NYU | TANDON SCHOOL OF ENGINEERING

Big Data

Weather Data Analysis

A Scalable Machine Learning Rainfall Predictor

Name: Mohit Patel
Net ID: msp552
Email: msp552@nyu.edu

Name: Devarsh Mahida
Net ID: djm746
Email: djm746@nyu.edu

Name: Shivam Raval
Net ID: sgr341
Email: sgr341@nyu.edu

Table of Contents

1. Introduction

This dataset has the change in global surface from 2008 to 2018 which can be used to get deep data insights for finding which part of the globe has been affected the most based during these many years. This will eventually help to find that why only some locations are affected the most, and because of what specific reasons does this happened. For example, the change in global surface from 2008 to 2018 Near NYC area will be from good condition to bad because of exponential development in the urban department and huge number of industries. This can be used further to append with any other project to recommend best environmentally safe places to reside from the viewpoint of future climatic changes. Also, it can be used for predicting, what vegetation will be most likely to be flourish for more profit for farmers and have a better plan for future. Another prominent aspect can be analyzing the socio-economic effects of the climate change over time. As due to adverse effects of the weather on the geography and its implications on the demography can deeply dent the progress and growth of an economy. Coastal areas can witness this catastrophe in the form of tsunamis and Hurricanes, causing the routine life and trade to come to a standstill. Different regions face issues such as heat waves, famines, forest fires etc. causing exacting effect on the economy of the region. This shows the socio-economic impact of these climatic changes over time. Over a short span it may seem irrelevant to co-relate such weather implications to the social and economic conditions of the populace. But over time with this vast amount of conclusive data, this does not seem as a farfetched concept.

2. Motivation

The Rainfall prediction process is full of uncertainty and it's affected by many factors. Hence it is one of the important exertions in Weather Data Analysis.

A prediction model has been built that uses big data analytical capabilities and machine learning to predict the occurrence of rainfall.
The Biggest motivations were to achieve scalability (20 million records) and efficiency (Run time – 10 minutes) over a distributed Architecture which are the two most prominent aspects of Big Data.

3. Architecture

The complete data analysis was done in two stages 1) Local stand-alone Mode (0.75 Million records) and Nyu HPC cluster (20 million records). The architecture is as follows:

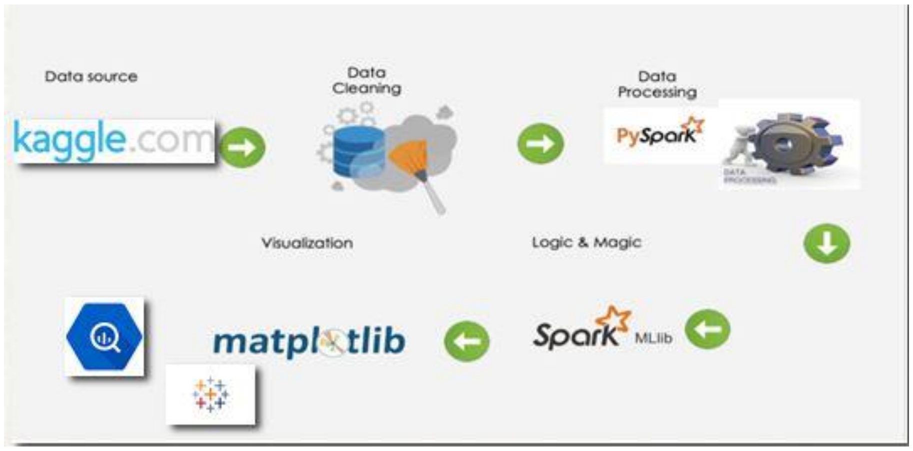
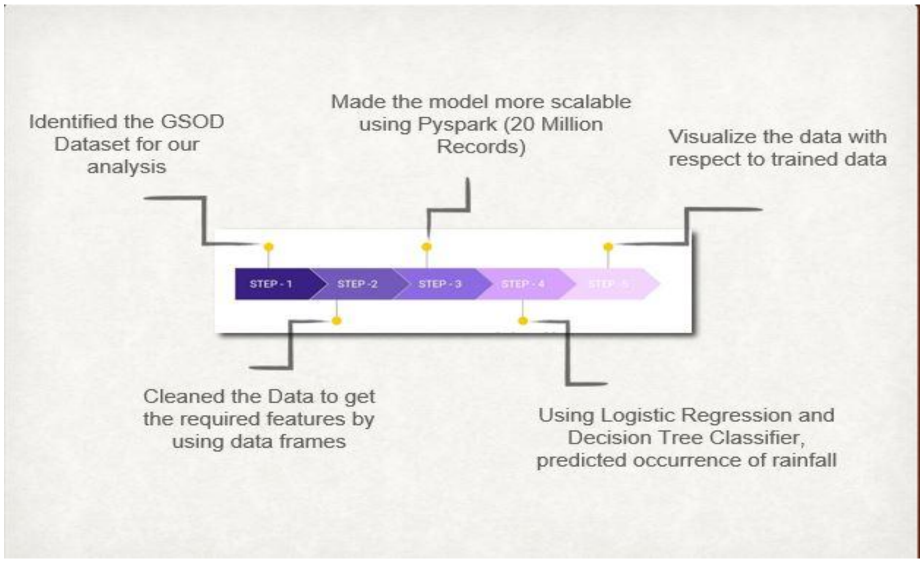


