# 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.

## 1 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.

## 2 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.

## 2.1 Data

The historical data collected consists of eleven attributes and the label ("Action") shown in [Table 1 Attributes [Borrowed from 4]] 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.

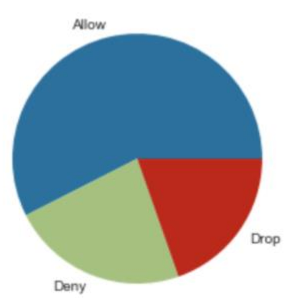
*Table 1 Attributes [Borrowed from 4]*

| Sr. No. | Feature name | Description |
|---|---|---|
| 1. | Source Port | Client Source Port |
| 2. | Destination Port | Client Destination Port |
| 3. | NAT Source Port | Network Address Translation Source Port |
| 4. | NAT Destination Port | Network Address Translation Destination Port |
| 5. | Elapsed Time (sec) | Elapsed Time for flow |
| 6. | Bytes | Total Bytes |
| 7. | Bytes Sent | Bytes Sent |
| 8. | Bytes Received | Bytes Received |
| 9. | Packets | Total Packets |
| 10. | pkts_sent | Packets Sent |
| 11. | pkts_received | Packets Received |
| 12. | Action | Class (allow, deny, drop, reset-both) |

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 1*

```
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
```



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.

## 2.2 Models

### 2.2.1 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.

### 2.2.2 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.

# 3 Results

### 3.1.1 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 2*

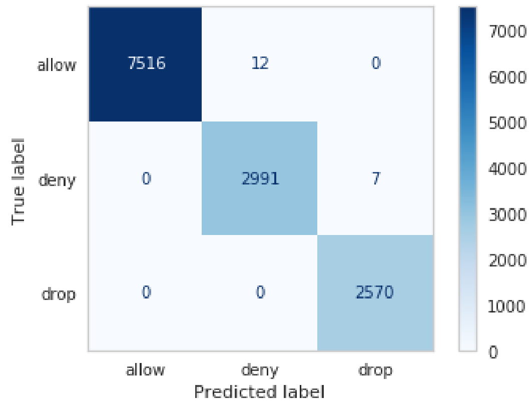



*Table 2*

| | **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 |

### 3.1.2 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 3*

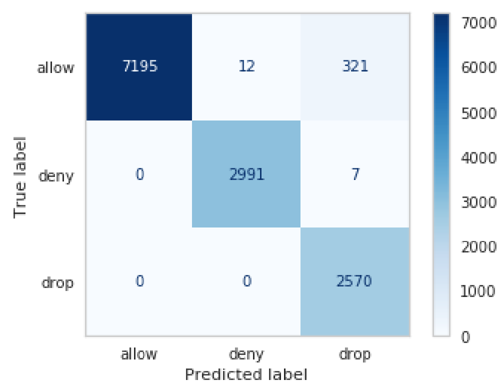



*Table 3*

| | **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 |

# 4 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 4*

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

# 5 Appendix

## 5.1 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
```

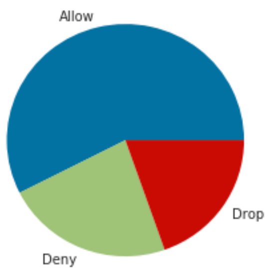
```
 4   Action                 65478 non-null  object
 5   Bytes                  65478 non-null  int64
 6   Bytes Sent             65478 non-null  int64
 7   Bytes Received         65478 non-null  int64
 8   Packets                65478 non-null  int64
 9   Elapsed Time (sec)     65478 non-null  int64
 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/inform
ation-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['defaul
t']))) * 100, 2), '%')

    position_counts = pd.DataFrame(y['Action'].value_counts())
    position_counts['Percentage'] = position_counts['Action']/position_count
s.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_man
ager.py:1333: UserWarning: findfont: Font family ['sans-serif'] not found. F
alling 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('cate
gory')
df_imputed["NAT Source Port"] = df_imputed["NAT Source Port"].astype('catego
ry')
df_imputed["NAT Destination Port"] = df_imputed["NAT Destination Port"].asty
pe('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 Destinat
ion 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=Tru
e,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 v
isualizer
    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=4
5)
    grid = gridcv(clf, params, cv=sss,scoring='accuracy',n_jobs =-1, refit=T
rue )
    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, scori
ng='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_forma
t='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 shoul
d be ', input, ' proabability:', prob)
```

