Cyber Security Case Study 5

Sanjay Pillay - Wednesday, Oct 17, 2021

**Abstract**

The report investigates the feasibility of using machine learning algorithms to automate firewall security to prevent malicious internet access to a company’s network behind the firewall using historical network logs with very high degree of accuracy.

# Introduction

This is a multiclass case study to predict if an incoming request into the network behind a fire wall should be allowed or not based on historical data of key attributes that identify if the request was malicious or legitimate.

With the explosion of internet connectivity, the volume of network traffic has grown exponentially increasing the threat of malicious activity by unknow sources penetrating the network infrastructure posing significant risk to an organizations business and reputation. Such attacks are mitigated using a combination of hardware and software device called firewall. A software or firmware device called firewall prevents un-authorized access to a network that sits behind the firewall, it prevents such access by inspecting the network packets, the source and destination of the request using specified rules and dropping such requests. Firewalls are generally deployed at the organizations perimeter to prevent mainly external illegitimate sources from gaining access to the network [2].

With the increase in volume of network interactions maintaining the firewall rules gets unmanageable, one way to deal with this issue is to use machine learning techniques and build models that can predict if a network request should be allowed or not learning from historical data captured so far via network logs [4].

In this report we use data generated from network logs [2.1] to classify an incoming request into the following three classifications “Allow”, “Deny” or “Drop” based on the features sets identified in the data.

# Methods

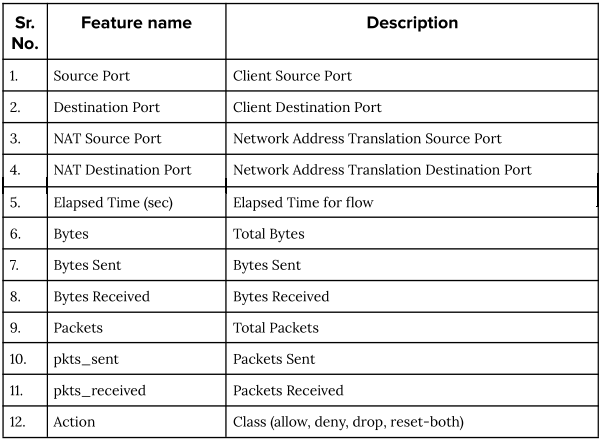
The labeled data from network logs [2.1] was analyzed, scaled using standard scalars and used to build two models, the first model used Support Vector Machine (SVM) and the second one was SVM based linear classifier using stochastic gradient decent (SGD). The models were evaluated for accuracy using f1 score of the model, the confusion matrix was also evaluated.

We first made a single stratified shuffle split of 80/20 % Train/Test to keep same class balance. The models were trained on the 80% Train split with 3-fold cross validation with an internal stratified shuffle split on this training data, best tuning parameters were identified using grid search. The best model identified by grid search was used to calculate f1 score and Confusion Matrix (CM) on the 20% test split that was held back. Due to the size of the data and features sets we limited our cross validation to 3. Later we also discuss additional techniques that can be used to reduce processing times for such larger datasets.

## Data

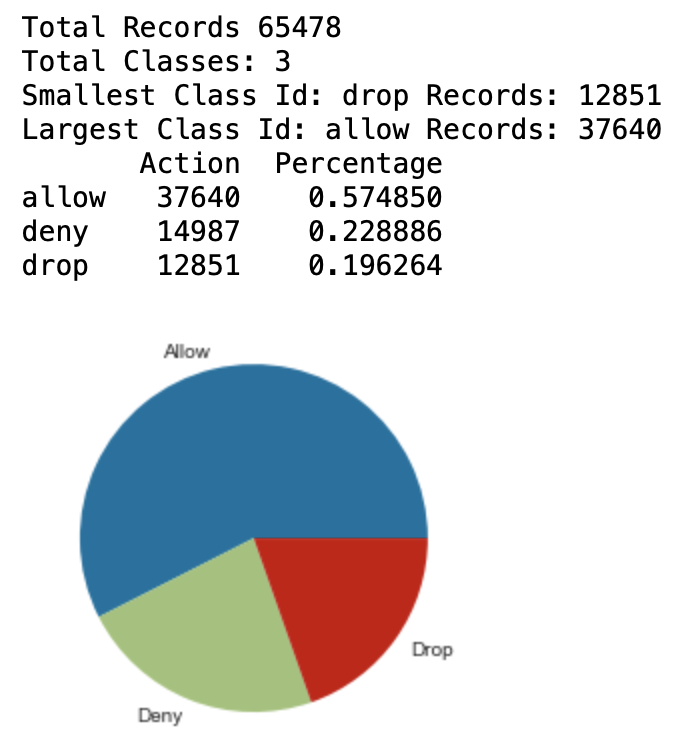
The historical data collected consists of eleven attributes and the label (“Action”) shown in [Table 1 Attributes] and a total of 75478 rows, there were no missing values. The label had four values allow, deny, drop and reset-both, since there were only 54 records for reset-both we chose to drop these records making this a three-class problem.