Fig 1. Pipeline Structure


Identified the GSOD Dataset for our analysis
Made the model more scalable using Pyspark (20 Million Records)
Visualize the data with respect to trained data
STEP - 1
STEP - 2
STEP - 3
STEP - 4
STEP - 5
Cleaned the Data to get the required features by using data frames
Using Logistic Regression and Decision Tree Classifier, predicted occurrence of rainfall

4. Dataset

We use NOAA Gsod Big query data set to access the weather data taken over 9000 weather stations across the globe of past 10 years (2008 - 2018). The Kaggle kernel API provides access to this data. Since the domain of our problem is change in Rainfall prediction, we decided to proceed with data records which have relevant and process-able feature columns.

Sample Dataset:

Weather.csv

4.1. Data Pre-processing and Cleaning

The data we extracted using Kaggle kernel API were messy and the column names were not consistent among different files for the same dataset.

Moreover, we had to deal with nulls, nan and missing values. All of them were represented differently, for e.g., nulls were represented as “nan”, “*”, “/”. The pre-processing included:

• Aggregating all the year data for the past 10 year into a single file (csv file format).

• Provided a single schema while merging the files.

• Imputed and removed nans, nulls and empty values according to the use case.

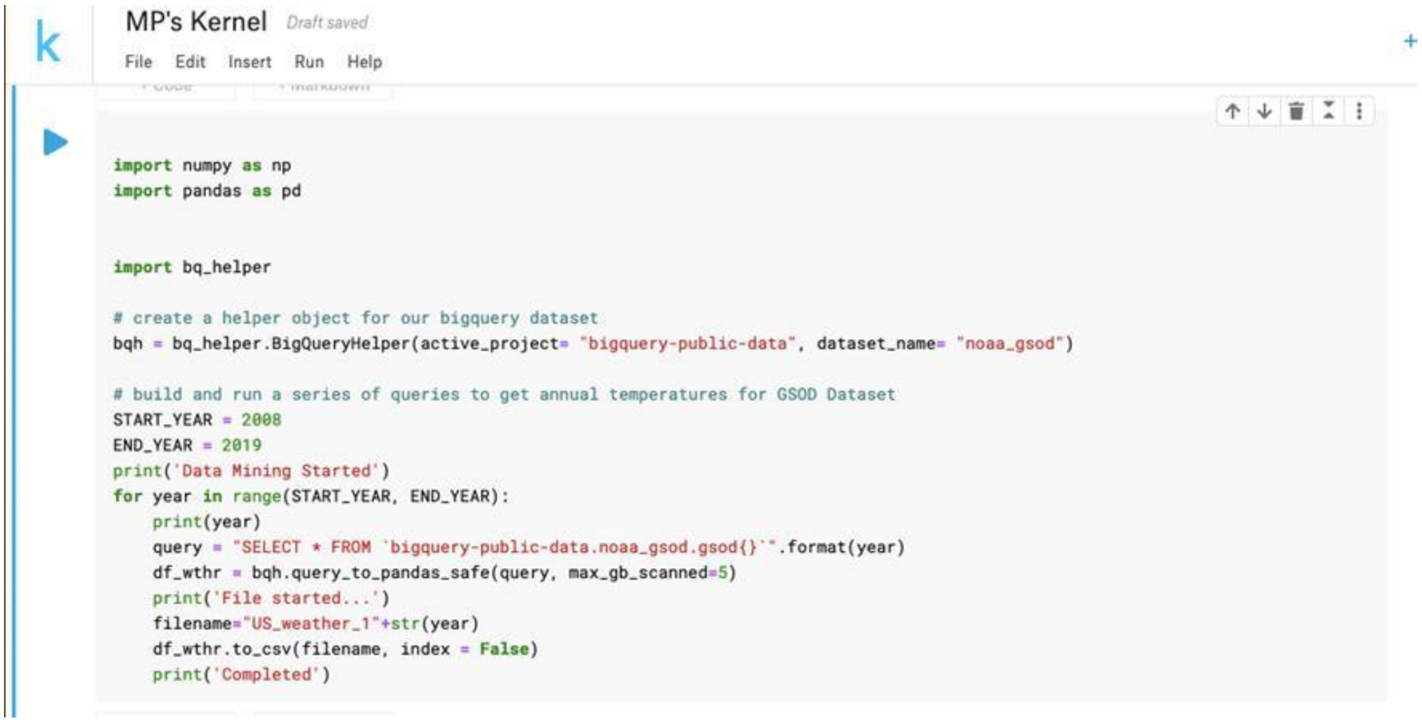
• Removed themetric sign, for e.g., $ and % sign from the column values so as to perform calculations on them.

• We used Pyspark Jupyter notebook to analyze the records of 9000 weather stations and took data from Kaggle Kernel API.

• In order to predict rainfall we decided to explore the max temperature, min temperature, precipitation, sea level pressure, snow depth etc. values from dataset.

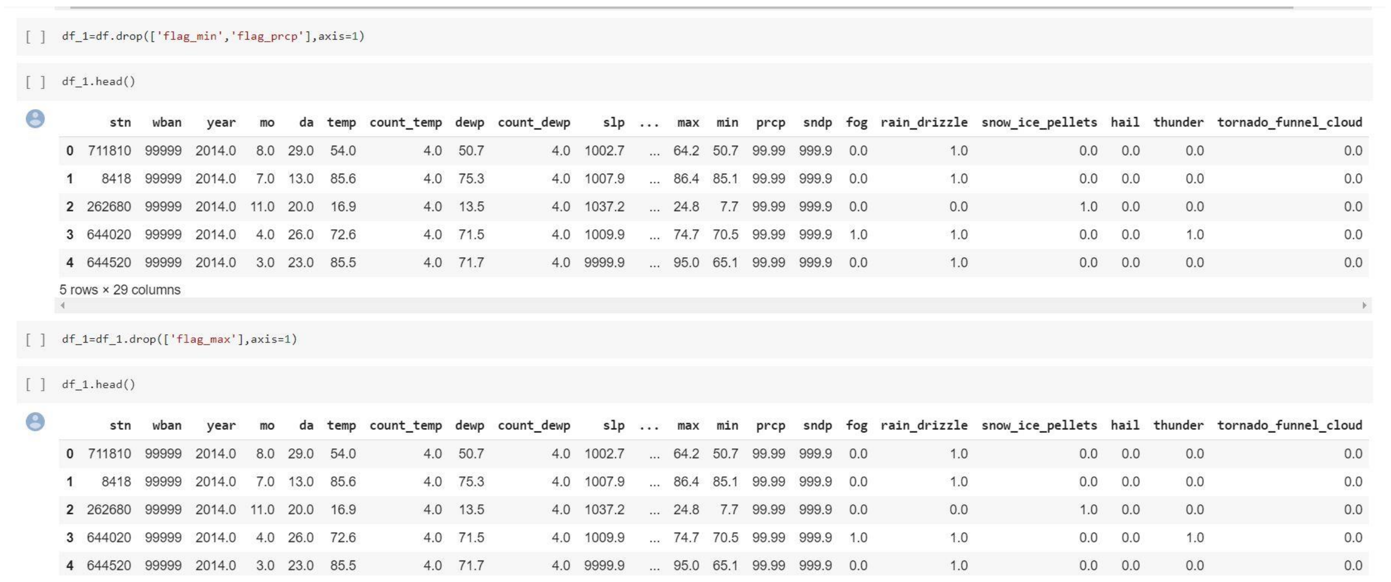
• Finally, we can predict the occurrence of rainfall as a Binary Classification.