```
    return grid.best_estimator_

numCVs=3
```

In [16]:

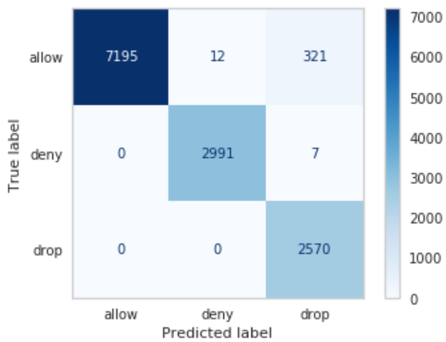
```
# #SVC 1
# from sklearn.svm import LinearSVC
# mdl = LinearSVC(loss = 'hinge', C = 100, class_weight = 'balanced',
#                  random_state=45, verbose=True)
# mdl.fit(X_train, y_train)
# #%time m = evaluate_clf_model_performance('SVC', params,mdl,X, y, numCVs)
```

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, n
umCVs)
```

```
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
*********************************************
```

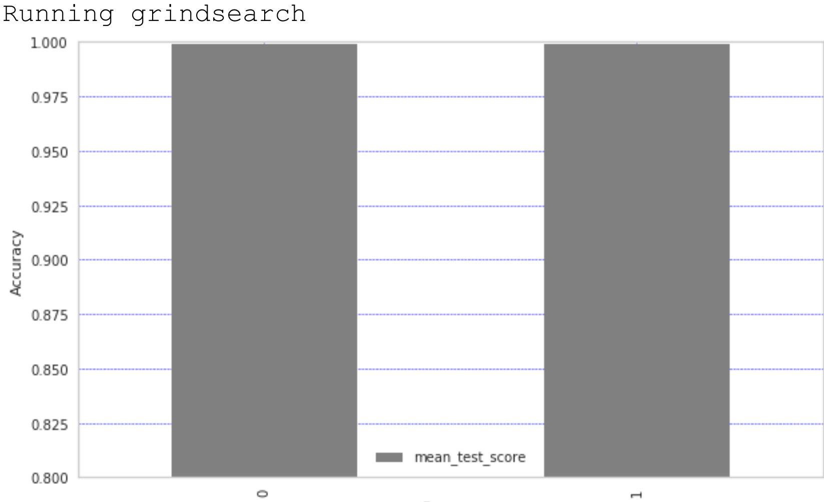



```
CPU times: user 9min 50s, sys: 29.6 s, total: 10min 19s
Wall time: 31min 48s
```

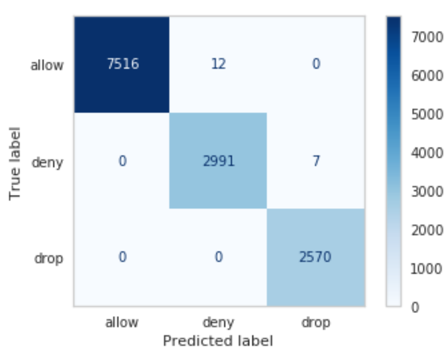
In [16]:

```python
#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, numC
Vs)
```

```
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
*********************************************
```

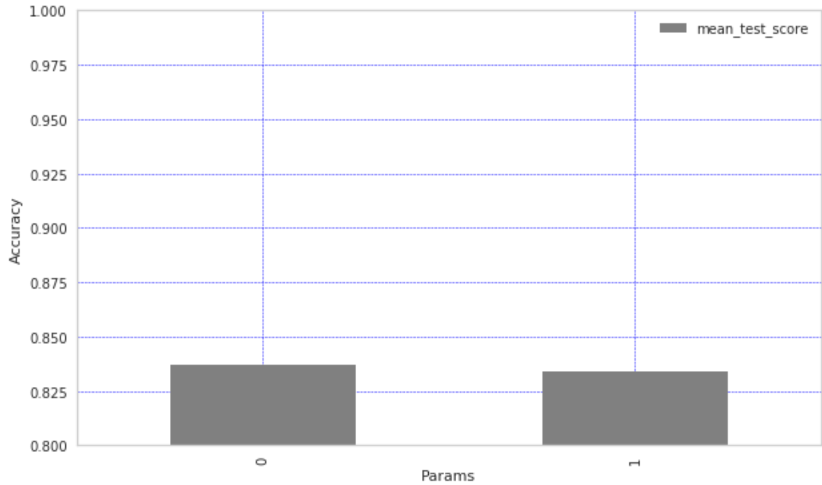



```
CPU times: user 1min 50s, sys: 27.4 s, total: 2min 17s
Wall time: 3min 9s
```