Table Attributes [Borrowed from 4]



The target class distribution [Figure 1] shows that the class ‘allow’ has 57% records while ‘deny’ and ‘drop’ are almost evenly distributed at 22 and 19 % respectively. We will use balanced class weights for our algorithms.

Figure



The attributes “Source Port”, “Destination Port”, “NAT Source Port” and “NAT Destination Port” although integer values from 0 to 65535 identifying port ranges were “one hot encoded” to be treated as categorical values adding 57628 additional features, the final dataset consists of 67635 features and 65476 rows, this data was scaled using standard scalar.

Since sklearn library accepts multiclass labels as strings for the models used, we did not use any label encoder for the targets.

## Models

### SVM (Linear)

The first model we tried was SVM. The tuning for the svm was done using gridsearch. The dataset we have (65476\*67635) is considered to be large for svm algorithms although most of the features are sparse so we limited the gridsearch parameters to only use few selected tuning parameters and using a linear kernel so the algorithm could complete on the available resource. We used 80% of the data to tune the model using 3-fold cross validation with stratified shuffle split and calculated the accuracy, confusion Matrix (CM) and f1 scores using the remaining 20% test data. The best parameters were {'C': 90, 'class\_weight': 'balanced', ‘loss’: ‘hinge’} and defaults for other parameters.

### SGDClassifier (Linear Classifier)

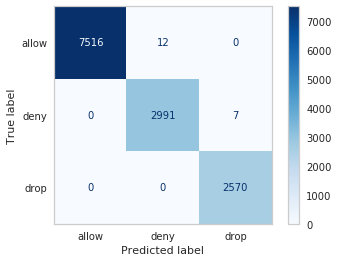
The second model we tried was SGDClassifier. The tuning for the SGDClassifier was done using gridsearch. The advantage of an SGD classifier is the model allows you to do a partial fit where you can load data in chunks if there are significant computing resource issues, since we had enough memory available, we opted to load the entire dataset into the model for evaluation. We used 80% of the data to tune the model using 3-fold cross validation with stratified shuffle split and calculated the accuracy, confusion Matrix (CM) and f1 scores using the remaining 20% test data. The best parameters were {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'log'} and defaults for other parameters.

# Results

### SVM (Linear)

The overall accuracy score for the SVM model using hinge loss was pretty high at 99.85, the confusion matrix [Figure 2] derived using the 20% test data shows that the individual f1 score of each of the three classes was also very high close to 100%, only 12 requests out of 7528 were categorized as ‘deny’ which should have been ‘allow’ and 7 out of 2998 of the ‘deny’ request were ‘drop’. Table 2 list the class and model f1 scores.

Figure



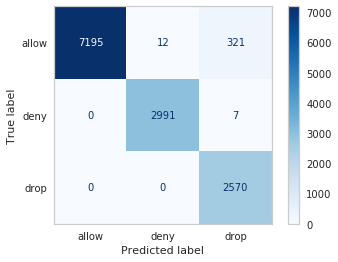
Table

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **f1** |
| allow | 1.0 | 0.96 | 0.98 |
| deny | 1.0 | 1.00 | 1.0 |
| drop | 0.89 | 1.0 | 0.94 |
| Model |  |  | 0.97 |

### SGDClassifier (Linear Classifier)

For the SGDClassifer model using log loss the overall f1 accuracy is 97.4, with the f1 score for ‘allow’ being 98%, ‘deny’ almost 100% and ‘drop’ at 94% [Table 3]. This model has a slight bit of lesser accuracy than SVM. Figure 3 shows the confusion matrix for the model derived from the 20% test data.

