Exploratory Analytics on UNSW-NB15 Dataset

Naveen Babu Thota, thotanaven044@gmail.com

University of East London

Index:

2. Understanding the Dataset
3. Hive Queries for Analysis
    a. Tabular and Graphical representation
4. PySpark Analysis
    a. Statistical Analysis
        i. Descriptive Statistics
        ii. Correlation
        iii. Hypothesis Testing
        iv. Density Approximation
    b. Classifier
        i. Binary Classifier
        ii. Multi Class Classifier

Abstract

The intent of the project is to understand Big Data and the concepts around it. It is more of an exploratory analysis into the data that is provided. Goal is to understand both the how and the why of the various tools that are available for the purpose. We have worked with the very popular UNSW-NB15 dataset created an Australian lab for the purpose of Network Intrusion Detection. The exercise eased us into storing the data on HDFS HADOOP, querying it using Hive and classifying it using PySpark. We explored why SQL like concept of Hive makes life much easier when having to deal with Big Data. We also realised the capabilities of PySpark for data processing. An analysis of our observations has been provided as a part of this paper.

Understanding the Dataset

The dataset UNSW-NB15 was created primarily to help create models that would detect network intrusion. It has three files

1. One for the features of the data titled UNSW-NB15-features.csv
2. One for the data itself, more than 2 million records UNSW-NB15.csv
3. One for the list of events.
    a. Attack type
        i. SubCategories
            1. Count

This data in the 2nd file has the records that belong to Normal category and 9 types of attack categories, namely Fuzzers, Shellcode, Worm, Backdoor, Reconnaissance, Analysis, DoS, Exploits, Generic. However there is no information as to which Sub Categories these attacks fall under.

Following are some of the initial raw observations on the data:

1. This data has imbalance
    a. 90% for normal category
    b. 10% for the rest of the attacks
2. Some attacks have very less data points
3. There is some redundant data in the 2nd file as will be made evident through the analysis from here on.

Hive Queries

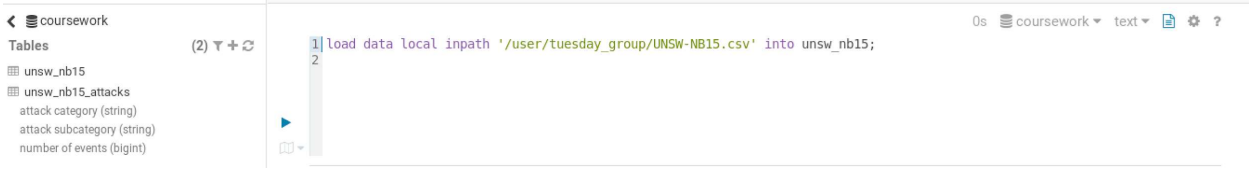
What, Why and How?

Hadoop has made it possible for us to quite easily store larger and larger datasets. However the ease of querying the data wasn't as it is today. One had to write processing jobs on MapReduce that took an advanced understanding of Java Programming to query the data. It wasn't really all that easily understandable for users across the organizations. All until one day someone at Facebook thought, what if we had the capability to query data on Hadoop through an SQL like query language. And as the thought materialized, it paved for the advent of Hive, an open source data warehouse system to query and analyze large sets of data that are primarily stored on Hadoop system. It opened up the data to anyone across the organization with basic SQL querying skills, thus leveraging the data more and better.

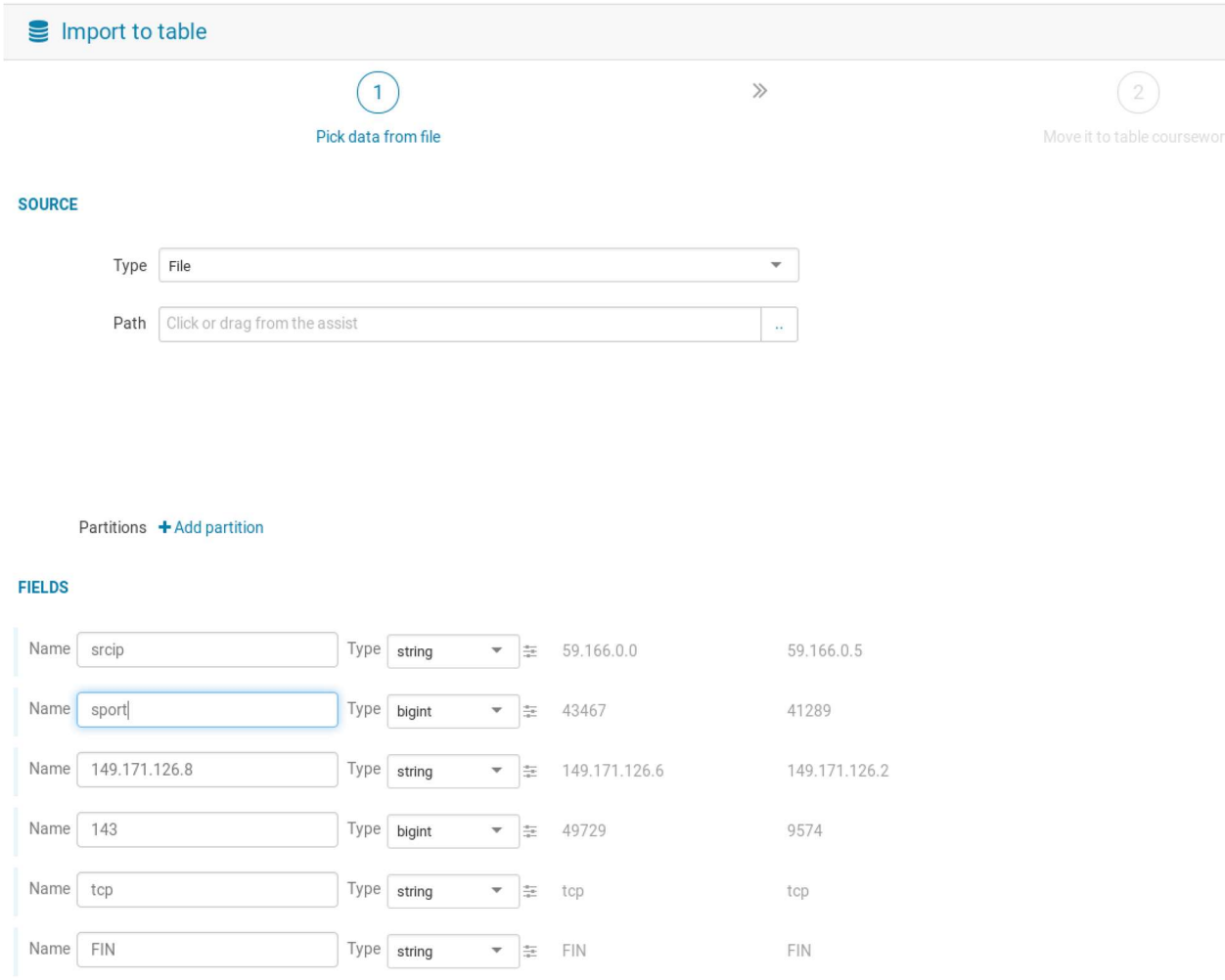
Hive basically has three functions: Data Summarization, Querying and Analysis.

