**SPARK – PYTHON – DECISION TREE**

**NETWORK INTRUSIONS DETECTION**

1. **Use Case Name:** Spark and Python: MLlib Decision Tree
2. **Use Case URL**: <https://www.codementor.io/jadianes/spark-python-mllib-decision-trees-du107qr0j>
3. **Introduction** - what is this use case

In this use case, I will use Spark Machine learning library MLlib to build a Decision tree classifier model for network attack detection. The use case goal is to help the learner understand how to use Spark ML library, especially the decision tree classifier model. Using Spark MLlib and Spark RDD improves the performance of model in dealing with large datasets. Furthermore, to provide a human-readable result I applied D3js to visualize the decision tree output as a diagram.

1. **Dataset used**: Dataset: KDD Cup 1999

**Source address:** <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

**Data description:** <http://kdd.ics.uci.edu/databases/kddcup99/task.html>

**Summary:** This dataset is used for KDD Cup and data mining tool competition. KDD Cup is the annual Data Mining and Knowledge Discovery competition organized by ACM Special Interest Group on Knowledge Discovery and Data Mining, the leading professional organization of data miners. In this use case, I used the KDD Cup data in 1999. The data was prepared and managed by MIT Lincoln Labs. The raw data is a wide variety of intrusions connection data which are simulated as in a military network environment. Basically, each observation in the data set was the information of TCP connection to a local-area network (LAN) simulating as U.S. Air Force LAN. Each observation has 42 attributes and was labeled as ‘normal’ or ‘attack’. The detail of data description is placed in the appendix.

1. **Technical Details**: Which aspect of Spark is applied

In this Use case, I applied Spark RDD – Resilient Distributed Dataset and Spark MLlib -Machine learning Library. The purpose is using MLlib Decision tree classifier to build a model to support user recognizes an attack network interaction.

1. **Results**:

After implementing the decision tree model with all 41 predictors of the dataset I received the full tree model with the accuracy was 91.5%. The result shows that Feature 22 – count was used as the first node split in the tree. At a second level was dst\_byte(5), dst\_host\_rerror\_rate(38), follow by service(2), flag(3) and so on. Based on this result, what I can derive is these features are more important in identifying intrusion connection because they were on the top of the tree. Then I can simplify the tree by minimizing the number of features used to train the decision tree. Finally, I ended up with a minimal tree which only uses 6 features and has 3 levels of depth. The accuracy of the minimal tree is acceptable 92% compare to full-tree 91.5% and the time spent on building the minimal tree was only 188.3 seconds. The minimal tree model has 8 rules to classify the network interaction. Due to the rules of prediction, I can characterize these following network interactions as an attack:

* If the network interactions have count <= 33.5, dst\_host\_srv\_serror\_rate <= 0.794, and dst\_host\_srv\_diff\_host\_rate > 0.49 (1)
* If the network interactions have count<=33.5, dst\_host\_srv\_serror\_rate > 0.794, and flags was not 'SF', 'S2', 'S1',’RSTO’, or ‘RSTR’
* If the network interactions have count > 33.5, dst\_bytes <= 2.0, and service was not urp\_i, or tftp\_u
* If the network interactions have count > 33.5, dst\_bytes > 2.0, and flags were not SF, S1, RSTR, or SH

The first rule can be explained as: the network interactions have less than 33.5 connection to the same server in the last 2 seconds, having the % of connections to current host and specified service that have an S0 error smaller than or equal 79.4% and the % of connections to the same service coming from diff hosts greater than 49% are predicted as ***Attack.*** A similar approach can be used for other rules.

The result from classification tree is pretty simple for interpretation, this is one of its benefits. Furthermore, the tree diagram provides more convenient for even data scientist or stakeholder to get an insight into the data. Finally, developers can base on this classify model to build an application for network intrusion detection.

To see the tree diagram please read the appendix for instruction.

1. **Insight:**

From this use case, I have learned how to used the MLlib Decision Tree to detect network attacks. The design of the Decision Tree Classifier function in MLlib is pretty simple. It does not need to include as many parameters as other libraries, and the documentation also clear and comprehensible. I also grasped the data preparation process for Decision Tree Model with Spark MLlib.