Figure



Table

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **f1** |
| allow | 1.0 | 0.96 | 0.98 |
| deny | 1.0 | 1.00 | 1.0 |
| drop | 0.89 | 1.0 | 0.94 |
| Model |  |  | 0.97 |

# Conclusion

The SVM classifier using a linear kernel gave is a very good accuracy and had a reasonable processing time to train the model on this data set, but if the data set grows more SVM models are prone to slow down or run out of memory. The SGD classifier took a lot longer to train as we loaded the entire dataset into memory to train the model, but when we used out of core method to train the model in data chunks using “partialfit” it was much faster [Table 4]. Although the SGDClassifier is a bit less accurate than SVM it has the advantage of out of core memory training if the dataset was to increase significantly. Using another out of core library such as “vowpal wabbit” we can significantly increase the processing time for training as it reduces the sparse feature matrix significant and it also uses a good hashing algorithm to overcome the issue introduced by out of core training of loading data quickly into memory.

To increase the model accuracy further we could add four additional features for the ports as follows: ports within ranges of 0 to1023 as ‘well-known’ ports, 1024 to 49151 as ‘registered’ ports and ‘dynamic’ ports for range in 49152 to 65535 [23].

Table

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **Time** |
| SVM | 99.85 | 1min 50s |
| SGDClassifier | 97.4 | 9min 27s |
| SGDClassifier (out of core) | 92.5 | 1min 10s |

