JIGSAW ACADEMY
THE ONLINE SCHOOL OF ANALYTICS

Case study for Spark: ML-Lib

Network Intrusion Detection

Time: 5 hours.

Business case scenario: Today's connected networks pose a great threat to security and security experts have to be on the edge all the time to detect patterns in network intrusion. Sifting through tons of security data to look for an attack is like looking for a pin in a haystack. The network data collected over a period is analyzed and classified into different types of attacks. This data can be used to detect future attacks by matching the current network data with the classified data. The amount of data makes it difficult to match each new network connection data with the old data. Big data has come to the rescue of security professionals buy offering machine learning solutions at scale to detect patterns in data and detect intrusions.

Data set:

KDD cup data was created back in 1999 as a problem posed to solve network intrusion detection. The data provided for this can be found at `http://kdd.ics.uci.edu/databases/kddcup99`. A 10% data is also available at the same location. This data contains 41 columns of features and a label indicating a specific network intrusion threat posed by the data. One of the labels is 'normal.' indicating that the data has no threat.

Apart from this historic data, a normal data set of below ten records is to be used. For this data, a check can be done to see if they amount to a network intrusion.

305,tcp,telnet,SF,1735,2766,0,0,0,3,0,1,2,1,0,0,1,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,2,4,1.00,0.00,0.50,0.50,0.00,0.00,0.00,0.00

0,tcp,telnet,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,5,0.83,1.00,0.00,0.00,0.83,0.33,0.00,5,6,1.00,0.00,0.20,0.33,1.00,0.83,0.00,0.00

0,tcp,telnet,RSTO,126,179,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,1.00,1.00,1.00,0.00,0.00,21,21,1.00,0.00,0.05,0.00,0.05,0.05,0.95,0.95

1,tcp,private,RSTR,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,48,2,0.00,0.00,1.00,1.00,0.04,0.54,0.00,218,2,0.01,0.12,0.22,0.00,0.01,0.00,0.44,1.00

0,tcp,smtp,SF,0,91,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,6,51,0.17,1.00,0.17,0.04,0.00,0.00,0.83,0.00

0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,1.00,1.00,1.00,0.00,0.00,16,2,0.06,1.00,0.06,1.00,0.00,0.00,0.94,1.00

3,tcp,telnet,SF,54,114,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,1,0.00,0.00,0.29,0.00,0.14,0.71,0.00,160,29,0.18,0.08,0.01,0.00,0.01,0.03,0.01,0.00

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,1,0.71,1.00,0.29,0.00,0.14,0.57,0.00,255,1,0.00,0.02,0.03,0.00,0.02,1.00,0.01,0.00

0,tcp,telnet,RSTO,125,179,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,1.00,1.00,1.00,0.00,0.00,7,7,1.00,0.00,0.14,0.00,0.14,0.14,0.86,0.86

718,tcp,telnet,SF,1412,25260,0,0,0,15,0,1,38,1,0,54,4,1,2,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,1,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00

Problem: With the existing threat data from Kdd site, build a model that can predict and classify a network connection as a particular threat. Using this model, predict the classification for the above ten records.

Output:

A list of classification labels like below for the above data:

buffer_overflow.

neptune.

guess_passwd.

portsweep.

ipsweep.

ipsweep.

satan.

portsweep.

guess_passwd.

multihop.

Above is sample output only. Your output may differ based on the model.

Solution: Apache Spark can be used to provide a solution for this. Because of the distributed nature f Spark, the solution can scale to handle millions of records in security data. Also the Spark uses distributed machine learning algorithms that can be used here. Down load the data and use the decision tree algorithms of Spark to find out if a particular connection belongs to one of the predefined security issues.

Introduction to Decision Trees:

Decision trees are a good Machine learning technique for classification of data based on existing data. In this technique, a model is built as a series of nodes in a binary tree. Each node represents a true or false decision based on the data. Based on the training data, the model is built and then applied to actual data to arrive at predicted classifications.

For example a bank has customer data like below for its personal loan customers. It has two features age and annual income of the customer. The customers are classified into four categories as ‘On time’, ‘one late payment in the last year’, ‘two late payments in the last year’ and ‘defaulted on loan’. Based on the existing data, a decision tree model is built.

Customer	Age	Annual Income	Loan payments
Customer 1	25	40000	On time
Customer 2	30	50000	1 Late payment
Customer 3	35	40000	2 Late payments
Customer 4	22	30000	Defaulted
Customer 5	30	60000	On time
Customer 6	26	30000	1 Late payment
customer 7	50	40000	On time
customer 8	45	80000	2 Late payments

When a customer walks in for a loan, above model is applied to predict the classification for the customer. For example a customer with 28 years age and annual income of 45000 may be classified as likely to have 1 Late payment. This will be used by the loan approver to decide on granting the loan as well as deciding the risk level of the customer. Note that exact fitting of the decisions to data is not expected and in fact overfitting the rules may lead to incorrect decisions on new set of data.

This is a supervised machine learning technique as the existing data is already classified. When building a model for decision trees, the existing data is divided into two parts: One part called training data to build the model and another part called test data to verify the accuracy of the model. Once you have the required accuracy, then the model is ready to be used on actual data.