Extracting Data from Big query data set using Kaggle Kernel and storing it into csv Files.



```
import numpy as np
import pandas as pd


import bq_helper

# create a helper object for our bigquery dataset
bqh = bq_helper.BigQueryHelper(active_project= "bigquery-public-data", dataset_name= "noaa_gsod")

# build and run a series of queries to get annual temperatures for GSOD Dataset
START_YEAR = 2008
END_YEAR = 2019
print('Data Mining Started')
for year in range(START_YEAR, END_YEAR):
    print(year)
    query = "SELECT * FROM `bigquery-public-data.noaa_gsod.gsod{}`".format(year)
    df_wthr = bqh.query_to_pandas_safe(query, max_gb_scanned=5)
    print('File started...')
    filename="US_weather_1"+str(year)
    df_wthr.to_csv(filename, index = False)
    print('Completed')
```

Dropping Columns with Missing / Nan Values.



```
[ ] df_1=df.drop(['flag_min','flag_prcp'],axis=1)
```

```
[ ] df_1.head()
```

	stn	wban	year	mo	da	temp	count_temp	dewp	count_dewp	slp	...	max	min	prcp	sndp	fog	rain_drizzle	snow_ice_pellets	hail	thunder	tornado_funnel_cloud
0	711810	99999	2014.0	8.0	29.0	54.0	4.0	50.7	4.0	1002.7	...	64.2	50.7	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0
1	8418	99999	2014.0	7.0	13.0	85.6	4.0	75.3	4.0	1007.9	...	86.4	85.1	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0
2	262680	99999	2014.0	11.0	20.0	16.9	4.0	13.5	4.0	1037.2	...	24.8	7.7	99.99	999.9	0.0	0.0	1.0	0.0	0.0	0.0
3	644020	99999	2014.0	4.0	26.0	72.6	4.0	71.5	4.0	1009.9	...	74.7	70.5	99.99	999.9	1.0	1.0	0.0	0.0	1.0	0.0
4	644520	99999	2014.0	3.0	23.0	85.5	4.0	71.7	4.0	9999.9	...	95.0	65.1	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 29 columns

```
[ ] df_1=df_1.drop(['flag_max'],axis=1)
```

```
[ ] df_1.head()
```

	stn	wban	year	mo	da	temp	count_temp	dewp	count_dewp	slp	...	max	min	prcp	sndp	fog	rain_drizzle	snow_ice_pellets	hail	thunder	tornado_funnel_cloud
0	711810	99999	2014.0	8.0	29.0	54.0	4.0	50.7	4.0	1002.7	...	64.2	50.7	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0
1	8418	99999	2014.0	7.0	13.0	85.6	4.0	75.3	4.0	1007.9	...	86.4	85.1	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0
2	262680	99999	2014.0	11.0	20.0	16.9	4.0	13.5	4.0	1037.2	...	24.8	7.7	99.99	999.9	0.0	0.0	1.0	0.0	0.0	0.0
3	644020	99999	2014.0	4.0	26.0	72.6	4.0	71.5	4.0	1009.9	...	74.7	70.5	99.99	999.9	1.0	1.0	0.0	0.0	1.0	0.0
4	644520	99999	2014.0	3.0	23.0	85.5	4.0	71.7	4.0	9999.9	...	95.0	65.1	99.99	999.9	0.0	1.0	0.0	0.0	0.0	0.0