Loading the data into Hive:

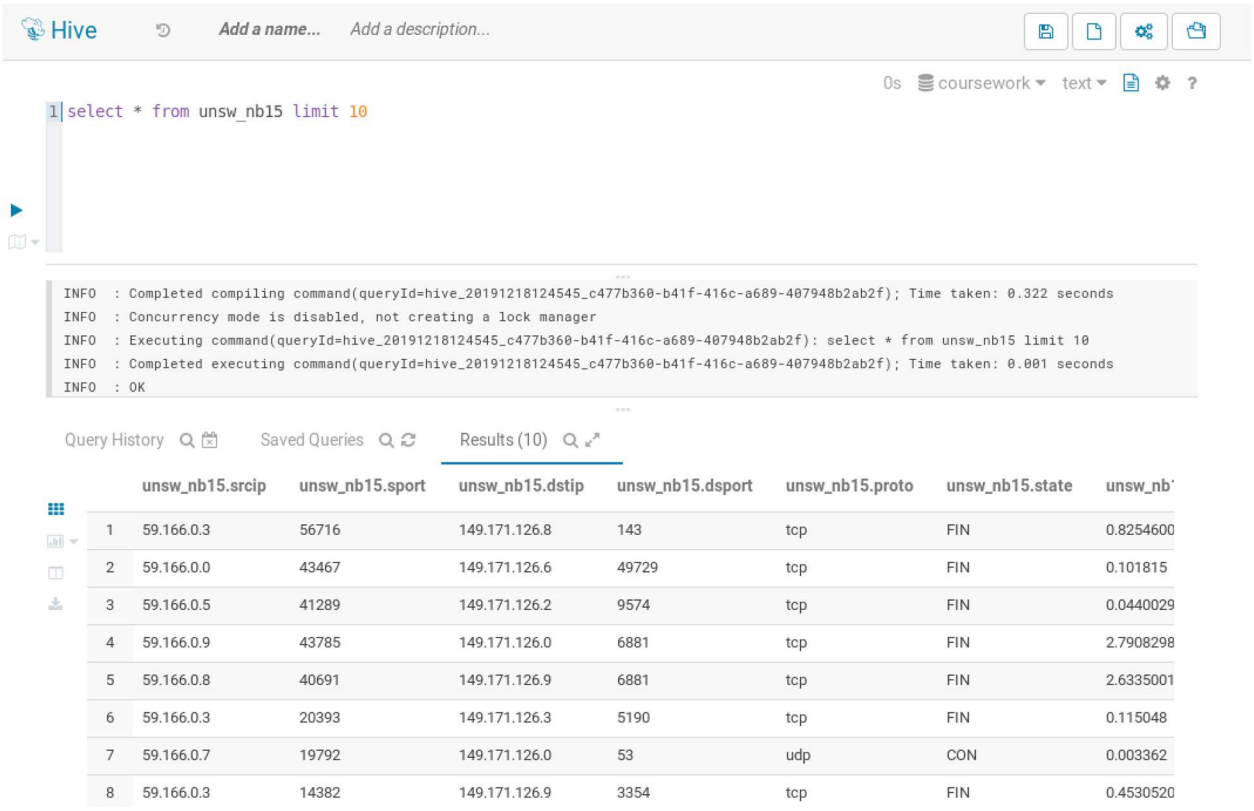
1. We will need two tables
    a. Unsw_nb15 traffic data
        i. Table named unsw_nb15
            1. 49 columns
                a. Header derived from features file
    b. Unsw_nb15 attack count data
        i. Table named unsw_nb15_attacks
            1. 3 columns
                a. Header is within the same file
2. Load Data using Load Command
    a. Create a table named unsw_nb15
    b. Load data file CSV into the table



3. We can either load data on to hive directly, by pressing '+' button and importing file as a table .
    a. Import to table, select the file
    b. Choose a table name
    c. Select field names and data types


Import to table
1
Pick data from file
2
Move it to table coursewor
SOURCE
Type
File
Path
Click or drag from the assist
..
Partitions
+ Add partition
FIELDS
Name
srcip
Type
string
59.166.0.0
59.166.0.5
Name
sport
Type
bigint
43467
41289
Name
149.171.126.8
Type
string
149.171.126.6
149.171.126.2
Name
143
Type
bigint
49729
9574
Name
tcp
Type
string
tcp
tcp
Name
FIN
Type
string
FIN
FIN