# Appendix

## Code

**Some of the output has been cleaned to reduce document.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **import** **os**  **import** **email**  *#All Python module imports*  *#https://pandas.pydata.org/docs/user\_guide/index.html#user-guide*  **import** **pandas** **as** **pd** *#Pandas Dataframe module*  **from** **imblearn.over\_sampling** **import** SMOTE  **import** **numpy** **as** **np**  **from** **math** **import** pi  *#scikit learn*  *#https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear\_model*  **import** **sklearn** **as** **skl**  *#https://seaborn.pydata.org*  **from** **yellowbrick.model\_selection** **import** FeatureImportances  **import** **seaborn** **as** **sns**  **import** **matplotlib.pyplot** **as** **plt**  **import** **matplotlib**  **import** **warnings**  *#Module for formating table for documentation*  *#https://pypi.org/project/tabulate/*  **from** **tabulate** **import** tabulate  **from** **IPython.display** **import** display, Markdown  *#Interactive mode*  **from** **IPython.core.interactiveshell** **import** InteractiveShell  InteractiveShell.ast\_node\_interactivity = "all"  **from** **IPython.display** **import** Image  **from** **sklearn.preprocessing** **import** MinMaxScaler  **from** **sklearn.feature\_selection** **import** SelectKBest, chi2  **from** **sklearn.model\_selection** **import** StratifiedShuffleSplit  **from** **sklearn.preprocessing** **import** StandardScaler  **from** **sklearn.linear\_model** **import** LogisticRegression  **from** **sklearn** **import** metrics **as** mt  **from** **sklearn.metrics** **import** plot\_confusion\_matrix  **from** **sklearn.model\_selection** **import** cross\_val\_score  **from** **sklearn.metrics** **import** classification\_report  **from** **sklearn.linear\_model** **import** LogisticRegression  **from** **sklearn.svm** **import** SVC  **from** **sklearn.decomposition** **import** PCA  **from** **sklearn.metrics** **import** confusion\_matrix  **from** **sklearn.metrics** **import** f1\_score, accuracy\_score  **from** **sklearn.model\_selection** **import** KFold, StratifiedKFold  **from** **sklearn.model\_selection** **import** GridSearchCV **as** gridcv  **from** **sklearn** **import** preprocessing  **from** **sklearn.model\_selection** **import** cross\_validate  **from** **sklearn.metrics** **import** make\_scorer  **from** **sklearn.metrics** **import** mean\_squared\_error  **from** **sklearn.metrics** **import** mean\_absolute\_error  **from** **sklearn.metrics** **import** r2\_score  **import** **pprint**  **import** **re**  **from** **sklearn.model\_selection** **import** cross\_val\_predict  **from** **html.parser** **import** HTMLParser  **from** **bs4** **import** BeautifulSoup  **import** **nltk**  **from** **nltk.corpus** **import** stopwords  **from** **sklearn.feature\_extraction.text** **import** TfidfVectorizer  **from** **sklearn.metrics** **import** roc\_curve  **from** **sklearn.metrics** **import** roc\_auc\_score  **from** **scipy.io** **import** arff  **from** **statsmodels.imputation** **import** mice  **import** **statsmodels** **as** **sm**  **from** **xgboost** **import** XGBClassifier  **from** **numpy** **import** arange  **from** **numpy** **import** argmax  **from** **sklearn.preprocessing** **import** QuantileTransformer  In [2]:  df = pd.read\_csv('./log2.csv')  df.shape  df.head()  Out[2]:  (65532, 12)  Out[2]:   |  | **Source Port** | **Destination Port** | **NAT Source Port** | **NAT Destination Port** | **Action** | **Bytes** | **Bytes Sent** | **Bytes Received** | **Packets** | **Elapsed Time (sec)** | **pkts\_sent** | **pkts\_received** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **0** | 57222 | 53 | 54587 | 53 | allow | 177 | 94 | 83 | 2 | 30 | 1 | 1 | | **1** | 56258 | 3389 | 56258 | 3389 | allow | 4768 | 1600 | 3168 | 19 | 17 | 10 | 9 | | **2** | 6881 | 50321 | 43265 | 50321 | allow | 238 | 118 | 120 | 2 | 1199 | 1 | 1 | | **3** | 50553 | 3389 | 50553 | 3389 | allow | 3327 | 1438 | 1889 | 15 | 17 | 8 | 7 | | **4** | 50002 | 443 | 45848 | 443 | allow | 25358 | 6778 | 18580 | 31 | 16 | 13 | 18 |   In [3]:  df['Action'].value\_counts()  Out[3]:  allow 37640  deny 14987  drop 12851  reset-both 54  Name: Action, dtype: int64  In [4]:  df\_imputed = df.drop(df[ df['Action'].isin(['reset-both']) ].index)  