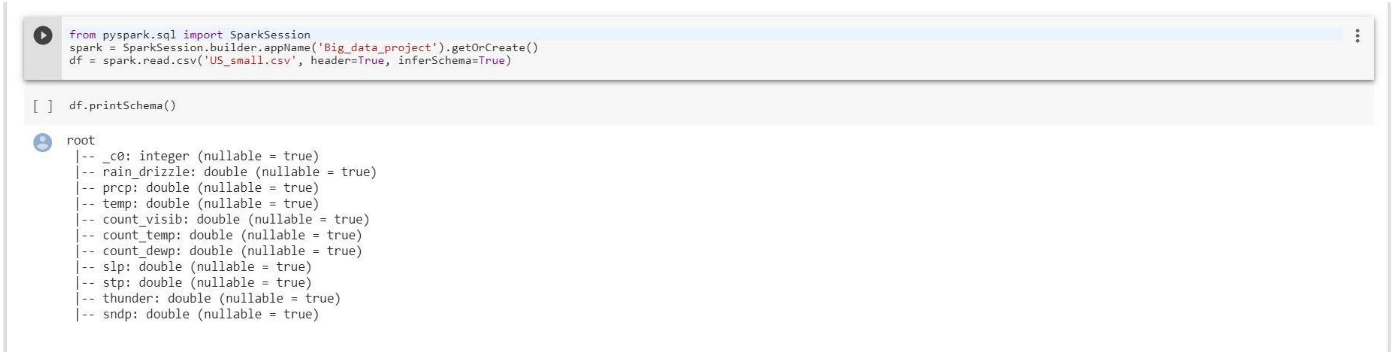
5. Machine Learning – SparkML (Local standalone Mode)

ML was an integral part of this project as conventional methods fail to predict and have their own limitations.

We used Logistic Regression classifier and Decision Tree Classifier as our choice of model because they give comparatively better fit ROC curves when compared to most other models. We extensively used MLlib in our project because it fit into Spark's APIs and interoperates with Pandas in Python.

Outline of steps followed:

- Data procurement :-

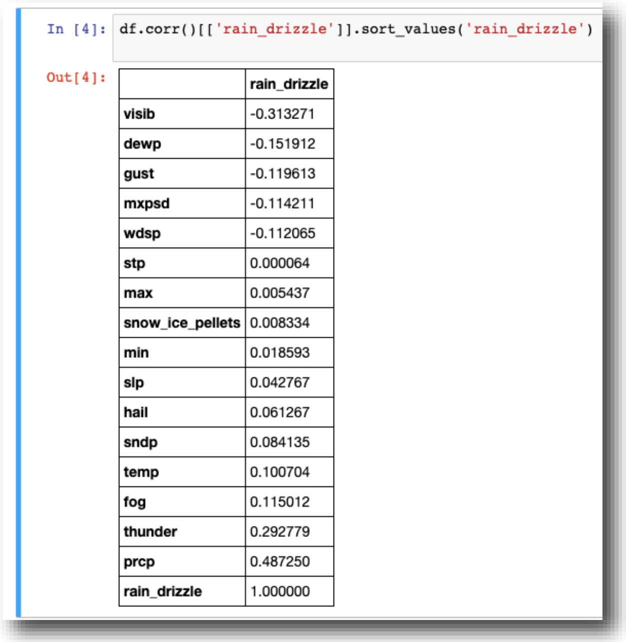


```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Big_data_project').getOrCreate()
df = spark.read.csv('US_small.csv', header=True, inferSchema=True)
```

```
[ ]  df.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- rain_drizzle: double (nullable = true)
 |-- prcp: double (nullable = true)
 |-- temp: double (nullable = true)
 |-- count_visib: double (nullable = true)
 |-- count_temp: double (nullable = true)
 |-- count_dewp: double (nullable = true)
 |-- slp: double (nullable = true)
 |-- stp: double (nullable = true)
 |-- thunder: double (nullable = true)
 |-- sndp: double (nullable = true)
```

- Feature Selection:-
  - ➢ Selecting features on the basis of their correlation with the target variable. Features having negative Correlation factor were not considered for prediction.