4. Check if the data is loaded correctly, by making sure it has both the proper Schema and Data



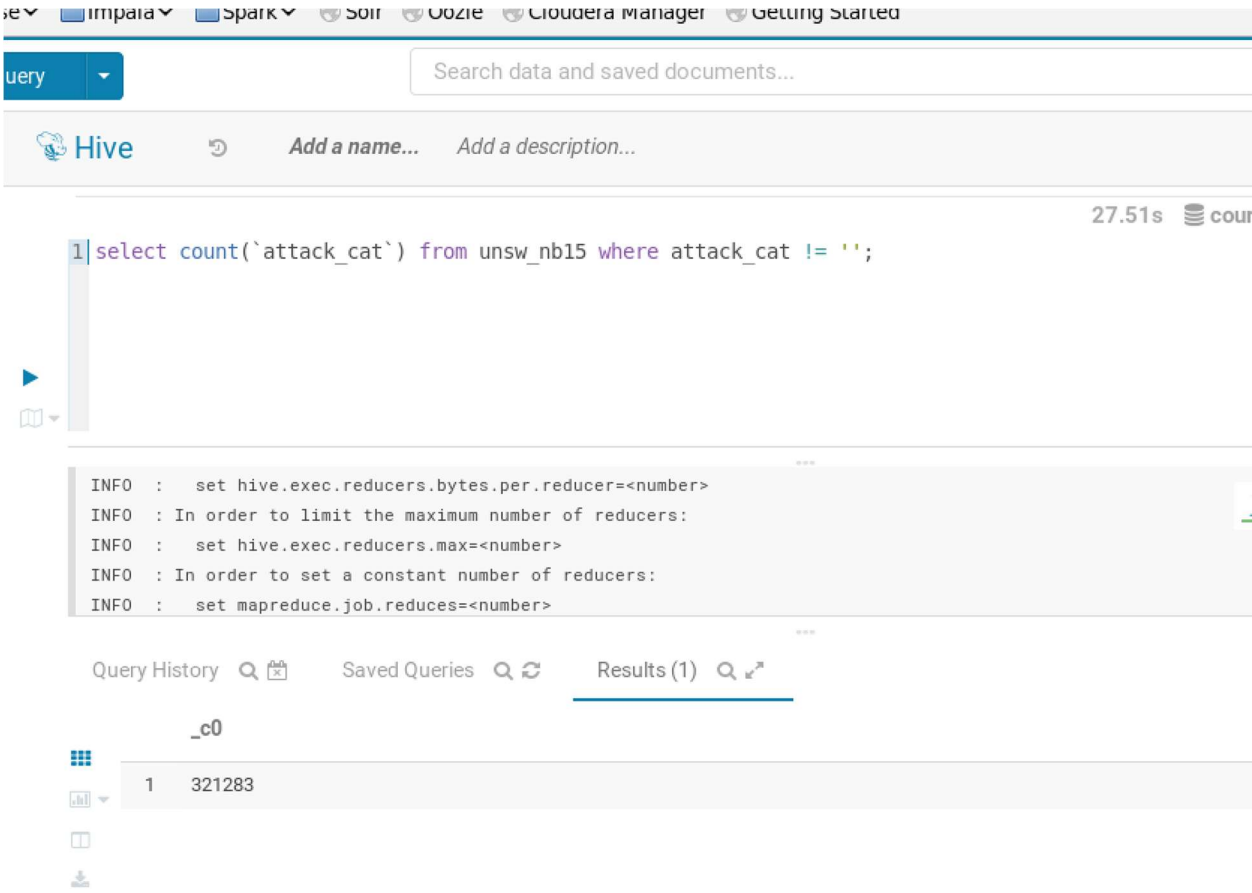
	unsw_nb15.srcip	unsw_nb15.sport	unsw_nb15.dstip	unsw_nb15.dsport	unsw_nb15.proto	unsw_nb15.state	unsw_nb
1	59.166.0.3	56716	149.171.126.8	143	tcp	FIN	0.8254600
2	59.166.0.0	43467	149.171.126.6	49729	tcp	FIN	0.101815
3	59.166.0.5	41289	149.171.126.2	9574	tcp	FIN	0.0440029
4	59.166.0.9	43785	149.171.126.0	6881	tcp	FIN	2.7908298
5	59.166.0.8	40691	149.171.126.9	6881	tcp	FIN	2.6335001
6	59.166.0.3	20393	149.171.126.3	5190	tcp	FIN	0.115048
7	59.166.0.7	19792	149.171.126.0	53	udp	CON	0.003362
8	59.166.0.3	14382	149.171.126.9	3354	tcp	FIN	0.4530520

Hive Queries

1. Preliminary Analysis

   a. Total number of records in unsw_nb15 with attack_cat as not empty, that is records falling within the 9 categories and not normal

Takeaway: Of about 2.5M total records, only a meagre 321283 records have attack details. Shows the imbalance of the data.



b. We wanted to get a list of attacks and the number of records for each to see the distribution by using count(attack_cat). Since in the main file, attack_cat is empty, we have replaced it with the string ‘Normal’ by using CASE statement

Takeaway: Though there are only 9 attacks, 1 normal categories, the number of records fetched by the following command were 14 meaning the data is unclean a bit. Like Fuzzers has two instances, one with “Fuzzers” the other with “Fuzzers ”,

Backdoor and Backdoors are same, Reconnaissance, shellcode have spaces at the end for some records.

```
SELECT
case
    when attack_cat=='' then "Normal"
    else attack_cat
end,
count(attack_cat) from unsw_nb15
group by attack_cat
```

```
INFO  :   set hive.exec.reducers.max=<number>
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : Starting Job = job_1575899805702_0070, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1575899805702_0070/
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1575899805702_0070
```

job_1575899805702_0070

Query History | Saved Queries | Results (14)

	_c0	_c1
1	Normal	2218456
2	Backdoor	1795
3	Exploits	44525
4	Reconnaissance	1759
5	Backdoors	534
6	DoS	16353
7	Shellcode	223
8	Analysis	2677
9	Fuzzers	5051
10	Fuzzers	19195
11	Generic	215481
12	Reconnaissance	12228
13	Shellcode	1288
14	Worms	174

c. From the understanding of the step, we realized we needed to clean the data a bit more to represent.

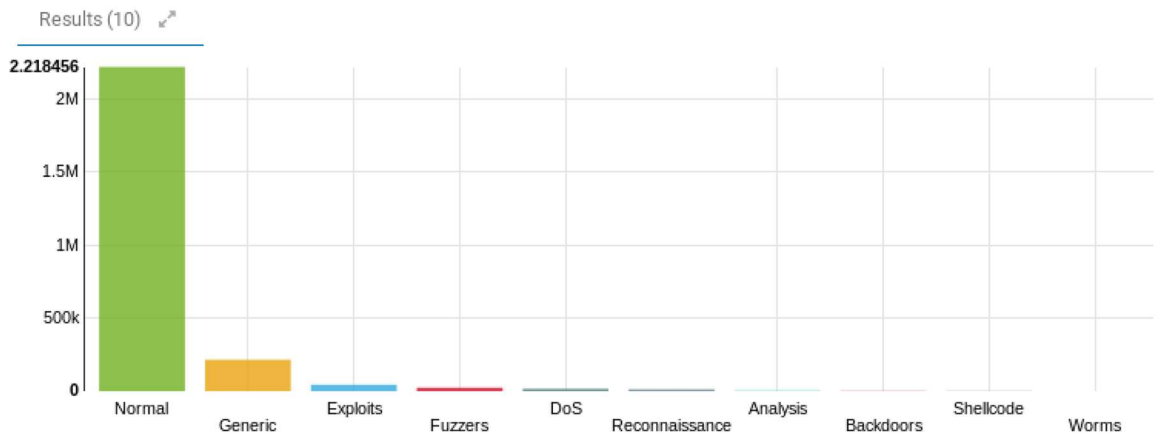
Takeway: We should clean the data as follows by using update statement if we were to analyze the data, but Hive doesn't let updates of tables. However, at least from the following statement grouping by like terms, it shows them as 10 entries.