In [19]:

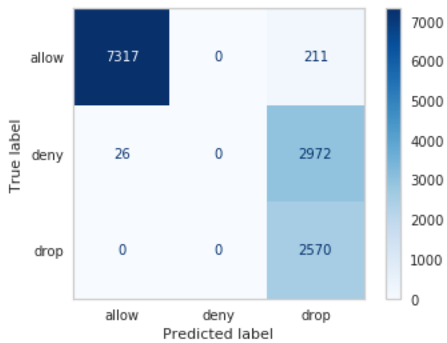
```
#sgd
from sklearn.linear_model import SGDClassifier
```

```
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
*********************************************
```

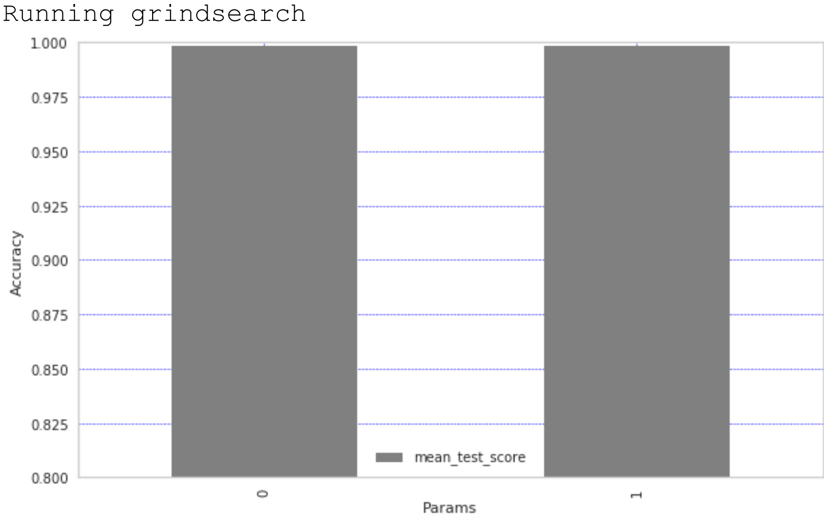



```
CPU times: user 7min 32s, sys: 28.5 s, total: 8min
Wall time: 30min 10s
```

In [18]:

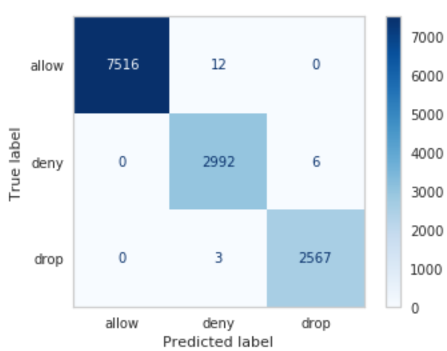
```
#SVC 1
from sklearn.svm import LinearSVC
params = [
      {'C': [90, 100], 'class_weight' :['balanced']},
     ]
mdl_svc_hs = LinearSVC(random_state=45)
%time m_svc_hs = evaluate_clf_model_performance('SVC', params, mdl_svc_hs, X
, y, numCVs)
```

```
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]:

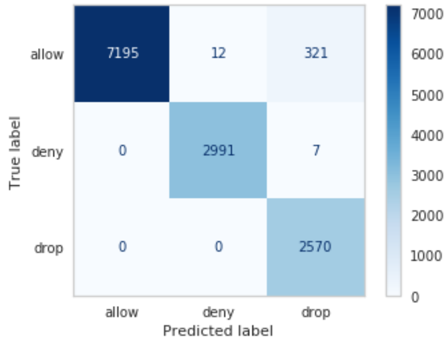
```
# from sklearn.preprocessing import LabelEncoder
# label_encoder = LabelEncoder().fit(y)
# ye = label_encoder.transform(y)
# ye
```

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]:

```python
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]:

```python
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()
```

```
.....................................................****** 0.96014050091631
03 *********
.....................................................****** 0.57483200977397
67 *********
.....................................................****** 0.89034819792302
99 *********
CPU times: user 3min 37s, sys: 3min 7s, total: 6min 45s
Wall time: 1min 31s
```

In [ ]:

# 6 References


1. Data set info: 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