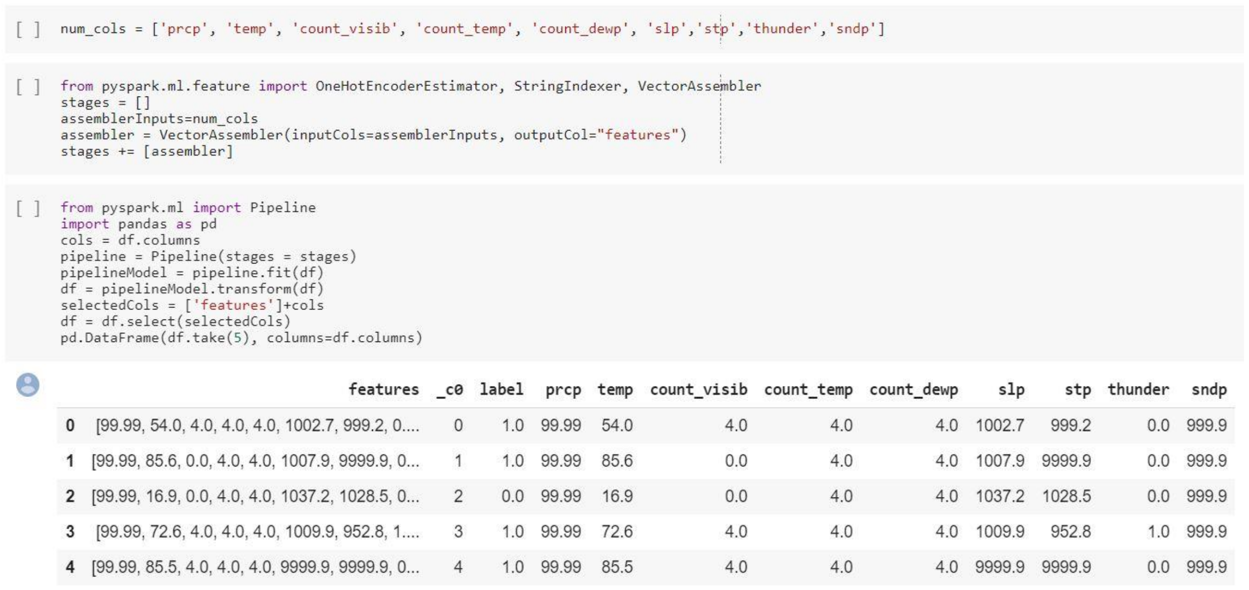


```
In [4]: df.corr()[['rain_drizzle']].sort_values('rain_drizzle')
```

Out[4]:

	rain_drizzle
visib	-0.313271
dewp	-0.151912
gust	-0.119613
mxpsd	-0.114211
wdsp	-0.112065
stp	0.000064
max	0.005437
snow_ice_pellets	0.008334
min	0.018593
slp	0.042767
hail	0.061267
sndp	0.084135
temp	0.100704
fog	0.115012
thunder	0.292779
prcp	0.487250
rain_drizzle	1.000000

- Data processing:-
  - ➢ There are no categorical columns so we do not need to transform them using One Hot Encoder.
  - ➢ Selecting the predictors and assembling them as a feature vector.
  - ➢ Recording each operation in stages and creating a pipeline so that we can run a combination of models i.e. Logistic Regression and Decision Tree Classifier in our case.



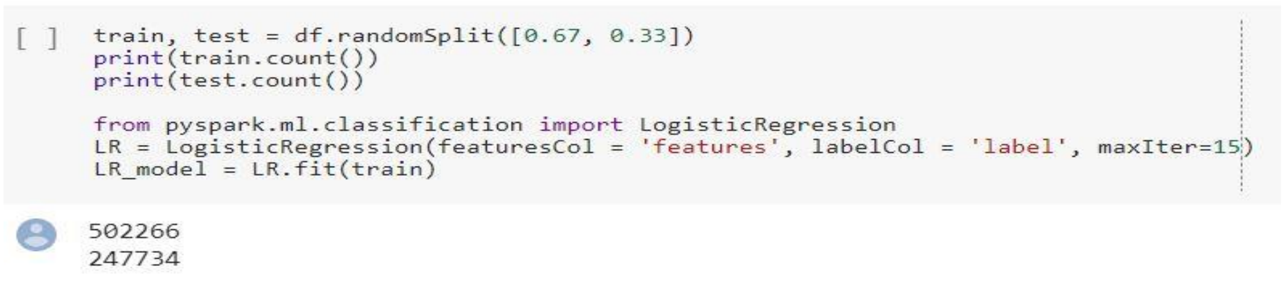
```
[ ]  num_cols = ['prcp', 'temp', 'count_visib', 'count_temp', 'count_dewp', 'slp','stp','thunder','sndp']
```

```
[ ]  from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, VectorAssembler
     stages = []
     assemblerInputs=num_cols
     assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
     stages += [assembler]
```

```
[ ]  from pyspark.ml import Pipeline
     import pandas as pd
     cols = df.columns
     pipeline = Pipeline(stages = stages)
     pipelineModel = pipeline.fit(df)
     df = pipelineModel.transform(df)
     selectedCols = ['features']+cols
     df = df.select(selectedCols)
     pd.DataFrame(df.take(5), columns=df.columns)
```

