the data makeup is 57.5% allowed requests, 22.9% denied requests and 19.6% drop requests as shown in Figure 2.2.c.

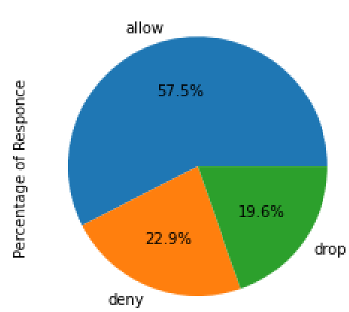


Figure 2.2c-Action Pie Graph After Drop of Reset-both

While examining the correlation matrix, the team saw very high correlations which lead to our deletion of certain features. The features Bytes and Packets were both deleted for being composite attributes and they were deemed less valuable than the attributes they were calculated from. The columns pkts_sent and pkts_recieved were also deleted for the massive overlap between the two features and the bytes attributes.

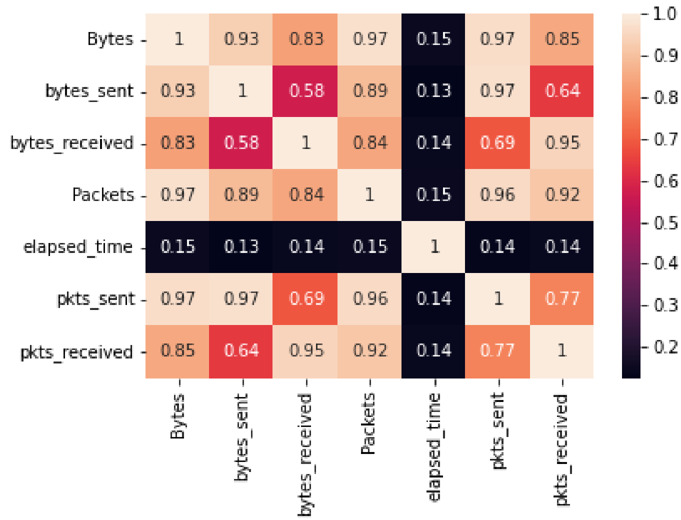


Figure 2.2c-Action Pie Graph After Drop of Reset-both

Next, the team split our dataset into data/target train and test subsets using the train_test_split function from the sklearn model_selection package. Then, the data was standardized using StandardScaler. Then, the model was ready to run.

2.3 Models

2.3.1 Support Vector Machine Model

By utilizing the classification based Support Vector Machine, the team was able to create a very good model using the baseline settings. Although the initial model was satisfactory, the team decided that further experimentation with the parameters was needed for a better view into the data. The first parameter tested was the gamma parameter; gamma was set to auto which in turn dropped all averages in the classification report. From there, the team ran many more models experimenting with the kernel, gamma and degree. The best model found after trial and error utilized the polynomial kernel, auto gamma and was 3rd degree. That said, the team still saw big drops in precision, recall, and f1-score of drop/deny.

	precision	recall	f1-score	support
allow	1.00	0.97	0.99	11192
deny	0.99	0.96	0.97	4508
drop	0.90	1.00	0.95	3944
accuracy			0.98	19644
macro avg	0.96	0.98	0.97	19644
weighted avg	0.98	0.98	0.98	19644

Figure 2.3a-Classification Report of Tuned SVM Model

For this reason, the team has decided to use the default model as our best model. Because of the success with the Radial Basis Function kernel the team is interested in examining the data with the mathematically similar K-Nearest Neighbours model.

2.2.2 Stochastic Gradient Descent Model

For the classification based Stochastic Gradient Descent Model, the team decided to follow the same method creating a baseline, default model and then doing experimental tuning. Once again the default setting does produce a solidly good model. Then, the team examined the max_iter and tol parameters but found that the default numbers worked best for our purposes. Moving on to the loss and penalty parameters, the team discovered that the penalty l1 worked the best for our focus. Thus, declaring the last model the best.

3. Results

3.1 Support Vector Machine Model

As shown in Figure 3.1a, the chosen SVM model shows high scores across the board. This is especially impressive in the cases of the deny and drop class. These classes did have less support in the model, therefore the high scores tell us that the model does not overcompensate for the dominant class of allow. The model has a 99% accuracy, which indicates a near perfect classification rate. The average also shows a well developed model.

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	11192
deny	0.99	0.97	0.98	4508
drop	0.96	1.00	0.98	3944
accuracy			0.99	19644
macro avg	0.98	0.99	0.99	19644
weighted avg	0.99	0.99	0.99	19644

Figure 3.1a-Classification Report of Best SVM Model

3.2 Stochastic Gradient Descent Model

The chosen SGD model also shows very high scores across the board. Shown in Figure 3.2a, it actually does better than the SVM model in terms of allow-recall, drop-precision, and drop-f1 score . This does cause a slight change in raising the macro average for precision by 1%. Based on that, this model is the overall best model.