df\_imputed.shape  Out[4]:  (65478, 12)  In [5]:  df\_imputed.info(verbose=**True**, null\_counts=**True**)  <class 'pandas.core.frame.DataFrame'>  Int64Index: 65478 entries, 0 to 65531  Data columns (total 12 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 Source Port 65478 non-null int64  1 Destination Port 65478 non-null int64  2 NAT Source Port 65478 non-null int64  3 NAT Destination Port 65478 non-null int64  4 Action 65478 non-null object  5 Bytes 65478 non-null int64  6 Bytes Sent 65478 non-null int64  7 Bytes Received 65478 non-null int64  8 Packets 65478 non-null int64  9 Elapsed Time (sec) 65478 non-null int64  10 pkts\_sent 65478 non-null int64  11 pkts\_received 65478 non-null int64  dtypes: int64(11), object(1)  memory usage: 6.5+ MB  In [6]:  *#Check class distribution*  %**matplotlib** inline  *# Adapted from:*  *# https://www.featureranking.com/tutorials/machine-learning-tutorials/information-gain-computation/*  **def** gini\_index(y):  probs = pd.value\_counts(y,normalize=**True**)  **return** 1 - np.sum(np.square(probs))  **def** plot\_class\_dist(y):  class\_ct = len(np.unique(y['Action']))  vc = pd.value\_counts(y['Action'])  print('Total Records', len(y['Action']))  print('Total Classes:', class\_ct)  print('Smallest Class Id:',vc.idxmin(),'Records:',vc.min())  print('Largest Class Id:',vc.idxmax(),'Records:',vc.max())  *#print('Accuracy when Guessing:', np.round( (1 / len(np.unique(y['default']))) \* 100, 2), '%')*    position\_counts = pd.DataFrame(y['Action'].value\_counts())  position\_counts['Percentage'] = position\_counts['Action']/position\_counts.sum()[0]  print(position\_counts)  plt.figure(figsize=(4,4))  plt.pie(position\_counts['Percentage'],labels = ['Allow', 'Deny', 'Drop']);    plot\_class\_dist(df\_imputed)  Total Records 65478  Total Classes: 3  Smallest Class Id: drop Records: 12851  Largest Class Id: allow Records: 37640  Action Percentage  allow 37640 0.574850  deny 14987 0.228886  drop 12851 0.196264  /hpc/applications/anaconda/3/lib/python3.6/site-packages/matplotlib/font\_manager.py:1333: UserWarning: findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans  (prop.get\_family(), self.defaultFamily[fontext]))    In [7]:  df["Source Port"].value\_counts().count()  df['Destination Port'].value\_counts().count()  df['NAT Source Port'].value\_counts().count()  df['NAT Destination Port'].value\_counts().count()  Out[7]:  22724  Out[7]:  3273  Out[7]:  29152  Out[7]:  2533  In [8]:  *#Convert ports to categorical*  df\_imputed["Source Port"] = df\_imputed["Source Port"].astype('category')  df\_imputed["Destination Port"] = df\_imputed["Destination Port"].astype('category')  df\_imputed["NAT Source Port"] = df\_imputed["NAT Source Port"].astype('category')  df\_imputed["NAT Destination Port"] = df\_imputed["NAT Destination Port"].astype('category')  df\_imputed.info(verbose=**True**, null\_counts=**True**)  <class 'pandas.core.frame.DataFrame'>  Int64Index: 65478 entries, 0 to 65531  Data columns (total 12 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 Source Port 65478 non-null category  1 Destination Port 65478 non-null category  2 NAT Source Port 65478 non-null category  3 NAT Destination Port 65478 non-null category  4 Action 65478 non-null object  5 Bytes 65478 non-null int64  6 Bytes Sent 65478 non-null int64  7 Bytes Received 65478 non-null int64  8 Packets 65478 non-null int64  9 Elapsed Time (sec) 65478 non-null int64  10 pkts\_sent 65478 non-null int64  11 pkts\_received 65478 non-null int64  dtypes: category(4), int64(7), object(1)  memory usage: 7.5+ MB  In [9]:  *#OHE columns*  ohe\_list = ['Source Port','Destination Port','NAT Source Port','NAT Destination Port']  *# get oheed columns and add to imputed and drop original columns*  pd\_ohe = pd.get\_dummies(df\_imputed[ohe\_list], prefix=ohe\_list,drop\_first=**True**,prefix\_sep="\*")  In [10]:  df\_imputed.loc[:,'Action'].value\_counts()  Out[10]:  allow 37640  deny 14987  drop 12851  Name: Action, dtype: int64  In [12]:  *#df\_target = df\_imputed.loc[:,'Action']*  *#df\_imputed.drop('Action', axis=1, inplace = True)*  df\_imputed = pd.concat([df\_imputed, pd\_ohe], axis=1)  df\_imputed.drop(ohe\_list, axis=1, inplace = **True**)  *#print\_colcounts(df\_imputed)*  print("\*\*\*\*\*\*Shape after OHE\*\*\*\*\*\*\*")  df\_imputed.shape  *#df\_target.shape*  \*\*\*\*\*\*Shape after OHE\*\*\*\*\*\*\*  Out[12]:  (65478, 57636)  In [13]:  X = df\_imputed.iloc[:,df\_imputed.columns != 'Action'].values  X.shape  y = df\_imputed['Action'].values  y.shape  *#Normalize data*  *##Scale the transformed data*  scl\_obj = StandardScaler()  scl\_obj.fit(X)  X\_scaled = scl\_obj.transform(X)  *#QuantileTransformer(output\_distribution='uniform').fit\_transform(X))*  X\_scaled.shape  *#X\_scaled*  Out[13]:  (65478, 57635)  Out[13]:  (65478,)  Out[13]:  StandardScaler()  Out[13]:  (65478, 57635)  In [14]:  *# stt = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=45)*  *# train\_index\_clf, test\_index\_clf = next(stt.split(X, y))*  *# X\_train = X[train\_index\_clf]*  *# y\_train = y[train\_index\_clf]*  *# X\_test = X[test\_index\_clf]*  *# y\_test = y[test\_index\_clf]*  In [15]:  **import** **warnings**  warnings.filterwarnings('ignore')  **from** **yellowbrick.classifier** **import** ROCAUC  **def** plot\_roc(est, X\_test, y\_test, X\_train, y\_train):  visualizer = ROCAUC(est, classes=['allow','deny', 'drop'])  visualizer.fit(X\_train, y\_train) *# Fit the training data to the visualizer*  visualizer.score(X\_test, y\_test) *# Evaluate the model on the test data*  visualizer.show()  **def** evaluate\_clf\_model\_performance(model\_name, params, clf, X, y, nCV = 10, n\_jobs = 10):  *# Lets split to train and test 80/20%*  print('Generating stratifiedtest train split')  stt = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=45)  train\_index\_clf, test\_index\_clf = next(stt.split(X, y))  X\_train = X[train\_index\_clf]  y\_train = y[train\_index\_clf].ravel()  X\_test = X[test\_index\_clf]  y\_test = y[test\_index\_clf].ravel()    *# We prepare the grid search object to be passed to GSCV*  print('Running grindsearch')  sss = StratifiedShuffleSplit(n\_splits=nCV, test\_size=0.2, random\_state=45)  grid = gridcv(clf, params, cv=sss,scoring='accuracy',n\_jobs =-1, refit=**True** )  grid.fit(X\_train, y\_train)  model\_stat = pd.DataFrame()  model\_stat['model\_name'] =[str(model\_name)]  res = grid.cv\_results\_  *#print(res)*  *# Lets store the scores for t-test validation of models*  *#cvscore = cross\_val\_score(grid.best\_estimator\_, X\_train, y\_train, scoring='f1\_weighted', cv=nCV,n\_jobs= n\_jobs)*  *#model\_stat['scores'] = [cvscore]*  *#grid.cv\_results\_.keys()*  *#res.keys()*  *#res['params']*  grid\_scr = pd.DataFrame()  grid\_scr['params'] = res['params']  grid\_scr['mean\_test\_score'] = res['mean\_test\_score']  grid\_scr = pd.DataFrame(grid\_scr)  *#print(grid\_scr)*  grid\_scr.plot.bar(color='grey',figsize=(10,6))  plt.ylabel('Accuracy')  plt.xlabel('Params')  plt.grid(color='blue', linestyle='--', linewidth=0.5)  plt.ylim(0.80,1.0)  plt.show()  print("Best parameters set found on development set:")  print()  print(grid.best\_params\_)  *#model\_stat['score'] = [grid.best\_score\_]*  print()  print("Grid scores on development set:")  print()  means = res['mean\_test\_score']  stds = res['std\_test\_score']  **for** mean, std, params **in** zip(means, stds, res['params']):  print("**%0.5f** (+/-**%0.03f**) for **%r**"  % (mean, std \* 2, params))  print()  *#plot\_roc(grid.best\_estimator\_, X\_test, y\_test, X\_train, y\_train)*  *#plt.show()*  print("Detailed classification report:")  print()  print("The model is trained on the full development set.")  print("The scores are computed on the test set.")  print()  *#build CM using test/Train*  y\_true, y\_pred = y\_test, grid.best\_estimator\_.predict(X\_test)  print("\*\*\*\*\*\*", accuracy\_score( y\_true, y\_pred), "\*\*\*\*\*\*\*\*\*")  *#y\_predprob = grid.best\_estimator\_.predict\_proba(X\_test)*    *#y\_pred*  print(classification\_report(y\_true, y\_pred, target\_names=['allow','deny', 'drop']))  s = classification\_report(y\_true, y\_pred, target\_names=['allow','deny', 'drop'])  model\_stat['CM'] = s  plot\_confusion\_matrix(grid, X\_test,y\_test,cmap=plt.cm.Blues,values\_format='d',display\_labels = ['allow','deny', 'drop'])  model\_stat['time\_refit'] = [grid.refit\_time\_]  print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  print("Time to refit: ", grid.refit\_time\_)  print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  model\_stat['model\_param'] = [str(grid.best\_params\_)]  