Query:

```
SELECT
  CASE
    WHEN attack_cat LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN attack_cat LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN attack_cat LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN attack_cat LIKE '%Shellcode%'   THEN 'Shellcode'
    WHEN attack_cat==" then 'Normal'
    ELSE attack_cat
  END AS group_by_value,
COUNT(*) AS group_by_count
FROM unsw_nb15 a
GROUP BY
  CASE
    WHEN attack_cat LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN attack_cat LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN attack_cat LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN attack_cat LIKE '%Shellcode%'   THEN 'Shellcode'
    WHEN attack_cat==" then 'Normal'
    ELSE attack_cat
  END
```

```
SELECT
  CASE
    WHEN attack_cat LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN attack_cat LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN attack_cat LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN attack_cat LIKE '%Shellcode%'   THEN 'Shellcode'
    when attack_cat=='' then 'Normal'
    else attack_cat
  END AS group_by_value
, COUNT(*) AS group_by_count
FROM unsw_nb15 a
GROUP BY
  CASE
    WHEN attack_cat LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN attack_cat LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN attack_cat LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN attack_cat LIKE '%Shellcode%'   THEN 'Shellcode'
    when attack_cat=='' then 'Normal'
    else attack_cat
  END
```

1	Exploits	44525
2	Normal	2218456
3	Reconnaissance	13987
4	Backdoors	2329
5	DoS	16353
6	Shellcode	1511
7	Analysis	2677
8	Fuzzers	24246
9	Generic	215481
10	Worms	174



d. We have done the c step on unsw_nb15_attacks tables as well, but using sum instead of count as it has the count already within one of its columns “Number of events”

Takeaway: White the attack counts match in both of them, there are certain redundant records in one of the tables. For ex. Total Normal records 2218761-2218456=305 records not accounted for.

Query:

```
SELECT
 CASE
   WHEN  `attack category`  LIKE '%Fuzzers%'   THEN 'Fuzzers'
   WHEN `attack category` LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
   WHEN `attack category` LIKE '%Backdoor%'   THEN 'Backdoors'
   when `attack category`LIKE  '%normal%' then 'Normal'
   WHEN `attack category` LIKE '%Shellcode%'   THEN 'Shellcode'
   when `attack category` == '' then 'Total Records'
   else `attack category`
 END AS group_by_value,
SUM(`number of events`)
from unsw_nb15_attacks
GROUP BY
```

```
CASE
  WHEN  `attack category`  LIKE '%Fuzzers%'   THEN 'Fuzzers'
 WHEN `attack category` LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
 WHEN `attack category` LIKE '%Backdoor%'   THEN 'Backdoors'
 when `attack category`LIKE  '%normal%' then 'Normal'
 WHEN `attack category` LIKE '%Shellcode%'   THEN 'Shellcode'
 when `attack category` == '' then 'Total Records'
 else `attack category`
END
```

```
SELECT
  CASE
    WHEN  `attack category`  LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN `attack category` LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN `attack category` LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN `attack category` LIKE '%Shellcode%'   THEN 'Shellcode'
    when `attack category` == 'normal'  then 'Normal'
    when `attack category` == ''  then 'Total Records'
    else `attack category`
  END AS group_by_value
, sum( `number of events`) AS group_by_count
FROM unsw_nb15_attacks a
GROUP BY
  CASE
    WHEN  `attack category`  LIKE '%Fuzzers%'   THEN 'Fuzzers'
    WHEN `attack category` LIKE '%Reconnaissance%'   THEN 'Reconnaissance'
    WHEN `attack category` LIKE '%Backdoor%'   THEN 'Backdoors'
    WHEN `attack category` LIKE '%Shellcode%'   THEN 'Shellcode'
    when `attack category` == 'normal' then 'Normal'
    when `attack category` == ''  then 'Total Records'
    else `attack category`
  END
```

	group_by_value	group_by_count
1	Analysis	2677
2	Backdoors	2329
3	DoS	16353
4	Exploits	44525
	Fuzzers	24246
	Show row details	215481
7	Normal	2218761
8	Reconnaissance	13987
9	Shellcode	1511
10	Total Records	2540044
11	Worms	174

2. We wanted to identify how many records there are for each ip, port combination, both for the source and the destination. This was all an exercise to identify features that are relevant.

Takeaway: For each combination of srcip, sport there are about 3 records on average, and for dstip, dsport there are 5 records on average. Given there are about 692914 records in the first screenshot and 423196 in the second screenshot below. That's good enough sample to work with.

```
1 select distinct(concat(srcip,":",sport)) from unsw_nb15
```

```
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : Starting Job = job_1575899805702_0073, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1575899805702_007
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1575899805702_0073
```

Query History Saved Queries Results (692,914)

	_c0
1	10.40.170.2:0
2	10.40.182.1:36101
3	10.40.182.1:54977

```
1 select distinct(concat(dstip,":",dsport)) from unsw_nb15
```

```
INFO  : In order to set a constant number of reducers:
INFO  :   set mapreduce.job.reduces=<number>
INFO  : Starting Job = job_1575899805702_0074, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1575899805702_00
INFO  : Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1575899805702_0074
```

Query History | Saved Queries | Results (423,196)

	_c0
1	10.40.170.2:0
2	10.40.85.1:0
3	10.40.85.30:0
4	127.0.0.1:25
5	149.171.126.0:10003
6	149.171.126.0:10015
7	149.171.126.0:10018
8	149.171.126.0:10021

3. We wanted to check if there was any relation between sttl and dttl and the attack_cat

Takeaway: Most of the attacks happened when the difference was 254. There are 11 sttl-dttl values only where attacks happened.

When the difference is 254 most of the attacks were “Generic”

When the difference is -190 most of the attacks were “Exploits”

When the difference is 2 most of the attacks were “Fuzzers”

It’s an important observation, as changing time to live configuration could ideally reduce these particular attacks, or there could be alternative configuration done for such packets to prevent these attacks. Even for the PySpark classifier these would be important metrics.

Query: select distinct(sttl-dttl) from unsw_nb15

1	-190
2	-1
3	0
4	3
5	30
6	60
7	63
8	252
9	255
10	-254
11	-194
12	-191
13	-2
14	1
15	31
16	64
17	190
18	-189
19	-3
20	2
21	29
22	32

Query:

```
select
        sttl-dttl, count(sttl-dttl)
from unsw_nb15 where attack_cat != ''
group by
        sttl-dttl
```

Query History | Saved Queries | Results (11)

	_c0	_c1
1	-190	22518
2	0	375
3	3	6
4	255	29
5	-191	5
6	1	3
7	-189	1
8	2	35806
9	62	180
10	194	1095
11	254	261265

Query:

```
select
        attack_cat,(sttl-dttl),count(attack_cat)
from unsw_nb15
        where sttl-dttl=254
group by
         attack_cat, (sttl-dttl)
```

	attack_cat	_c1	_c2
1	Analysis	254	2044
2	DoS	254	12535
3	Exploits	254	16124
4	Fuzzers	254	6776
5	Shellcode	254	649
6	Reconnaissance	254	6209
7		254	10258
8	Backdoor	254	1440
9	Backdoors	254	482
10	Fuzzers	254	1940
11	Generic	254	212196
12	Reconnaissance	254	737
13	Shellcode	254	112
14	Worms	254	21

Query:

```
select
        attack_cat,(sttl-dttl),count(attack_cat)
from unsw_nb15
        where sttl-dttl=-190
group by
         attack_cat, (sttl-dttl)
```

ry History Saved Queries Results (10)

	attack_cat	_c1	_c2
1		-190	7254
2	Backdoor	-190	41
3	Backdoors	-190	6
4	Generic	-190	1378
5	Worms	-190	4
6	Analysis	-190	609
7	DoS	-190	1779
8	Exploits	-190	18606
9	Fuzzers	-190	55
10	Reconnaissance	-190	40

Mozilla F...