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	11192
deny	0.99	0.97	0.98	4508
drop	0.97	1.00	0.98	3944
accuracy			0.99	19644
macro avg	0.99	0.99	0.99	19644
weighted avg	0.99	0.99	0.99	19644

Figure 3.2a-Classification Report of Best SGD Model

4. Conclusion

The protection of the assets one holds on technology can be paramount to their way of life. Ensuring that one is protected properly is the best way to prevent cyberattacks. A highly efficient firewall can make the difference.

The team was able to classify internet firewall requests based on various factors. The Support

Vector Machine model yielded an accuracy of 99%, precision of 98% and a recall of 99%. Also, the Stochastic Gradient Descent model yielded an accuracy of 99%, precision of 99% and a recall of 99%. Therefore, the team can conclude that the Stochastic Gradient Descent model will efficiently classify any new internet requests made.

5. Citation

5.1. "What Is a Firewall?" Forcepoint, 24 Oct. 2022, https://www.forcepoint.com/cyber-edu/firewall.

5.2. Written by Alison Grace Johansen for NortonLifeLock. "What Is a Firewall? Firewalls Explained and Why You Need One." *Norton*, https://us.norton.com/blog/emerging-threats/what-is-firewall#.

6. Code

MSDS7333 (/github/Alexy-Mor/MSDS7333/tree/main)
/ case study 5 (/github/Alexy-Mor/MSDS7333/tree/main/case study 5)

In [ ]:

```
#import packages
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.feature_selection import SelectPercentile,f_regression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.utils import resample
from sklearn.metrics import accuracy_score, precision_score, recall_score
#from sklearn.inspection import DecisionBoundaryDisplay
```

In [ ]:

```
#remove warnings after verifying code
warnings.filterwarnings("ignore")
```

In [ ]:

```
data = pd.read_csv(r'log2.csv',low_memory = False)
```

In [ ]:
```
data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65532 entries, 0 to 65531
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Source Port           65532 non-null  int64
 1   Destination Port      65532 non-null  int64
 2   NAT Source Port       65532 non-null  int64
 3   NAT Destination Port  65532 non-null  int64
 4   Action                65532 non-null  object
 5   Bytes                 65532 non-null  int64
 6   Bytes Sent            65532 non-null  int64
 7   Bytes Received        65532 non-null  int64
 8   Packets               65532 non-null  int64
 9   Elapsed Time (sec)    65532 non-null  int64
 10  pkts_sent             65532 non-null  int64
 11  pkts_received         65532 non-null  int64
dtypes: int64(11), object(1)
memory usage: 6.0+ MB
```

In [ ]:
```
data.head()
```

Out[ ]:

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapse Tim (se
0	57222	53	54587	53	allow	177	94	83	2	3
1	56258	3389	56258	3389	allow	4768	1600	3168	19	1
2	6881	50321	43265	50321	allow	238	118	120	2	119
3	50553	3389	50553	3389	allow	3327	1438	1889	15	1
4	50002	443	45848	443	allow	25358	6778	18580	31	1

In [ ]:
```
data = data.rename(columns={"Source Port": "source_port", "Destination Port'
```

In [ ]:
```
#basically addresses can't be int64
data["source_port"] = data["source_port"].astype(str)
data["destination_port"] = data["destination_port"].astype(str)
data["nat_source_port"] = data["nat_source_port"].astype(str)
data["nat_destination_port"] = data["nat_destination_port"].astype(str)
```

In [ ]:
```
data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65532 entries, 0 to 65531
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   source_port           65532 non-null  object
 1   destination_port      65532 non-null  object
 2   nat_source_port       65532 non-null  object
 3   nat_destination_port  65532 non-null  object
 4   Action                65532 non-null  object
 5   Bytes                 65532 non-null  int64 
 6   bytes_sent            65532 non-null  int64 
 7   bytes_received        65532 non-null  int64 
 8   Packets               65532 non-null  int64 
 9   elapsed_time          65532 non-null  int64 
 10  pkts_sent             65532 non-null  int64 
 11  pkts_received         65532 non-null  int64 
dtypes: int64(7), object(5)
memory usage: 6.0+ MB
```

In [ ]:
```
data.describe()
```

Out[ ]:

	Bytes	bytes_sent	bytes_received	Packets	elapsed_time	pkts_sen
count	6.553200e+04	6.553200e+04	6.553200e+04	6.553200e+04	65532.000000	65532.00000(
mean	9.712395e+04	2.238580e+04	7.473815e+04	1.028660e+02	65.833577	41.39953(
std	5.618439e+06	3.828139e+06	2.463208e+06	5.133002e+03	302.461762	3218.87128ℓ
min	6.000000e+01	6.000000e+01	0.000000e+00	1.000000e+00	0.000000	1.00000(
25%	6.600000e+01	6.600000e+01	0.000000e+00	1.000000e+00	0.000000	1.00000(
50%	1.680000e+02	9.000000e+01	7.900000e+01	2.000000e+00	15.000000	1.00000(
75%	7.522500e+02	2.100000e+02	4.490000e+02	6.000000e+00	30.000000	3.00000(
max	1.269359e+09	9.484772e+08	3.208818e+08	1.036116e+06	10824.000000	747520.00000(

In [ ]:
```
#check for null values
data.isnull().sum()
```

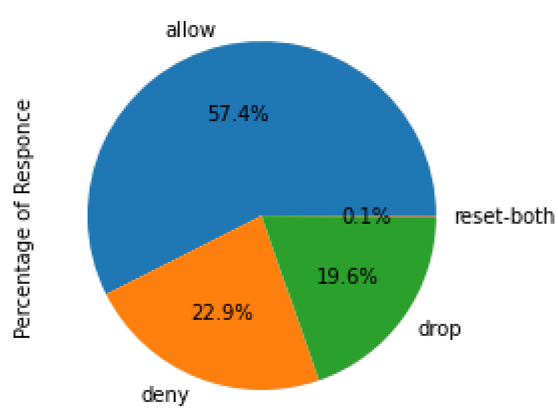
Out[ ]:
```
source_port             0
destination_port        0
nat_source_port         0
nat_destination_port    0
Action                  0
Bytes                   0
bytes_sent              0
bytes_received          0
Packets                 0
elapsed_time            0
pkts_sent               0
pkts_received           0
dtype: int64
```

In [ ]:
```
#view unique data in column 'class'
data['Action'].unique()
```

Out[ ]:
```
array(['allow', 'drop', 'deny', 'reset-both'], dtype=object)
```

In [ ]:
```
#Visualize the data
print(data["Action"].value_counts())

data.groupby('Action').size().plot(kind='pie',
                                       y = "Action",
                                       label = "Percentage of Responce",
                                       autopct='%1.1f%%');
```

```
allow         37640
deny          14987
drop          12851
reset-both       54
Name: Action, dtype: int64
```

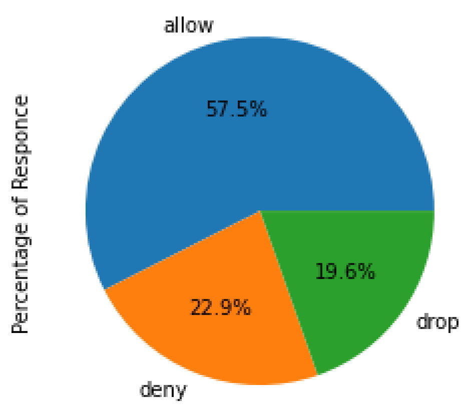


In [ ]:
```
data = data[data.Action != 'reset-both']
```

In [ ]:

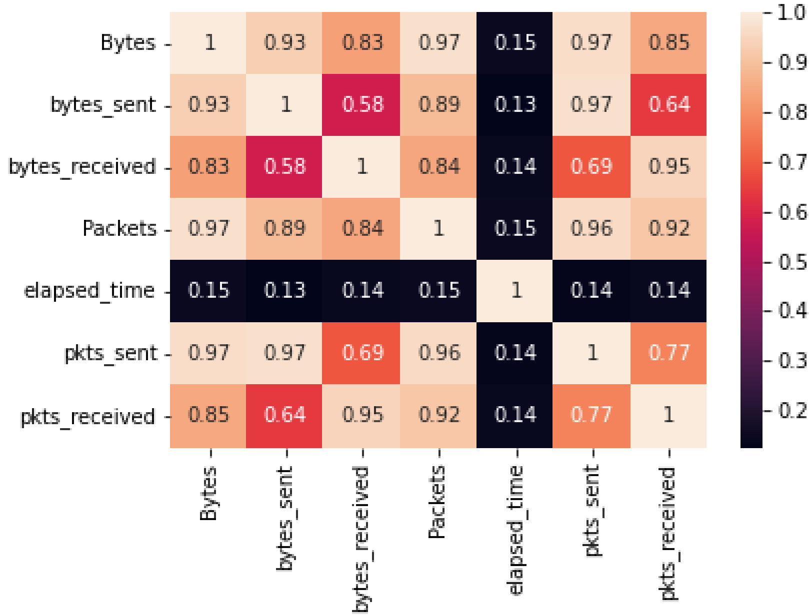
```
#Visualize the data
print(data["Action"].value_counts())

data.groupby('Action').size().plot(kind='pie',
                                   y = "Action",
                                   label = "Percentage of Responce",
                                   autopct='%1.1f%%');
```

```
allow    37640
deny     14987
drop     12851
Name: Action, dtype: int64
```