model\_stat['weighted\_f1\_score']=round(f1\_score(y\_true, y\_pred, average='weighted'),2)  *#model\_stat['accuracy']=accuracy\_score(y\_true, y\_pred)*  plt.grid(b=**None**);  plt.show()  print()  *# for input, prediction, prob in zip(y\_true, y\_pred, y\_predprob):*  *# if prediction != input:*  *# print(input, 'has been classified as ', prediction, 'and should be ', input, ' proabability:', prob)*    **return** grid.best\_estimator\_  numCVs=3  In [16]:  *# #SVC 1*  *# from sklearn.svm import LinearSVC*  *# mdl = LinearSVC(loss = 'hinge', C = 100, class\_weight = 'balanced',*  *# random\_state=45, verbose=True)*  *# mdl.fit(X\_train, y\_train)*  *# #%time m = evaluate\_clf\_model\_performance('SVC', params,mdl,X, y, numCVs)*  In [17]:  *#sgd*  **from** **sklearn.linear\_model** **import** SGDClassifier  params = [  {'alpha': [.0001,.001], 'loss': ['log'], 'class\_weight' :['balanced']}  ]  mdl\_sgd = SGDClassifier(max\_iter=3000, random\_state=45)  %**time** m\_sgd = evaluate\_clf\_model\_performance('Sgd', params, mdl\_sgd, X, y, numCVs)  Generating stratifiedtest train split  Running grindsearch  Best parameters set found on development set:  {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'log'}  Grid scores on development set:  0.77182 (+/-0.314) for {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'log'}  0.57488 (+/-0.000) for {'alpha': 0.001, 'class\_weight': 'balanced', 'loss': 'log'}  Detailed classification report:  The model is trained on the full development set.  The scores are computed on the test set.  \*\*\*\*\*\* 0.9740378741600488 \*\*\*\*\*\*\*\*\*  precision recall f1-score support  allow 1.00 0.96 0.98 7528  deny 1.00 1.00 1.00 2998  drop 0.89 1.00 0.94 2570  accuracy 0.97 13096  macro avg 0.96 0.98 0.97 13096  weighted avg 0.98 0.97 0.97 13096  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Time to refit: 507.3700575828552  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CPU times: user 9min 50s, sys: 29.6 s, total: 10min 19s  Wall time: 31min 48s  In [16]:  *#SVC 1*  **from** **sklearn.svm** **import** LinearSVC  params = [  {'C': [90, 100], 'loss' : ['hinge'], 'class\_weight' :['balanced']},  ]  mdl = LinearSVC(random\_state=45)  %**time** m\_lsvm = evaluate\_clf\_model\_performance('SVC', params, mdl, X, y, numCVs)  Generating stratifiedtest train split  Running grindsearch    Best parameters set found on development set:  {'C': 90, 'class\_weight': 'balanced', 'loss': 'hinge'}  Grid scores on development set:  0.99863 (+/-0.001) for {'C': 90, 'class\_weight': 'balanced', 'loss': 'hinge'}  0.99863 (+/-0.001) for {'C': 100, 'class\_weight': 'balanced', 'loss': 'hinge'}  Detailed classification report:  The model is trained on the full development set.  The scores are computed on the test set.  \*\*\*\*\*\* 0.9985491753207086 \*\*\*\*\*\*\*\*\*  precision recall f1-score support  allow 1.00 1.00 1.00 7528  deny 1.00 1.00 1.00 2998  drop 1.00 1.00 1.00 2570  accuracy 1.00 13096  macro avg 1.00 1.00 1.00 13096  weighted avg 1.00 1.00 1.00 13096  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Time to refit: 30.94728684425354  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CPU times: user 1min 50s, sys: 27.4 s, total: 2min 17s  Wall time: 3min 9s  In [19]:  *#sgd*  **from** **sklearn.linear\_model** **import** SGDClassifier  params = [  {'alpha': [.0001,.001], 'loss': ['hinge'], 'class\_weight' :['balanced']}  ]  mdl\_sgd\_h = SGDClassifier(max\_iter=3000, random\_state=45)  %**time** m\_hsgd = evaluate\_clf\_model\_performance('Sgd', params, mdl\_sgd\_h, X, y, numCVs)  Generating stratifiedtest train split  Running grindsearch    Best parameters set found on development set:  {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'hinge'}  Grid scores on development set:  0.83707 (+/-0.371) for {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'hinge'}  0.83405 (+/-0.204) for {'alpha': 0.001, 'class\_weight': 'balanced', 'loss': 'hinge'}  Detailed classification report:  The model is trained on the full development set.  