DS7333 Quantifying the World: Case Study 5

Rick Fontenot, Feby Thomas Cheruvathoor

1. **Introduction**

­

The goal for this case study is to use information associated with internet connection requests to determine the action taken by firewalls to accept or deny connections. Support Vector Machines (SVM) and Stochastic Gradient Descent (SGD) modeling tools were requested to make predictions on the actions.

1. **Methods**

**Data description**

This Internet Firewall Data Set originates from the UCI Machine Learning repository (<https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data#>) and includes 65,532 observations of connection requests.

There are 11 features that can be used to make predictions:

Source Port: Categorical feature with 22,692 unique Port ID’s

Destination Port: Categorical feature with 3264 unique Port ID’s

NAT Source Port: Categorical feature with 29,143 unique Port ID’s

NAT Destination Port: Categorical feature with 2533 unique Port ID’s

Bytes: Continuous feature

Bytes Sent: Continuous feature

Bytes Received: Continuous feature

Packets: Continuous feature

Elapsed Time (sec): Continuous feature

Pkts sent: Continuous feature

Pkts received: Continuous feature

The target variable Action has 4 distinct categories:

Allow: 37640 observations

Deny: 14987 observations

Drop: 12851 observations

reset-both: 54 observations

**Processing data & Feature Creation**

Class Imbalance Issue:

Due to the extremely low count of observations for the action “reset-both” these observations were dropped, as re-sampling would not be feasible to overcome the extreme class imbalance.

Duplicates:

After removing the “reset-both” observations, the remaining dataset includes 8,362 exact duplicate rows. Only the first observation of the duplicates were kept, all others removed. This reduced the dataset to 57,116 observations.

Missing Values:

There were no missing values in the dataset that required imputing or removal

Categorical Encoding:

Although the four features associated with port id’s are numeric, the values do not represent any ordinal relationship. These features were first converted to strings and then one-hot encoded for proper use in modeling. The first dummy column for each port was dropped to prevent multicollinearity issues.

After data processing and feature creation the dataset contains 57,116 observations with 57,636 features

**Training Split & Cross Validation Strategy**

While some of the modeling tools to be used can handle very large datasets, support vector machines will be challenged by the size of this dataset vs. the computing memory available. Prior to determining the training and test data splits, learning curves were assessed on LinearSVC and SGD tools to determine the effects of the training set size. The figures below show a divergence between training and test accuracy when there are less than 10,000 training instances. Using approximately 35,000 instances for training is in a stable part of the curve for accuracy performance on the test sets, and there does not appear to be performance improvements for increasing training instances beyond that point. Also note that while there is divergence at 5,000 training instances, the test accuracy is still relatively high indicating that this training size could be useful for saving time to assess multiple combinations of hyperparameters for tuning.