4. Join tables unsw_nb15 and unsw_nb15_attacks to check the data

Takeaway: Main table doesn't have sub category information, may be some analysis will help

get into it

Query:

```
select * from unsw_nb15
   inner join unsw_nb15_attacks on unsw_nb15.attack_cat = unsw_nb15_attacks. `attack
category`
```

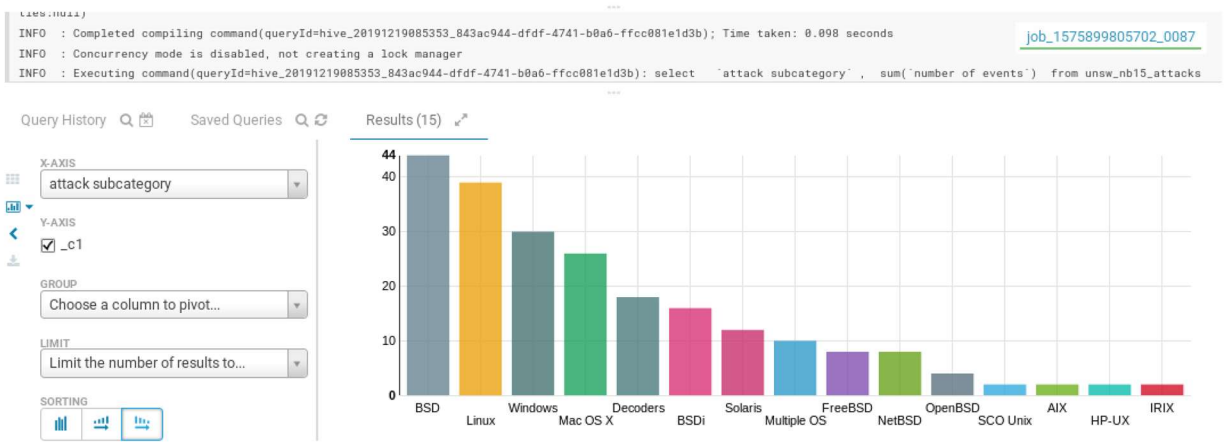
ry History Saved Queries Results (8,501,589)

	unsw_nb15.srcip	unsw_nb15.sport	unsw_nb15.dstip	unsw_nb15.dsport	unsw_nb15.proto	unsw_nb15.state	unsw_nb15.dur	unsw_nb15
1	59.166.0.3	56716	149.171.126.8	143	tcp	FIN	0.82546001999999996	7812
2	59.166.0.0	43467	149.171.126.6	49729	tcp	FIN	0.101815	4238
3	59.166.0.5	41289	149.171.126.2	9574	tcp	FIN	0.044002999000000001	2750
4	59.166.0.9	43785	149.171.126.0	6881	tcp	FIN	2.7908298999999999	10476
5	59.166.0.8	40691	149.171.126.9	6881	tcp	FIN	2.6335001	13350
6	59.166.0.3	20393	149.171.126.3	5190	tcp	FIN	0.115048	1958

5. Some visualization graphs for unsw_nb15_attacks

Query: (Shellcode has two entries as mentioned above hence the summing)

```
select `attack subcategory`, sum(`number of events` ) from unsw_nb15_attacks where `attack
category` == "Shellcode" or `attack category` == "Shellcode " group by `attack subcategory`
```



Query:
select `attack subcategory`, `number of events` from unsw_nb15_attacks where `attack category` == "Exploits"

We have written queries such as above to identify which of the features have more relevance to attack_cat and used the information for PySpark Classification.

Pyspark

What, why, how?

While Hadoop is more of a storage, spark is more of a distributed in-memory computing framework. Spark's API is primarily of Scala. But Spark also supports Java, Python, R as is evident from their documentation on the website. While we may write in different languages the logic for each of them, the execution happens the same.

PySpark is collaboration between Spark and Python. It makes it easier to do data processing. We can work with both structured and unstructured data using PySpark, if its unstructured we use RDD. If it is structured we can use DataFrame. Which is what we have done in the coursework as UNSW_NB15 is structured data.

Setup:

Downloaded the SPARK zip file from the Apache Spark website onto the device (MacBook Pro with Hadoop setup in our case). Set SPARK_HOME path in ~/.bash_profile. Jupyter as set up as a tool to work with. Created an alias snotebook to work with. System has Python 3.7.4 already. Anaconda set up is done already.

```
(base) UKC02YF3H0JG5H:~ npt03$ snotebook
[I 17:10:08.612 NotebookApp] The port 8888 is already in use, trying another por
t.
[I 17:10:08.613 NotebookApp] The port 8889 is already in use, trying another por
t.
[I 17:10:08.614 NotebookApp] The port 8890 is already in use, trying another por
t.
[I 17:10:08.615 NotebookApp] The port 8891 is already in use, trying another por
t.
[I 17:10:08.616 NotebookApp] The port 8892 is already in use, trying another por
t.
[I 17:10:08.935 NotebookApp] JupyterLab extension loaded from /opt/anaconda3/lib
/python3.7/site-packages/jupyterlab
[I 17:10:08.935 NotebookApp] JupyterLab application directory is /opt/anaconda3/
share/jupyter/lab
[I 17:10:08.937 NotebookApp] Serving notebooks from local directory: /Users/npt0
3
[I 17:10:08.937 NotebookApp] The Jupyter Notebook is running at:
[I 17:10:08.938 NotebookApp] http://localhost:8974/?token=b078d030275cad7cf6f9c7
32f90c719a3459fb394dacf526
[I 17:10:08.938 NotebookApp]  or http://127.0.0.1:8974/?token=b078d030275cad7cf6
```

Loading the data into PySpark:

To work with PySpark, we need to create a sparkcontext first.

```
from pyspark.sql import SQLContext

from pyspark import SparkContext

sc =SparkContext() #To create spark instance

sqlContext = SQLContext(sc) #To work with structured data

# inferSchema is true, so the schema and the datatypes are inferred

data = sqlContext.read.format('com.databricks.spark.csv').options(header='true',

inferschema='true').load('HADOOP_LOCATION/UNSW-NB15_1.csv')
```

data.show(5) #show the first five

```
data.show(5)
```

```
+----------+-----+-------------+------+-----+-----+-----------+------+------+----+----+-----+-----+-------+---------+
---------+-----+-----+----+----+----------+----------+-------+-------+-----------+-----------+---------+---------+---
-------+----------+---------+----------+------------+------------+------------+---------------+------------+---------
-------+----------+---------+----------+----------+----------+----------+---------------+---------------+-------
-------+----------+-----+
|     srcip|sport|        dstip|dsport|proto|state|        dur|sbytes|dbytes|sttl|dttl|sloss|dloss|service|    Sload|
Dload|Spkts|Dpkts|swin|dwin|     stcpb|     dtcpb|smeansz|dmeansz|trans_depth|res_bdy_len|     Sjit|     Djit|     St
ime|     Ltime|  Sintpkt|   Dintpkt|      tcprtt|      synack|      ackdat|is_sm_ips_ports|ct_state_ttl|ct_flw_http_m
thd|is_ftp_login|ct_ftp_cmd|ct_srv_src|ct_srv_dst|ct_dst_ltm|ct_src_ltm|ct_src_dport_ltm|ct_dst_sport_ltm|ct_dst_src_
ltm|attack_cat|Label|
+----------+-----+-------------+------+-----+-----+-----------+------+------+----+----+-----+-----+-------+---------+
---------+-----+-----+----+----+----------+----------+-------+-------+-----------+-----------+---------+---------+---
-------+----------+---------+----------+----------+----------+----------+---------------+---------------+-------
-------+----------+-----+
|59.166.0.3|56716|149.171.126.8|   143|  tcp|  FIN| 0.82546002|  7812| 16236|  31|  29|   30|   32|      -| 75090.25|
156111.73|  122|  126| 255| 255|2751097753|2748686736|     64|    129|          0|          0|445.25928| 474.9451|142
1970774|1421970775|6.8190908|  6.599896|5.9700001E-4|4.6899999E-4|     1.28E-4|              0|           0|
0|           0|         0|         2|         7|         1|         4|              1|              1|
1|      null|    0|
|59.166.0.0|43467|149.171.126.6| 49729|  tcp|  FIN|   0.101815|  4238| 65628|  31|  29|    7|   30|      -|328438.84|
5087030.5|   72|   74| 255| 255| 961515433|3225510659|     59|    887|          0|          0|      0.0|91.579567|142
1970775|1421970775| 1.429493|  1.387192|      6.8E-4|5.4600002E-4|     1.34E-4|              0|           0|
0|           0|         0|         7|         4|         1|         6|              1|              1|
1|      null|    0|
|59.166.0.5|41289|149.171.126.2|  9574|  tcp|  FIN|0.044002999|  2750| 29104|  31|  29|    7|   17|      -|488693.97|
5181101.5|   44|   48| 255| 255|3291096757|1191410228|     63|    606|          0|          0|78.126968|62.206562|142
1970775|1421970775| 1.014977|0.92583001|     0.00125|     4.85E-4|     7.65E-4|              0|           0|
0|           0|         0|         3|         5|         3|         3|              1|              1|
1|      null|    0|
|59.166.0.9|43785|149.171.126.0|  6881|  tcp|  FIN|  2.7908299| 10476|395734|  31|  29|   16|  143|      -|29863.518|
1130840.8|  180|  320| 255| 255|3934392726|3961690324|     58|   1237|          0|          0|2707.4927| 2018.976|142
1970772|1421970775|15.589459| 8.7470121|6.8400003E-4|5.3199998E-4|1.5199999E-4|              0|           0|
0|           0|         0|        11|         4|         3|         2|              1|              1|
1|      null|    0|
|59.166.0.8|40691|149.171.126.9|  6881|  tcp|  FIN|  2.6335001| 13350|548216|  31|  29|   21|  197|      -|40381.238|
1661560.6|  232|  438| 255| 255|   1518931|  18267719|     58|   1252|          0|          0|718.33679|500.57288|142
1970773|1421970775|11.399026| 6.0251832|     6.19E-4|     4.89E-4|      1.3E-4|              0|           0|
0|           0|         0|        16|         7|         7|         1|              1|              1|
1|      null|    0|
+----------+-----+-------------+------+-----+-----+-----------+------+------+----+----+-----+-----+-------+---------+
---------+-----+-----+----+----+----------+----------+-------+-------+-----------+-----------+---------+---------+---
```

data.printSchema()

```
data.printSchema()
```

```
root
 |-- srcip: string (nullable = true)
 |-- sport: integer (nullable = true)
 |-- dstip: string (nullable = true)
 |-- dsport: integer (nullable = true)
 |-- proto: string (nullable = true)
 |-- state: string (nullable = true)
 |-- dur: double (nullable = true)
 |-- sbytes: integer (nullable = true)
 |-- dbytes: integer (nullable = true)
 |-- sttl: integer (nullable = true)
 |-- dttl: integer (nullable = true)
 |-- sloss: integer (nullable = true)
 |-- dloss: integer (nullable = true)
 |-- service: string (nullable = true)
 |-- Sload: double (nullable = true)
 |-- Dload: double (nullable = true)
 |-- Spkts: integer (nullable = true)
 |-- Dpkts: integer (nullable = true)
 |-- swin: integer (nullable = true)
 |-- dwin: integer (nullable = true)
 |-- stcpb: long (nullable = true)
 |-- dtcpb: long (nullable = true)
 |-- smeansz: integer (nullable = true)
 |-- dmeansz: integer (nullable = true)
 |-- trans_depth: integer (nullable = true)
 |-- res_bdy_len: integer (nullable = true)
 |-- Sjit: double (nullable = true)
 |-- Djit: double (nullable = true)
 |-- Stime: integer (nullable = true)
 |-- Ltime: integer (nullable = true)
 |-- Sintpkt: double (nullable = true)
 |-- Dintpkt: double (nullable = true)
 |-- tcprtt: double (nullable = true)
 |-- synack: double (nullable = true)
 |-- ackdat: double (nullable = true)
 |-- is_sm_ips_ports: integer (nullable = true)
 |-- ct_state_ttl: integer (nullable = true)
 |-- ct_flw_http_mthd: integer (nullable = true)
 |-- is_ftp_login: integer (nullable = true)
 |-- ct_ftp_cmd: integer (nullable = true)
 |-- ct_srv_src: integer (nullable = true)
```

```
from pyspark.sql.functions import col # to be able to use SQL functions like count, aggregate etc

data.groupBy("attack_cat") \          #Grouping by attack_cat and getting count like in HIVE
  .count() \
  .orderBy(col("count").desc()) \
  .show()
```

```
+---------------+-------+
|     attack_cat|  count|
+---------------+-------+
|           null|2218456|
|        Generic| 215481|
|       Exploits|  44525|
|       Fuzzers |  19195|
|            DoS|  16353|
|Reconnaissance |  12228|
|        Fuzzers|   5051|
|       Analysis|   2677|
|       Backdoor|   1795|
| Reconnaissance|   1759|
|     Shellcode |   1288|
|      Backdoors|    534|
|      Shellcode|    223|
|          Worms|    174|
+---------------+-------+
```