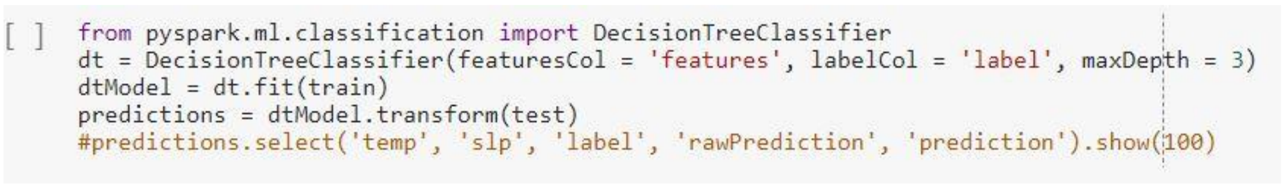
```
                                      features  _c0  label   prcp  temp  count_visib  count_temp  count_dewp     slp     stp  thunder   sndp
0   [99.99, 54.0, 4.0, 4.0, 4.0, 1002.7, 999.2, 0....    0    1.0  99.99  54.0          4.0         4.0         4.0  1002.7   999.2      0.0  999.9
1  [99.99, 85.6, 0.0, 4.0, 4.0, 1007.9, 9999.9, 0...    1    1.0  99.99  85.6          0.0         4.0         4.0  1007.9  9999.9      0.0  999.9
2  [99.99, 16.9, 0.0, 4.0, 4.0, 1037.2, 1028.5, 0...    2    0.0  99.99  16.9          0.0         4.0         4.0  1037.2  1028.5      0.0  999.9
3   [99.99, 72.6, 4.0, 4.0, 4.0, 1009.9, 952.8, 1....    3    1.0  99.99  72.6          4.0         4.0         4.0  1009.9   952.8      1.0  999.9
4  [99.99, 85.5, 4.0, 4.0, 4.0, 9999.9, 9999.9, 0...    4    1.0  99.99  85.5          4.0         4.0         4.0  9999.9  9999.9      0.0  999.9
```

- Training the model:-
  - ➢ We split the data set to get a train test split of 67:33.
  - ➢ Training the logistic regression model to predict rainfall.
  - ➢ Note : The test-train count is less because this performance analysis was on the 0.75 million Data records chosen randomly in local stand-alone mode.



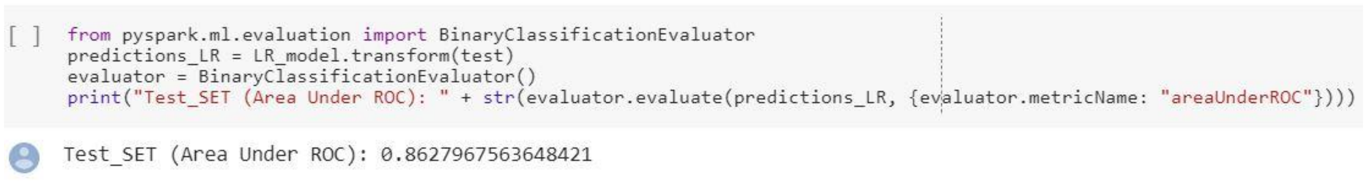
```
[ ]  train, test = df.randomSplit([0.67, 0.33])
     print(train.count())
     print(test.count())

     from pyspark.ml.classification import LogisticRegression
     LR = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=15)
     LR_model = LR.fit(train)
```

```
502266
247734
```