Apache Spark is an ideal tool for data science work, especially for Machine Learning. By apply RDD as data structure for the dataset and MLlib I can easily to handle large dataset and training machine learning model faster. The full dataset contains 4.8 million records and the model only took 286 seconds to read and training full tree model.

In building the decision tree model, deeper trees are more expressive with higher accuracy, but they are more costly to train and more likely to overfit. Instead of attempting to obtain the highest accuracy model we should consider other factors like performance, time-consuming, and robustness.

1. **Debugging Details**:

There are some differences in the result from my model and tutorial model. The reason could be because the spark was used in the tutorial is Spark 1.3.1 and in my use case is Spark 2.3.1. However, the training task performs better, and the accuracy is higher. There are some modifications to make the code fit to my case like changing the features used in the minimal tree, and modifying data preparation code for the minimal tree. To visualize the tree diagram, the solution that I can find is parsing the model result in JSON structure and using D3js to create the graph from the JSON file. The problematics was HTML and spark debugging. Spark Debugging can be frustrating since its’ features in-memory and lazy-execution. I had applied Python debugger to detect the problems. For HTML and JavaScript, lacking experience in these subject also caused many issues. For example, XMLHttpRequest was one error I faced when creating the tree diagram in my local machine, I solved the problem by uploading the code file to the cloud for testing, another solution can be used is create a shortcut of the web browser without checking security. But I had wasted time on this problem a lot.

**Achievements:** I have learned how to use MLlib decision tree classifier, export the result into JSON structure and visualize it as a diagram.

**Key features of coding practices:** RDD and Labelpoint object, Decision Tree Algorithm, recursive function, JSON structure, D3js, and JavaScript.

1. **Conceptual Framework**:

* **Big data platform**: Spark Apache, Spark RDD
* **Machine Learning**: Spark MLlib – Decision Tree Classifier
* **Visualization**: D3js

1. **5 V’s**:

|  |  |
| --- | --- |
| Velocity | The KDD cup 1999 dataset was generated by MIT Lincoln Lab in nine weeks with 4898431 connections. In other words, there were 54 connections generated per minute. By 2020 Cisco expects that there will be more than 250 things will connect to the internet by second in the worldwide. I assume that only 5% of this number are the connections to the use case’s LAN, that means the dataset will increase 12 records per second. |
| Volume | Each connection records in the dataset consist of about 100 bytes. This dataset is 743MB uncompressed. Although it is not much as considered big data, this use case is just an example of Spark MLlib application. In this example, we can analyze and training model with 743MB data in around 3 minute, which can prove the capability of Spark in Big data analytics. |
| Value | The more development in information technology the more sophisticated cyber attack. Therefore, the capable of distinguishing normal and attack network plays a crucial role in an organization. The output data from the model can help organization recognized 92% of network intrusion. |
| Variety | The dataset was generated as structured data with both continuous and discrete features. However, the raw data was binary TCP dump data from network traffic. |
| Veracity | The dataset was created and managed by MIT Lincoln Labs for Knowledge Discovery and Data Mining Tools Competition. The data is high quality, accuracy and trustworthy. |

1. **References (at least 5 references including the main project source, lecture, presentation, etc.)**

<https://www.codementor.io/jadianes/spark-python-mllib-decision-trees-du107qr0j>

<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

<https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html>

<https://github.com/jadianes/spark-py-notebooks/blob/master/nb9-mllib-trees/nb9-mllib-trees.ipynb>

<https://github.com/sanxofon/json_html_viewer/blob/master/json2js/json2jsondata.py>