In [ ]:

```
#Delete Packets, Bytes, Bytes_recieved
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



In [ ]:

```
data = data.drop(data.columns[[5, 8, 10, 11]], axis=1)
```

In [ ]:

```
x = data.loc[:,data.columns !='Action'] # Features
y = data['Action']  # Labels
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,ranc
```

In [ ]:

```
#target_names = ['allow', 'deny', 'drop', 'reset-both']
target_names = ['allow', 'deny', 'drop']
```

In [ ]:

```
svcm0 = make_pipeline(StandardScaler(), SVC())
svcm0.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()), ('svc', SVC())])
```

In [ ]:

```
Pred0 = svcm0.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred0, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00     11192
        deny       0.99      0.97      0.98      4508
        drop       0.96      1.00      0.98      3944

    accuracy                           0.99     19644
   macro avg       0.98      0.99      0.99     19644
weighted avg       0.99      0.99      0.99     19644
```

In [ ]:

```
svcm1 = make_pipeline(StandardScaler(), SVC(gamma='auto'))
svcm1.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

In [ ]:

```
Pred1 = svcm1.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred1, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00     11192
        deny       0.99      0.97      0.98      4508
        drop       0.96      1.00      0.98      3944

    accuracy                           0.99     19644
   macro avg       0.98      0.99      0.99     19644
weighted avg       0.99      0.99      0.99     19644
```

In [ ]:

```
#sig kernel/auto gammma - across board metris down
#sig kernel/scale gammma - sligthly better
#poly kernel/auto gammma/degree = 3 - better!
#poly kernel/auto gammma/degree = 4 - not as good!
#poly kernel/auto gammma/degree = 5 - HORRID
svcm2 = make_pipeline(StandardScaler(), SVC(kernel = 'poly', degree = 3, gam
svcm2.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(kernel='poly'))])
```

In [ ]:

```
Pred2 = svcm2.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred2, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      0.97      0.99     11192
        deny       0.99      0.96      0.97      4508
        drop       0.90      1.00      0.95      3944

    accuracy                           0.98     19644
   macro avg       0.96      0.98      0.97     19644
weighted avg       0.98      0.98      0.98     19644
```

In [ ]:

```
sgdm0 = make_pipeline(StandardScaler(),
                     SGDClassifier())
sgdm0.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdclassifier', SGDClassifier())])
```

In [ ]:

```
Pred0_5 = sgdm0.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred0_5, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      0.99      1.00     11192
        deny       0.99      0.96      0.97      4508
        drop       0.95      1.00      0.97      3944

    accuracy                           0.99     19644
   macro avg       0.98      0.98      0.98     19644
weighted avg       0.99      0.99      0.99     19644
```

In [ ]:

```
sgdm1 = make_pipeline(StandardScaler(),
                     SGDClassifier(max_iter=1000, tol=1e-3))
sgdm1.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdclassifier', SGDClassifier())])
```

In [ ]:

```
Pred3 = sgdm1.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred3, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      0.99      1.00     11192
        deny       0.99      0.95      0.97      4508
        drop       0.94      1.00      0.97      3944

    accuracy                           0.99     19644
   macro avg       0.98      0.98      0.98     19644
weighted avg       0.99      0.99      0.99     19644
```

In [ ]:

```
#loss ='log', max_iter=1000, tol=1e-3 - precision drop
#loss ='hinge', max_iter=1000, tol=1e-3, penalty =  'l1' - all scores went u
#loss ='hinge', max_iter=1000, tol=1e-3, penalty =  'elasticnet' - mid
sgdm2 = make_pipeline(StandardScaler(),
                     SGDClassifier(loss ='hinge', max_iter=1000, tol=1e-3, p
sgdm2.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdclassifier', SGDClassifier(penalty='l1'))])
```

In [ ]:

```
Pred4 = sgdm2.predict(X_test)
```

In [ ]:

```
#metrics
print(classification_report(y_test, Pred4, target_names=target_names))
```

```
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00     11192
        deny       0.99      0.97      0.98      4508
        drop       0.97      1.00      0.98      3944

    accuracy                           0.99     19644
   macro avg       0.99      0.99      0.99     19644
weighted avg       0.99      0.99      0.99     19644
```