The scores are computed on the test set.  precision recall f1-score support  allow 1.00 0.97 0.98 7528  deny 0.00 0.00 0.00 2998  drop 0.45 1.00 0.62 2570  accuracy 0.75 13096  macro avg 0.48 0.66 0.53 13096  weighted avg 0.66 0.75 0.69 13096  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Time to refit: 372.19185972213745  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CPU times: user 7min 32s, sys: 28.5 s, total: 8min  Wall time: 30min 10s  In [18]:  *#SVC 1*  **from** **sklearn.svm** **import** LinearSVC  params = [  {'C': [90, 100], 'class\_weight' :['balanced']},  ]  mdl\_svc\_hs = LinearSVC(random\_state=45)  %**time** m\_svc\_hs = evaluate\_clf\_model\_performance('SVC', params, mdl\_svc\_hs, X, y, numCVs)  Generating stratifiedtest train split  Running grindsearch    Best parameters set found on development set:  {'C': 90, 'class\_weight': 'balanced'}  Grid scores on development set:  0.99860 (+/-0.001) for {'C': 90, 'class\_weight': 'balanced'}  0.99860 (+/-0.001) for {'C': 100, 'class\_weight': 'balanced'}  Detailed classification report:  The model is trained on the full development set.  The scores are computed on the test set.  precision recall f1-score support  allow 1.00 1.00 1.00 7528  deny 1.00 1.00 1.00 2998  drop 1.00 1.00 1.00 2570  accuracy 1.00 13096  macro avg 1.00 1.00 1.00 13096  weighted avg 1.00 1.00 1.00 13096  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Time to refit: 30.401391744613647  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CPU times: user 1min 50s, sys: 29.7 s, total: 2min 19s  Wall time: 3min 14s  In [21]:  *# from sklearn.preprocessing import LabelEncoder*  *# label\_encoder = LabelEncoder().fit(y)*  *# ye = label\_encoder.transform(y)*  *# ye*  Out[21]:  array([0, 0, 0, ..., 2, 2, 2])  In [56]:  *#sgd*  **from** **sklearn.linear\_model** **import** SGDClassifier  params = [  {'alpha': [.0001], 'loss': ['log'], 'class\_weight' :['balanced']}  ]  mdl\_sgd = SGDClassifier(max\_iter=3000, random\_state=45)  %**time** m\_hsgd = evaluate\_clf\_model\_performance('Sgd', params, mdl\_sgd, X, y, numCVs)  Generating stratifiedtest train split  Running grindsearch  Best parameters set found on development set:  {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'log'}  Grid scores on development set:  0.77182 (+/-0.314) for {'alpha': 0.0001, 'class\_weight': 'balanced', 'loss': 'log'}  Detailed classification report:  The model is trained on the full development set.  The scores are computed on the test set.  precision recall f1-score support  allow 1.00 0.96 0.98 7528  deny 1.00 1.00 1.00 2998  drop 0.89 1.00 0.94 2570  accuracy 0.97 13096  macro avg 0.96 0.98 0.97 13096  weighted avg 0.98 0.97 0.97 13096  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Time to refit: 507.3656919002533  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    CPU times: user 9min 47s, sys: 30.1 s, total: 10min 17s  Wall time: 22min 44s  In [47]:  stt = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=45)  train\_index\_clf, test\_index\_clf = next(stt.split(X, y))  X\_train = X[train\_index\_clf]  y\_train = y[train\_index\_clf]  X\_test = X[test\_index\_clf]  y\_test = y[test\_index\_clf]  In [55]:  **def** partial\_fit():  **for** i **in** range(3):  clf = SGDClassifier(loss='log', alpha=.0001)  **for** j **in** range(((math.ceil(len(X\_train)/1000)))):  print(".", end="")  *#print(j\*1000, j\*1000 + 1000 - 1)*  *#print(X[j\*1000:j\*1000 + 1000 - 1,:].shape)*  \_ = clf.partial\_fit(X\_train[j\*1000:j\*1000 + 1000 - 1,:], y\_train[j\*1000:j\*1000 + 1000 - 1], classes=['allow','deny', 'drop'])  print("\*\*\*\*\*\*", accuracy\_score( y\_test, clf.predict(X\_test)), "\*\*\*\*\*\*\*\*\*")  %**time** partial\_fit()  .....................................................\*\*\*\*\*\* 0.9601405009163103 \*\*\*\*\*\*\*\*\*  .....................................................\*\*\*\*\*\* 0.5748320097739767 \*\*\*\*\*\*\*\*\*  .....................................................\*\*\*\*\*\* 0.8903481979230299 \*\*\*\*\*\*\*\*\*  CPU times: user 3min 37s, sys: 3min 7s, total: 6min 45s  Wall time: 1min 31s  In [ ] |

# References

1. Data set info: [https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data#](https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data)
2. Firewall basics:<https://searchsecurity.techtarget.com/definition/firewall>
3. Network ports: <https://www.lifewire.com/port-0-in-tcp-and-udp-818145>
4. Classification of firewall paper: [https://ieeexplore-ieee-org.proxy.libraries.smu.edu/stamp/stamp.jsp? tp=&arnumber=8355382](https://ieeexplore-ieee-org.proxy.libraries.smu.edu/stamp/stamp.jsp?%20tp=&arnumber=8355382)