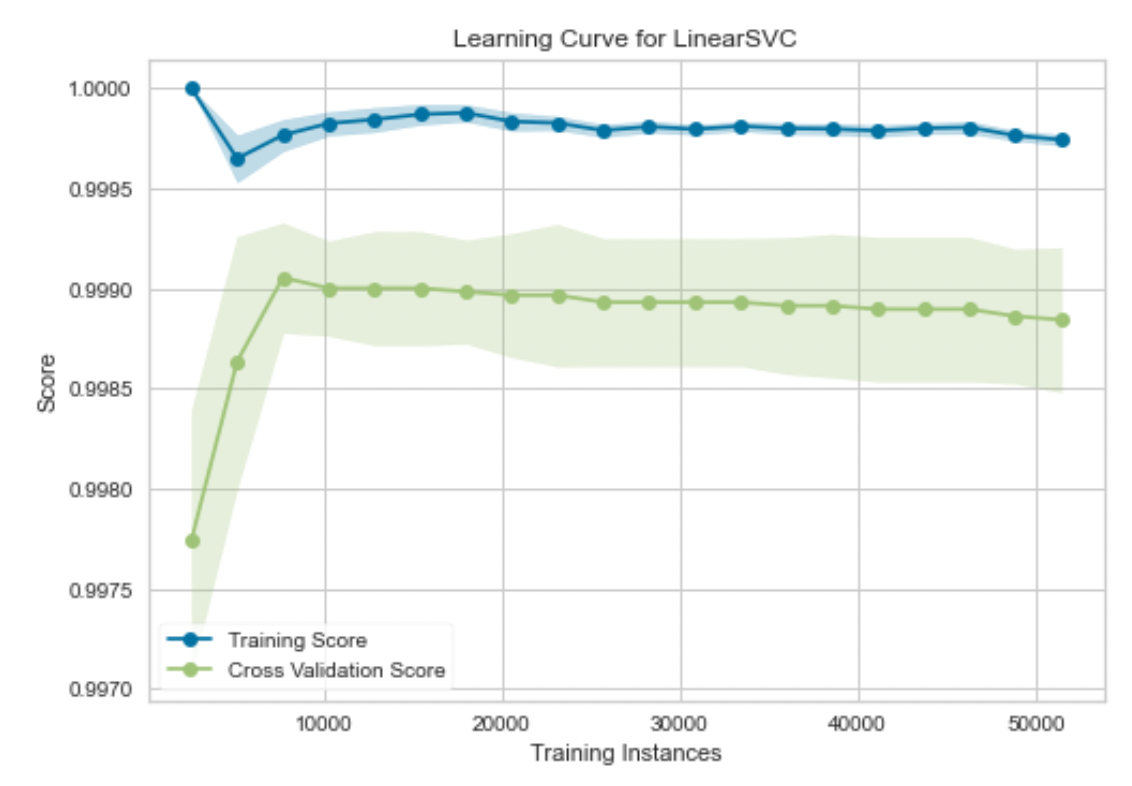


Figure 1: Training Instances vs. Accuracy using default LinearSVC model

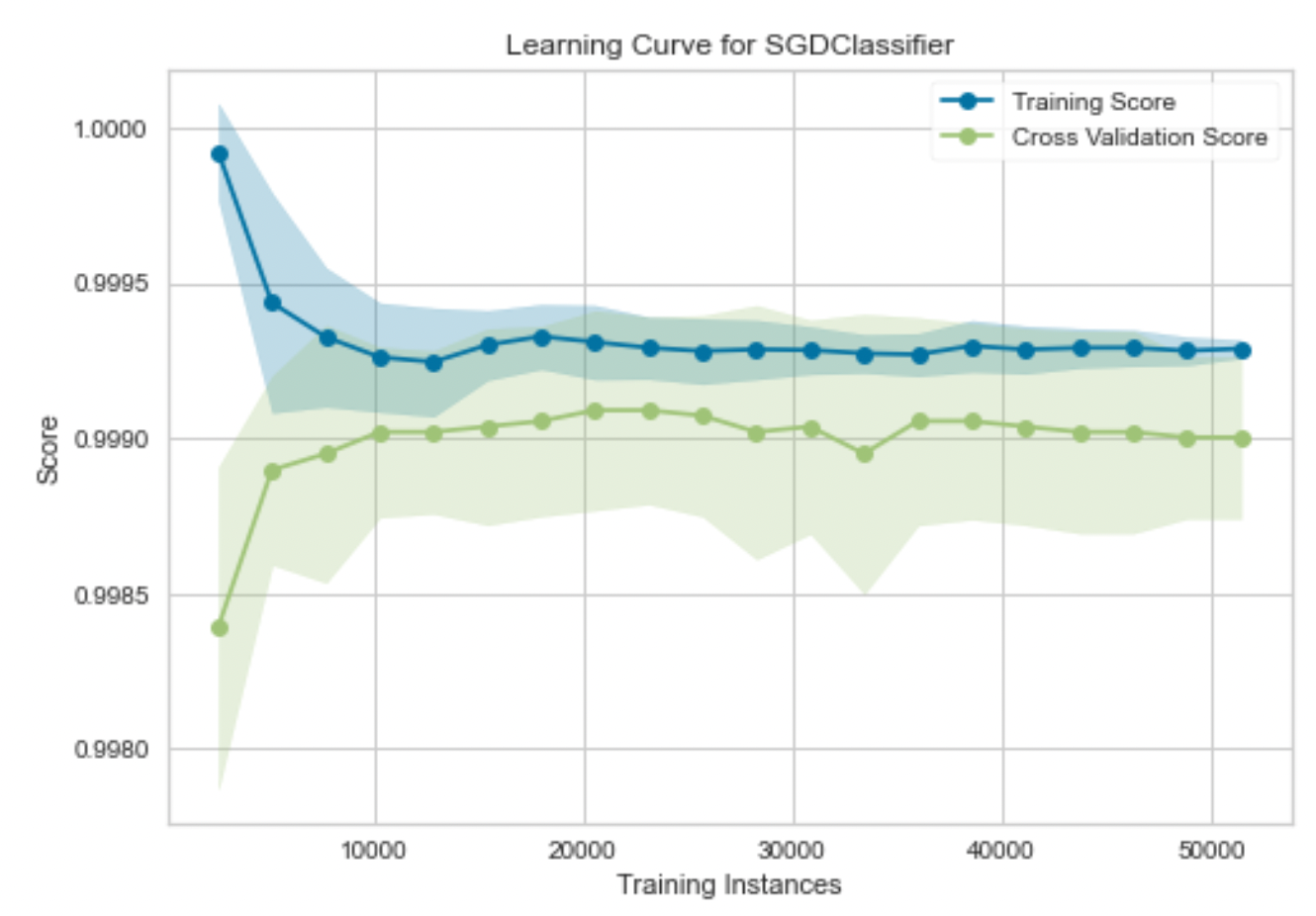


Figure 2: Training Instances vs. Accuracy using default SGDClassifier model

Based on the insights gained from the learning curves, the data was split into 60% Training, 20% Validation to be used for model tuning, and 20% Test to provide a final unbiased assessment of performance.