Statistics:

data.describe().select("sttl","dttl","swin","dwin").show() # for statistics like mean, variance, std

```
]: data.describe().select("sttl","dttl","swin","dwin").show()
```

```
+-----------------+-----------------+------------------+------------------+
|             sttl|             dttl|              swin|              dwin|
+-----------------+-----------------+------------------+------------------+
|          2539739|          2539739|           2539739|           2539739|
|62.78149802007214|  30.770440978384|150.10695075360107|149.76404347060858|
|74.62669956848488|42.85192218951865|125.47913122211037|125.54062192608494|
|                0|                0|                 0|                 0|
|              255|              254|               255|               255|
+-----------------+-----------------+------------------+------------------+
```

dftemp = data #to clean the data, set NULL as normal, keep one Fuzzers etc
dftemp = dftemp.na.replace(['Fuzzers '], ['Fuzzers'], 'attack_cat')
dftemp = dftemp.na.replace(['Reconnaissance '], ['Reconnaissance'], 'attack_cat')
dftemp = dftemp.na.replace(['Backdoor'], ['Backdoors'], 'attack_cat')
dftemp = dftemp.na.replace(['Shellcode '], ['Shellcode'], 'attack_cat')
dftemp = dftemp.na.fill('Normal')

#Check the data is clean

```
dftemp.groupBy("attack_cat") \
    .count() \
    .orderBy(col("count").desc()) \
    .show()
```

```
+--------------+-------+
|    attack_cat|  count|
+--------------+-------+
|        Normal|2218456|
|       Generic| 215481|
|      Exploits|  44525|
|       Fuzzers|  24246|
|           DoS|  16353|
|Reconnaissance|  13987|
|      Analysis|   2677|
|     Backdoors|   2329|
|     Shellcode|   1511|
|         Worms|    174|
+--------------+-------+
```

#Analytics

import numpy as np
df = dftemp.rdd
mat = sc.parallelize(df.collect())

summary = Statistics.colStats(mat)
print(summary.mean()) # a dense vector containing the mean value for each column
print(summary.variance()) # column-wise variance
print(summary.numNonzeros())

```
print(Statistics.corr(mat, method="pearson"))

kd = KernelDensity()
kd.setSample(mat)
kd.setBandwidth(3.0)

# Find density estimates for the given values
densities = kd.estimate([-1.0, 2.0, 5.0])
```

Classifiers

We have created a new df with only some of the features that we identified from the above Hive and Pyspark analysis. This subset of features would mostly cover everything. Some of the features are not useful. If we were to use srcip or sport, we would too tightly defining the model and it wouldn't work for generic cases.

```
df = dftemp.select('proto', 'state',
'service','sttl','dttl','swin','dwin','sbytes','dbytes','attack_cat','Label' )
```

```
df = dftemp.select('proto', 'state', 'service','sttl','dttl','swin','dwin','sbytes','dbytes','attack_cat','Label' )
df.printSchema()
```

```
root
 |-- proto: string (nullable = false)
 |-- state: string (nullable = false)
 |-- service: string (nullable = false)
 |-- sttl: integer (nullable = true)
 |-- dttl: integer (nullable = true)
 |-- swin: integer (nullable = true)
 |-- dwin: integer (nullable = true)
 |-- sbytes: integer (nullable = true)
 |-- dbytes: integer (nullable = true)
 |-- attack_cat: string (nullable = false)
 |-- Label: integer (nullable = true)
```

Binary Classification:

We have used Label which has only 0 or 1 for normal or attack, to answer the question, was there an attack? But the following code should work even if we were to replace Label with any other variable, any other categorical column. It's a very generic piece of code.

StringIndexer, even if of string type categorical columns, we would be able to index them using StringIndexer. For example for Labels, it would index them into 0 and 1. For Attack_cat, it would index them into 0 to 9.

VectorAssembler, all the features we decide would help our model are assembled using this into a feature set to be used by the model. It is of Vector format.

We combine all of the categorical and numerical features.

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
categorytoclassify = [‘proto’, ‘state’, ‘service’]
stages = []
for categoricalCol in categorytoclassify:
    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol +
'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],
outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
label_stringIdx = StringIndexer(inputCol = 'Label', outputCol = 'labelindex')
stages += [label_stringIdx]
numericCols = ['sttl','dttl','swin','dwin','sbytes','dbytes']
assemblerInputs = [c + "classVec" for c in categorytoclassify] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
Cols = df.columns
print.cols()
```

```
cols = df.columns
print(cols)
```

```
['proto', 'state', 'service', 'sttl', 'dttl', 'swin', 'dwin', 'sbytes', 'dbytes', 'attack_cat', 'Label']
```

#Creating a pipeline and fitting the df into it, we are only passing the labelIndex and featureset we have created above and the cols chosen.

```
from pyspark.ml import Pipeline
```

```
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
finalcols = ['labelindex', 'features'] + cols
df = df.select(finalcols)
df.printSchema()
```

```
root
 |-- labelindex: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- proto: string (nullable = false)
 |-- state: string (nullable = false)
 |-- service: string (nullable = false)
 |-- sttl: integer (nullable = true)
 |-- dttl: integer (nullable = true)
 |-- swin: integer (nullable = true)
 |-- dwin: integer (nullable = true)
 |-- sbytes: integer (nullable = true)
 |-- dbytes: integer (nullable = true)
 |-- attack_cat: string (nullable = false)
 |-- Label: integer (nullable = true)
```

```
#the new df has labelindex and features vector as well. Splitting it
train, test = df.randomSplit([0.7, 0.3], seed = 100)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 1778668
Test Dataset Count: 761071
```

```
#Fitting the dataset into the pipeline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['labelindex', 'features'] + cols
df = df.select(selectedCols)
df.printSchema()
#Passing the dataset features and target label to the lr model and passing train dataset
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'labelindex', maxIter=10)
lrModel = lr.fit(train)
#Checking the model accuracy with the test dataset
```

```
predictions = lrModel.transform(test)
predictions.select('labelindex', 'probability').show(100)
```

```
: predictions = lrModel.transform(test)
  predictions.select('labelindex', 'prediction').show(100)
```

```
+----------+----------+
|labelindex|prediction|
+----------+----------+
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
+----------+----------+
```

#Grouping the test data by labelIndex and the actual prediction

```
predictions.groupBy("prediction") \
    .count() \
    .orderBy(col("count").desc()) \
    .show()
```

