Spark for Intrusion detection

Intro. Big Data Environment

CSC 5355– Project #3

Asmaa Dalil

- ***INTRODUCTON:***

Spark, a renowned open-source distributed computing platform, excels in handling large datasets efficiently. Paired with Databricks, a cloud-based facilitator for Spark applications, the process becomes streamlined.

To leverage Databricks for Spark cluster creation and application execution, follow these steps:

1. Register and log in to Databricks.

2. Create a Spark cluster through the "Clusters" section.

3. Configure cluster settings as needed.

4. Initiate a notebook via the "Workspace."

5. Import and upload CSV files as tables.

6. Choose Python and link to the created cluster for the notebook.

7. Begin writing and executing Spark code for seamless data processing and analysis.

Technologies used

- DataBricks

- Spark

- scikitLearn

- ***Data Exploration:***

```
import pyspark
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from graphviz import Source
import pandas as pd
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName("decisiontree").getOrCreate()
df_Train_Set = spark.read.format("csv").option("header","true").option("inferSchema","true").load("UNSW_NB15_training-set.csv")
df_Test_Set = spark.read.format("csv").option("header","true").option("inferSchema","true").load("UNSW_NB15_testing-set.csv")
```

- ***Check the Statistics***
  statistics confirmation from (Table 6. A part of UNSW-NB15 data set distribution.)

```
stats_train_set = df_Train_Set.groupBy("Attack_cat").count()
stats_train_set.show()

stats_test_set = df_Test_Set.groupBy("Attack_cat").count()
stats_train_set.show()
```

```
+--------------+-----+
|    Attack_cat|count|
+--------------+-----+
|         Worms|   44|
|     Shellcode|  378|
|       Fuzzers| 6062|
|      Analysis|  677|
|           DoS| 4089|
|Reconnaissance| 3496|
|      Backdoor|  583|
|      Exploits|11132|
|        Normal|37000|
|       Generic|18871|
+--------------+-----+

+--------------+-----+
|    Attack_cat|count|
+--------------+-----+
|         Worms|   44|
|     Shellcode|  378|
|       Fuzzers| 6062|
|      Analysis|  677|
|           DoS| 4089|
|Reconnaissance| 3496|
|      Backdoor|  583|
|      Exploits|11132|
|        Normal|37000|
|       Generic|18871|
+--------------+-----+
```

- ***Merging CSV Files in Spark:***

To combine two CSV files in Spark, the 'union' method within the 'SparkSession' class proves useful. By employing this method, which accepts two dataframes as parameters, we obtain a

new dataframe encompassing the rows from both dataframes. Subsequently, converting this Spark dataframe to a Pandas dataframe facilitates more effective data manipulation and visualization.

```
[127]: # Merge the two dataframes
spark_df = df_Train_Set.union(df_Test_Set)
# Convert the Spark dataframe to a Pandas dataframe
data = spark_df.toPandas()
```

```
[129]: #show 30 top rows
data.head(30)
```

[129]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_s
0	1	0.000011	udp	-	INT	2	0	496	0	90909.090200	...	1	2	0	0	0	
1	2	0.000008	udp	-	INT	2	0	1762	0	125000.000300	...	1	2	0	0	0	
2	3	0.000005	udp	-	INT	2	0	1068	0	200000.005100	...	1	3	0	0	0	
3	4	0.000006	udp	-	INT	2	0	900	0	166666.660800	...	1	3	0	0	0	
4	5	0.000010	udp	-	INT	2	0	2126	0	100000.002500	...	1	3	0	0	0	
5	6	0.000003	udp	-	INT	2	0	784	0	333333.321500	...	1	2	0	0	0	
6	7	0.000006	udp	-	INT	2	0	1960	0	166666.660800	...	1	2	0	0	0	
7	8	0.000028	udp	-	INT	2	0	1384	0	35714.285220	...	1	3	0	0	0	
8	9	0.000000	arp	-	INT	1	0	46	0	0.000000	...	2	2	0	0	0	
9	10	0.000000	arp	-	INT	1	0	46	0	0.000000	...	2	2	0	0	0	
10	11	0.000000	arp	-	INT	1	0	46	0	0.000000	...	2	2	0	0	0	
11	12	0.000000	arp	-	INT	1	0	46	0	0.000000	...	2	2	0	0	0	

- ***Data Cleaning:***

```
: print(data.dtypes)
```

```
proto              object
state              object
swin                int32
dwin                int32
trans_depth         int32
ct_srv_src          int32
ct_state_ttl        int32
ct_dst_ltm          int32
ct_src_dport_ltm    int32
ct_ftp_cmd          int32
is_sm_ips_ports     int32
label               int32
dtype: object
```

Find the categorical variables

To transform categorical data into numerical data within a Pandas dataframe, the `get_dummies` method proves effective. This function requires the dataframe and a list of columns containing categorical data, producing a new dataframe with the categorical columns converted into numerical format.