**Class Imbalance**

After removing the “reset-both” action category (54 observations) there appear to be sufficient samples of the remaining 3 action categories, so no re-sampling was performed to obtain even balancing.

**Random Forest modeling**

A random forest model using default parameters for the sklearn RandomForestClassifier was created as an initial benchmark for further model comparisons. Original port features were used rather than the one-hot encoded versions. No grid searches or hyperparameter tuning was performed. The intent of this model was to check accuracy on a simple, fast training model that the SVM and SGD models can be compared against.

**Linear Support Vector Classifier modeling**

The first support vector model attempted used sklearn LinearSVC due to it’s speedier training time compared to other SVM tools with multiple kernel options.

Grid searches were utilized to tune hyperparameters in multiple passes.

The first grid search focused on selecting the most appropriate loss function and regularization parameter (C):

Range:

parameters = {'C':[1,10,100,1000], 'loss':['hinge', 'squared\_hinge']}

After selecting the “hinge” loss function with regularization of C=10, a second search was performed to optimize the number of iterations to be performed along with the tolerance for stopping criteria

Range:

parameters = {'C':[1], 'loss':['hinge'], 'tol':[1e-6, 1e-4, 1e-2], 'max\_iter':[1000, 2000, 5000]}

Reducing the tolerance from a default of 1e-4 to 1e-6 improved the performance while still needing less than the maximum iteration=1000.

The final parameters for the tuned model are:

C=1, loss='hinge', max\_iter=1000, tol=1e-6

**Support Vector (non-linear) modeling**

Additional support vector models utilizing non-linear kernels were assessed using the sklearn SVC tool. This modeling technique is extremely costly with respect to training time due to memory processing limitations. Based on the learning curves referenced above, initial analysis of these models were performed using only portions of the full training set.

Grid searches were utilized to tune hyperparameters in multiple passes, using 5,000 training instances.

The first grid search focused on selecting the most appropriate kernel and regularization parameter (C):

Range:

parameters = {'C': [0.1,1, 10],'kernel': ['rbf', 'poly', 'sigmoid']}

After selecting the “rbf” kernel, a second search was performed to optimize the regularization parameter(C), the kernel coefficient (Gamma), the number of iterations to be performed with the tolerance for stopping criteria, and evaluation of using one-vs-one (ovo) decision criteria vs. the default one-vs-rest (ovr).

Range:

parameters = {'C': [1, 10],'kernel': ['rbf'],'gamma':['scale', 'auto'],'tol':[1e-6, 1e-3], 'max\_iter':[-1],'decision\_function\_shape':['ovo', 'ovr']}

The final parameters used in the tuned model are:

C=10, decision\_function\_shape='ovo', gamma='scale', kernel='rbf', tol= 1e-06

The final tuned model was then re-trained increasing the training instances by 5,000 per step to assess the feasibility of training time as well as changes in test accuracy with more training.

**Stochastic Gradient Descent modeling**

The first SGD model attempted used sklearn SGDClassifier due to its easy to use interface within Python.

A grid search was utilized to asses the number of iterations and stopping criteria for the “hinge” loss function.

Range:

parameters = {

"max\_iter": [1000, 5000],

"tol": [1e-3, 1e-1],

"early\_stopping": [True, False],

"n\_iter\_no\_change": [5,10]}

The search resulted in final model with a full 5,000 iterations and no stopping criteria. The final tuned model was then re-trained increasing the training instances by 5,000 per step to assess the feasibility of training time as well as changes in test accuracy with more training, and was able to achieve training with all instances.

**Out-of-Core Stochastic Gradient Descent modeling**

Additional SGD models were created using the Vowpal Wabbit (VW) package which utilizes out-of-core computing to reduce the training time and make further assessment of model parameters.

A custom function was used in python to transform the data into VW format for modeling with the vowpalwabbit and Workspace wrappers.

While this package does not have the simple intuitive grid search tools comparable to sklearn packages, hyperparameters were manually iterated through for basic model tuning.

All models used a one-against-all (ooa) decision criteria. Searches increasing the number of passes and varying the L2 regularization parameter did not significantly affect accuracy.