```
: predictions.groupBy("labelindex") \
      .count() \
      .orderBy(col("count").desc()) \
      .show()
```

```
+----------+------+
|labelindex| count|
+----------+------+
|       0.0|665040|
|       1.0| 96036|
+----------+------+
```

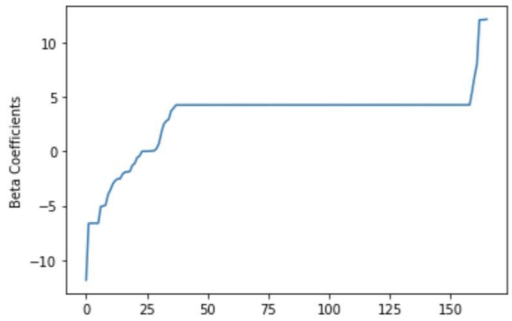
```
predictions.groupBy("prediction") \
    .count() \
    .orderBy(col("count").desc()) \
    .show()
```

```
+----------+------+
|prediction| count|
+----------+------+
|       0.0|657090|
|       1.0|103986|
+----------+------+
```

the accuracy of prediction is pretty high. Excep for 7950 records it has predicted rest of them accurately. Changing a few features might increase the accuracy.

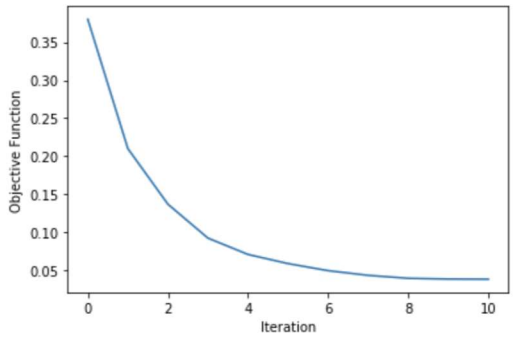
#Plotting beta coefficients

```
import matplotlib.pyplot as plt
import numpy as np
beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show();
```




```
trainingSummary = lrModel.summary

# Obtain the objective per iteration
objectiveHistory = trainingSummary.objectiveHistory
plt.plot(objectiveHistory)
plt.ylabel('Objective Function')
plt.xlabel('Iteration')
plt.show()
```



#Using binaryclassification evaluator to find the accuracy of the model

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator=BinaryClassificationEvaluator(rawPredictionCol="features",labelCol="labelindex")
predictions.select("labelindex","prediction","probability").show(5)
print("The area under ROC for train set is {}".format(evaluator.evaluate(train)))
print("The area under ROC for test set is {}".format(evaluator.evaluate(test)))
accuracy = evaluator.evaluate(predictions)
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator=BinaryClassificationEvaluator(rawPredictionCol="features",labelCol="labelindex")
predictions.select("labelindex","prediction","probability").show(5)
print("The area under ROC for train set is {}".format(evaluator.evaluate(train)))
print("The area under ROC for test set is {}".format(evaluator.evaluate(test)))
accuracy = evaluator.evaluate(predictions)
```

```
+----------+----------+--------------------+
|labelindex|prediction|         probability|
+----------+----------+--------------------+
|       0.0|       0.0|[0.99999976805861...|
|       0.0|       0.0|[0.99999976795561...|
|       0.0|       0.0|[0.99999976792127...|
|       0.0|       0.0|[0.99999976788692...|
|       0.0|       0.0|[0.99999976788692...|
+----------+----------+--------------------+
only showing top 5 rows

The area under ROC for train set is 0.9906049908070171
The area under ROC for test set is 0.9905752126393315
```

```
print(accuracy)
```

```
0.9905752126393315
```

Test error being 1-0.99057

Decision Tree Classifier:

Using the same feature set, but attack_cat as the labelIndex this time.

```
from pyspark.ml.feature import StringIndexer
l_indexer = StringIndexer(inputCol="attack_cat", outputCol="labelIndex")
df = l_indexer.fit(df).transform(df)
df.show(3)
# grouping by labelIndex, to check how many indices are created
df.groupBy("labelIndex").count().orderBy("labelIndex").show()
```

```
df.groupBy("labelIndex").count().orderBy("labelIndex").show()
```

```
+----------+-------+
|labelIndex|  count|
+----------+-------+
|       0.0|2218456|
|       1.0| 215481|
|       2.0|  44525|
|       3.0|  24246|
|       4.0|  16353|
|       5.0|  13987|
|       6.0|   2677|
|       7.0|   2329|
|       8.0|   1511|
|       9.0|    174|
+----------+-------+
```

```
#Split the dataset
(trainingData, testData) = df.randomSplit([0.7, 0.3])
#Importing the classifier and its evaluator
```

```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
#Training the model and passing the test data to it
dt = DecisionTreeClassifier(labelCol="labelIndex", featuresCol="features")
model = dt.fit(trainingData)
predictions = model.transform(testData)
#Checking the predictions
```

```
+----------+----------+
|prediction|labelIndex|
+----------+----------+
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
```

#Evaluating the accuracy

```
evaluator = MulticlassClassificationEvaluator(\
labelCol="labelIndex", predictionCol="prediction",\
metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))
print("accuracy = %g" % accuracy)
```

```
Test Error = 0.030948
accuracy = 0.969052
```

Random Forest Classifier:

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="labelIndex",\
featuresCol="features", numTrees=10)
model = rf.fit(trainingData)
predictions = model.transform(testData)
predictions.select("prediction", "labelIndex").show(5)evaluator =\
MulticlassClassificationEvaluator(labelCol="labelIndex",\
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

print("Accuracy = %g" % accuracy)

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="labelIndex",\
featuresCol="features", numTrees=10)
model = rf.fit(trainingData)
```

```
predictions = model.transform(testData)
```

```
predictions.select("prediction", "labelIndex").show(5)
```

```
+----------+----------+
|prediction|labelIndex|
+----------+----------+
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
+----------+----------+
only showing top 5 rows
```

```
evaluator =\
MulticlassClassificationEvaluator(labelCol="labelIndex",\
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
print("Accuracy = %g" % accuracy)
```

```
Test Error = 0.0344017
Accuracy = 0.965598
```

Conclusions

From the Hive Queries and PySpark models, it is our observation that the dataset is quite imbalanced and that accounts for the very high accuracy we see in our classifiers as well. If we were to undersample the dataset and use decided number of records from each of the categories, the model would be better to be used real-time in cyber security.

References

Susan Li (2018) Machine Learning with PySpark an

d MLlib — Solving a Binary Classification Problem, Medium

Dhiraj Rai (2018) Logistic Regression in Spark ML, Medium

Igor Bobriakov (2018) Practical Apache Spark in 10 minutes. Part 4 — MLlib, Medium

Basic Statistics - RDD-based API, Spark website