Tasks in the project.

Task 1. Get KDD cup data (10% data for practice)

This can be obtained from http://kdd.ics.uci.edu/databases/kddcup99. The 10% data is available in the the gzipped file kddcup.data_10_percent.gz. Download and unzip this file to get the data. The list of features in the file are available in the file kddcupnames. This is an uncompressed file and can be checked as is.

The data is comma separated and has the following fields:

1. duration
2. protocol_type
3. service
4. flag
5. src_bytes
6. dst_bytes
7. land
8. wrong_fragment
9. urgent
10. hot
11. num_failed_logins
12. logged_in
13. num_compromised
14. root_shell
15. su_attempted
16. num_root
17. num_file_creations
18. num_shells

```
19.     num_access_files
20.     num_outbound_cmds
21.     is_host_login
22.     is_guest_login
23.     count
24.     srv_count
25.     serror_rate
26.     srv_serror_rate
27.     rerror_rate
28.     srv_rerror_rate
29.     same_srv_rate
30.     diff_srv_rate
31.     srv_diff_host_rate
32.     dst_host_count
33.     dst_host_srv_count
34.     dst_host_same_srv_rate
35.     dst_host_diff_srv_rate
36.     dst_host_same_src_port_rate
37.     dst_host_srv_diff_host_rate
38.     dst_host_serror_rate
39.     dst_host_srv_serror_rate
40.     dst_host_rerror_rate
41.     dst_host_srv_rerror_rate
42.     label
```

The fields 2, 3 and 4 above have non-numeric data. We need to remove these fields for building our model as Spark uses only numeric values for features. The last field label is the classification label.

Task 2. Load and parse the data in Spark

Start your spark shell and load the data in spark. You can load the data using the textFile command. You can specify it as a regular file system file by using the prefix “file://’ and then the path of the file.

Task 3. Check the number of fields in data

You can do this by getting only the first line of data and counting the number of fields.

Task 4. Get the number of distinct labels and create a map.

The last field in data is the label that specifies the required classification. You can extract the last field in data and get distinct values. You can then create a map to get a unique number for each label. This is required as Spark uses decimal values only for Vectors and labeled points in Machine learning algorithms. The zipWithIndex method on RDDs is very handy to achieve this.

Task 5. Parse the data to create an RDD of labeled points

Parse the comma separated data to :

1. Remove the fields that have string values : Fields 1, 2 and 3.
2. Remove and extract the last field as label
3. Use the previously created map to get a unique number for each label

4. Create a labeled point with the above numeric label and Vector of features

Task 6. Divide the data into build and test

We need to create two sets of data. One set for training the model and another set for testing the model. We can divide the data into two parts : 90% for training and 10% for testing. Use the readily available randomSplit method of Spark to do this.

Task 7. Build a decision tree model

Now we are set to build our first decision tree model. Use the DecisionTree.trainClassifier method of Spark machine learning library to build the model. The model takes some hyper parameters as below:

training data : An RDD of labeled points

Number of distinct labels : Number of classifications in data

A map for category features. This is to specify how many distinct values are there for each category feature. A map from Int to Int is expected. We will not use this and specify an empty map.

An impurity evaluation method. Currently two methods are available, “gini” and “entropy”. You can use either of them to build the model. Use “gini” for the first model.

Maximum depth of tree to build : This is the depth of the decision tree built. Specifying too high a depth will need more memory and time and also may cause the tree to over fit the data. Use a depth of 4 for this example.

Maximum number of bins : This is how many memory bins to be used by the algorithm which is the same as number of decision rules to try. A higher value will need more memory and processing power and may result in a better model. Use 100 in this example.

Task 8. Check the accuracy of the model

Once the model is built, we need to use the test data to evaluate the accuracy of the model. Spark provides the multiClassMetrics API to evaluate the classification based on the predicted labels and actual labels. There are different measures defined like below:

Confusion matrix: An NxN matrix of number of cases of predicted value vs actual value. Each row in confusion matrix represents an actual value and each column represents the predicted value. The diagonal of the confusion matrix represents the ‘True Positives’ where the actual value matches the predicted value.

Precision: This is a percentage number where the true positives occur. This will be the sum of the values in the diagonal of the confusion matrix divided by the total number of values. A higher percentage refers to a better model.

First build a matrix of predicted labels for the test data to the actual lablels. You can use the predict function of the model to predict the label for the features. Then pass this matrix to multiClassMetrics API to get the metrics object. From the metrics object, print out the confusion matrix and the precision.

Task 9. Refine the model with different impurity.

Do the steps 7 and 8 with impurity set to “entropy”. Check which one has better precision.

Task 10. Create a reverse map from decimal labels to string labels.

The model has decimal labels where as we originally have human readable labels. So from the distinct label map we created earlier, create a reverse map that gives the string labels given the numbers. Use this reverse map to check the predicted labels for some of the test data. Scala provides a swap method on the map to reverse the map.

Task 11. Load the normal data shown above, parse and predict the classification labels using the model. The output may differ based on your data and also since the models have certain randomness built into them.