Here's an example of utilizing the `get_dummies` method to convert categorical data to numerical data in a Pandas dataframe: using the method, we convert the categorical data in the "proto" and "state" columns. Consequently, the dataframe will incorporate two additional columns,

"proto_index" and "state_index," containing the numerical representations corresponding to the "proto" and "state" columns.

convert the categorical data into numerical.

```
[137]: # One-hot encode original boolean columns
       #data = pd.get_dummies(data, columns=original_boolean_columns)
       data = pd.get_dummies(data, columns=["proto","state"])
```

```
[138]: data.head()
```

[138]:

	swin	dwin	trans_depth	ct_srv_src	ct_state_ttl	ct_dst_ltm	ct_src_dport_ltm	ct_ftp_cmd	is_sm_ips_ports	label	...	state_CLO	state_CON	state_ECO	state_FIN	state
0	0	0	0	2	2	1	1	0	0	0	...	False	False	False	False	
1	0	0	0	2	2	1	1	0	0	0	...	False	False	False	False	
2	0	0	0	3	2	1	1	0	0	0	...	False	False	False	False	
3	0	0	0	3	2	2	2	0	0	0	...	False	False	False	False	
4	0	0	0	3	2	2	2	0	0	0	...	False	False	False	False	

5 rows × 154 columns

- *Data Training and Testing:*

1- Splitting the data

To perform dataset splitting into training and test sets in Python, the `train_test_split` function from the `sklearn.model_selection` module proves to be highly effective. This function accepts the input data and target data as parameters and returns the split datasets.

In order to split the data into training and test sets with the test set comprising 20% of the data, we utilize the `train_test_split` function. This function requires three arguments: the input data, the target data, and the test set size. Specifically, we provide the input data by removing the 'label' column from the data using the `drop` method. The target data is then extracted from the 'label' column.

Splitting Data for Training and Testing:

Utilize sklearn's `train_test_split` function to divide the dataset into training and testing subsets, allocating 20% for testing.

```
[39]: ## Data splitting
      X_train, X_test, y_train, y_test = train_test_split(data.drop('label', axis=1), data['label'], test_size=0.2)
```

After executing the `train_test_split` function, four datasets are returned: the input data for the training set (`X_train`), the input data for the test set (`X_test`), the target data for the training set (`y_train`), and the target data for the test set (`y_test`).

2- Model Training:

Model Training

Train a Decision Tree model using sklearn's `DecisionTreeClassifier`, with the Gini impurity as the criterion.

```
# Train the Decision Tree model using Gini impurity
model = DecisionTreeClassifier(criterion='gini')
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

To train a Decision Tree model in Python, the `DecisionTreeClassifier` class from the `sklearn.tree` module proves to be a valuable tool. This class accepts various hyperparameters that govern the Decision Tree's behavior, such as the criterion for node splitting and the maximum depth of the tree.

To employ the `DecisionTreeClassifier` class for model training, follow these steps:

1. Create an instance of the `DecisionTreeClassifier`.

2. Specify the criterion for splitting nodes; for instance, setting it to 'gini' means the Decision Tree will use the Gini impurity measure to determine the attribute to split on at each node.

3. Utilize the `fit` method to train the Decision Tree model on the training data. This process fits the model to the data, enabling subsequent predictions on new data.

- *Dicision Tree Visualization:*

To visualize a Decision Tree model in Python, the `export_graphviz` function from the `sklearn.tree` module proves to be a valuable tool. This function takes the trained Decision Tree model as input and generates a dot file, which can be rendered and visualized using the `graphviz` library.

Here's a concise guide on visualizing a Decision Tree model:

1. Use the `export_graphviz` function to generate a dot file representing the trained Decision Tree model.

2. Utilize the `Source` class from the `graphviz` library to create a graphviz object representing the Decision Tree.

3. Use the `render` method to save the Decision Tree as a PDF file.

4. Use the `view` method to open the generated PDF file for visualization.

This process provides a visual representation of the Decision Tree model, where each node in the tree corresponds to a decision or prediction made by the model. The tree illustrates the different splits made at each node and the final predictions at the leaves, including information such as

Gini impurity value, the number of samples in the node, and whether it is categorized as "Normal" or "Attack."

In the root node, the "ct_state_ttl" column is featured with a Gini impurity of 0.499. The subsequent nodes include "state" with a Gini impurity of 0.107 and "dwin" with a Gini impurity of 0.418, both as children of "ct_state_ttl."

- *Confusion Matrix*

To calculate a confusion matrix in Python, you can leverage the `confusion_matrix` function from the `sklearn.metrics` module. This function requires both the true labels and the predicted labels as inputs, providing a confusion matrix that illustrates the count of correct and incorrect model predictions.

Here's a step-by-step guide on how to compute a confusion matrix:

1. Utilize the `predict` method to generate predictions on the test data.

2. Apply the `confusion_matrix` function to compute the confusion matrix based on the true and predicted labels.

The resulting confusion matrix output indicates the model's predictions on the test set:

- True Negatives (TN): 14082

- False Positives (FP): 4382

- False Negatives (FN): 816

- True Positives (TP): 31949

Using these values, various performance metrics can be calculated to gain insights into the model's effectiveness. For instance, accuracy can be computed by dividing the total number of correct predictions (TN + TP) by the total number of predictions (TN + TP + FP + FN). Precision, another metric, represents the ratio of true positives to the total number of positive predictions (TP + FP). Additionally, recall measures the ratio of true positives to the total number of actual positive instances (TP + FN).

```
# Compute and print the confusion matrix
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

```
Confusion matrix:
[[14388  4382]
 [  816 31949]]
```

Conslusion:

Spark is a powerful tool for distributed data processing, offering scalability and efficiency for working with large datasets. In intrusion detection, Spark's capabilities enable quick and efficient processing of extensive network traffic data. Its distributed and parallel nature, coupled with machine learning libraries like MLlib, facilitates the rapid training and evaluation of models for accurate predictions on new data. Overall, Spark proves valuable in enhancing intrusion detection systems by efficiently handling large datasets and enabling effective machine learning model training.