Scaling the learning rate did have an effect on accuracy, and the best result came from using the “adax” approach which adaptive learning rates with x2 instead of g2x2.

After tuning hyperparameters, the readable\_model and save\_resume tools were added to capture outputs so that the final weights could be assessed for feature importance.

The final tuned model included the following parameters and calls:

vowpalwabbit.Workspace("--oaa 3 --adax -l 50 --quiet --cache --passes 2000 --loss\_function hinge --l2 0.0001 --save\_resume --readable\_model vw\_output.txt")

1. **Results**

The initial Random Forest model set a high benchmark with accuracy of 98.86% and training time significantly less than a minute including all training instances.

While the linear support vector classifier trained in a reasonable time, it still took longer than the random forest and did not obtain as high accuracy. Using a support vector with the rbf kernel was able to match the accuracy of the random forest as training instances were increased to about 40% but the training time was astronomical at 230 minutes. Using the full training set with SVC-rbf was not feasible.

The VW-SGD model trained the fastest and while not quite as high accuracy as the random forest model, it’s is extremely close at 98.83%. Due to the ability to assess feature weights and assess feature importance, this was the final model chosen for further analysis.



Figure 3: Performance Summary for each model considered

**VW-SGD model results**

The confusion matrix and classification reports show model predictions perform well for all three classes of the actions taken on connections requests:

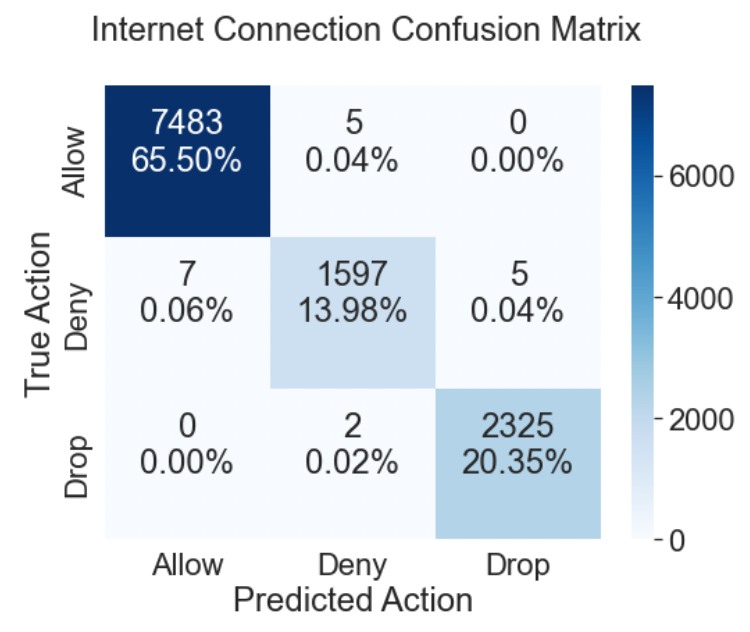


Figure 4: Confusion Matrix for the VW SGD predictions



Figure 5: Classification Report for the VW SGD predictions

Precision and recall are perfect on the Allow and drop actions. There are a few misclassifications on the deny action, but no further model improvements were obtained to reach perfect predictions.

**Feature Importance**

Feature weights from the SGD model were assessed to understand feature importance. The top 10 weights are all associated with ports that have very few observations. If there’s only one observation on a port intuitively it’s prediction would be based heavily on that single observation. Further study of weights was performed by ranking weights for ports that have at least 10 observations.



Figure 6: Top 10 feature weights overall



Figure 7: Top 10 feature weights for ports with at least 10 observations

Examining these most significant features further shows that four Destination Ports (37965, 51050, 31573, & 55442) are denied connection 100% of the time independent of the source port requested or the size of information sent in request.

The other source and destination ports in the top 10 weighted features had the connection allowed independent of the other ports associated with the request.



Figure 8: Percentage of actions taken on top 10 features

1. **Conclusions**

With just the information on the source and destination ports associated with an internet request, a very high accuracy model to predict the actions of a firewall can be made. In addition to aiding a firewall management team, these models could potentially be used to debug denied connections by predicting which alternate source or destination ports would have success give constraints on one side of a connection.

The survey of modeling techniques also provides insight into the value of rapidly training model types such as the out-of-core stochastic gradient descent method. While other modeling methods could possibly achieve higher accuracy, the 99.8% accuracy of a model that trains in just a few seconds would be easy to constantly update with new data.

**Appendix**

1. **Code**

A rendered notebook containing code for the base analysis can be accessed at:

<https://nbviewer.org/github/rickfontenot/QTW/blob/main/Case%20Study%205/case5_rick.ipynb>