<https://github.com/tristaneljed/Decision-Tree-Visualization-Spark>

<https://bl.ocks.org/d3noob/43a860bc0024792f8803bba8ca0d5ecd>

<https://d3js.org/>

<https://blogs.cisco.com/news/cisco-connections-counter>

1. **A list with short descriptions of all fields in dataset used**

|  |  |  |  |
| --- | --- | --- | --- |
| ***column index*** | ***feature name*** | ***description*** | ***type*** |
| 0 | duration | length (number of seconds) of the connection | continuous |
| 1 | protocol\_type | type of the protocol, e.g. tcp, udp, etc. | discrete |
| 2 | service | network service on the destination, e.g., http, telnet, etc. | discrete |
| 3 | flag | normal or error status of the connection | discrete |
| 4 | src\_bytes | number of data bytes from source to destination | continuous |
| 5 | dst\_bytes | number of data bytes from destination to source | continuous |
| 6 | land | 1 if connection is from/to the same host/port; 0 otherwise | discrete |
| 7 | wrong\_fragment | number of ``wrong'' fragments | continuous |
| 8 | urgent | number of urgent packets | continuous |
| 9 | hot | number of ``hot'' indicators | continuous |
| 10 | num\_failed\_logins | number of failed login attempts | continuous |
| 11 | logged\_in | 1 if successfully logged in; 0 otherwise | discrete |
| 12 | num\_compromised | number of ``compromised'' conditions | continuous |
| 13 | root\_shell | 1 if root shell is obtained; 0 otherwise | discrete |
| 14 | su\_attempted | 1 if ``su root'' command attempted; 0 otherwise | discrete |
| 15 | num\_root | number of ``root'' accesses | continuous |
| 16 | num\_file\_creations | number of file creation operations | continuous |
| 17 | num\_shells | number of shell prompts | continuous |
| 18 | num\_access\_files | number of operations on access control files | continuous |
| 19 | num\_outbound\_cmds | number of outbound commands in an ftp session | continuous |
| 20 | is\_host\_login | 1 if the login belongs to the ‘host’ list; 0 otherwise | discrete |
| 21 | is\_guest\_login | 1 if the login is a ‘guest’ login; 0 otherwise | discrete |
| 22 | count | number of connections to the same host as the current connection in the past two seconds | continuous |
| 23 | srv\_count | number of connections to the same service as the current connection in the past two seconds | continuous |
| 24 | serror\_rate | % of connections that have ``SYN'' errors | continuous |
| 25 | srv\_serror\_rate | % of connections that have ``SYN'' errors | continuous |
| 26 | rerror\_rate | % of connections that have ``REJ'' errors | continuous |
| 27 | srv\_rerror\_rate | % of connections that have ``REJ'' errors | continuous |
| 28 | same\_srv\_rate | % of connections to the same service | continuous |
| 29 | diff\_srv\_rate | % of connections to different services | continuous |
| 30 | srv\_diff\_host\_rate | % of connections to different hosts | continuous |
| 31 | dst\_host\_count | count of connections having same destination host | continuous |
| 32 | dst\_host\_srv\_count | count of connections having same dst host and using same service | continuous |
| 33 | dst\_host\_same\_srv\_rate | % of connections having same dst port and using same service | continuous |
| 34 | dst\_host\_diff\_srv\_rate | % of different services on current host | continuous |
| 35 | dst\_host\_same\_src\_port\_rate | % of connections to current host having same src port | continuous |
| 36 | dst\_host\_srv\_diff\_host\_rate | % of connections to same service coming from diff. hosts | continuous |
| 37 | dst\_host\_serror\_rate | % of connections to current host that have an S0 error | continuous |
| 38 | dst\_host\_srv\_serror\_rate | % of connections to current host and specified service that have an S0 error | continuous |
| 39 | dst\_host\_rerror\_rate | % of connections to current host that have an RST error | continuous |
| 40 | dst\_host\_srv\_rerror\_rate | % of connections to the current host and specified service that have an RST error | continuous |
| 41 | label | normal or attack | discrete |

1. **Instruction:**

* **IS5315\_Final**: This file generate the decision tree model and generate the final result in Json structure. The result will be assign to the file structure.js
* **Structure.js**: JavaScript file which include the Decision Tree model result in json type. The file has structure “structure = ‘json file’”.
* **TreeDiagram.html**: this file generates the tree in javascript. Getting the data source from structure.js and apply D3js to visualize the tree.

To use **TreeDiagram.html**, please put the to the cloud, in the same folder with IS5315\_Final. To make the file useable in local machine you can clon the shortcute for chrome (if you use chrome) on your desktop and then in the shortcut properties add the parameter as follow:

"C:\Program Files\Google\Chrome\Application\chrome.exe" --disable-web-security --user-data-dir="C:\tmpChromeSession"