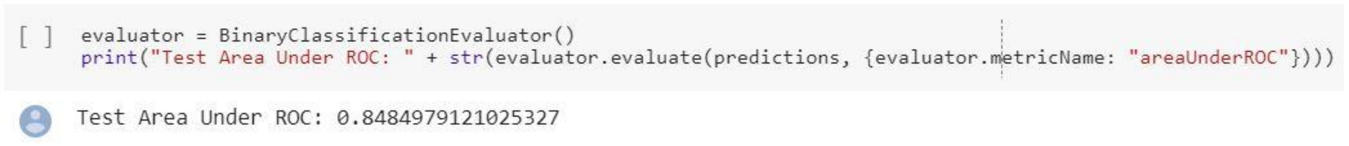
```
[ ]  from pyspark.ml.classification import DecisionTreeClassifier
     dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
     dtModel = dt.fit(train)
     predictions = dtModel.transform(test)
     #predictions.select('temp', 'slp', 'label', 'rawPrediction', 'prediction').show(100)
```

- Testing:-
  - ➢ Using the Binary Classification evaluator getting the testing Accuracy via AUC (Area under Curve) of the ROC curve for logistic Regression.



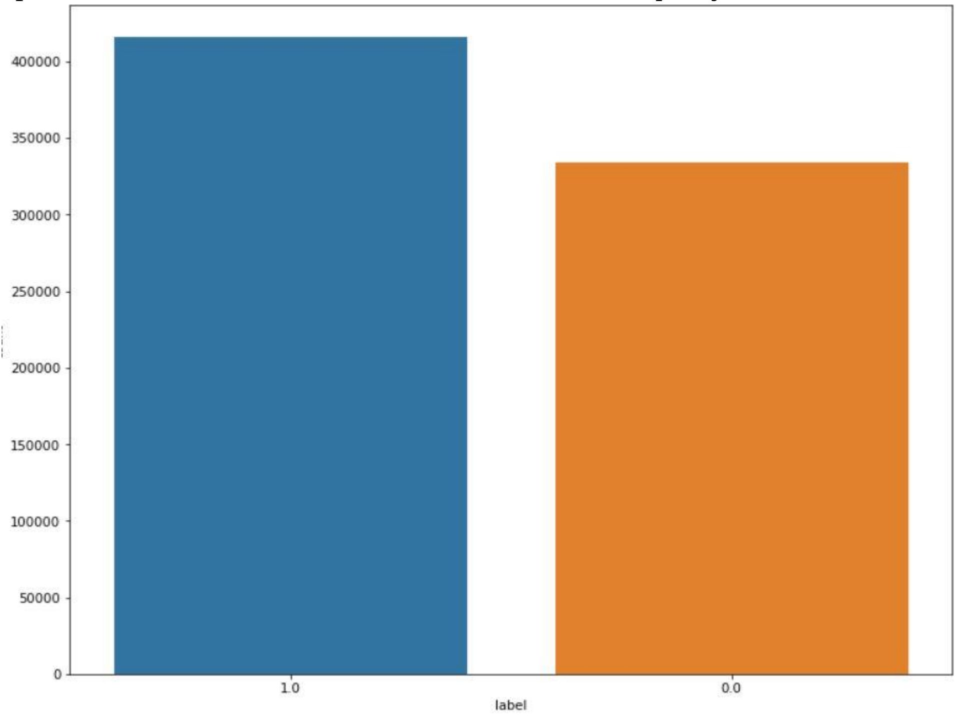
```
[ ]  from pyspark.ml.evaluation import BinaryClassificationEvaluator
     predictions_LR = LR_model.transform(test)
     evaluator = BinaryClassificationEvaluator()
     print("Test_SET (Area Under ROC): " + str(evaluator.evaluate(predictions_LR, {evaluator.metricName: "areaUnderROC"})))

     Test_SET (Area Under ROC): 0.8627967563648421
```

  - ➢ Using the Binary Classification evaluator getting the testing Accuracy via AUC (Area under Curve) of the ROC curve for Decision Tree Classifier.



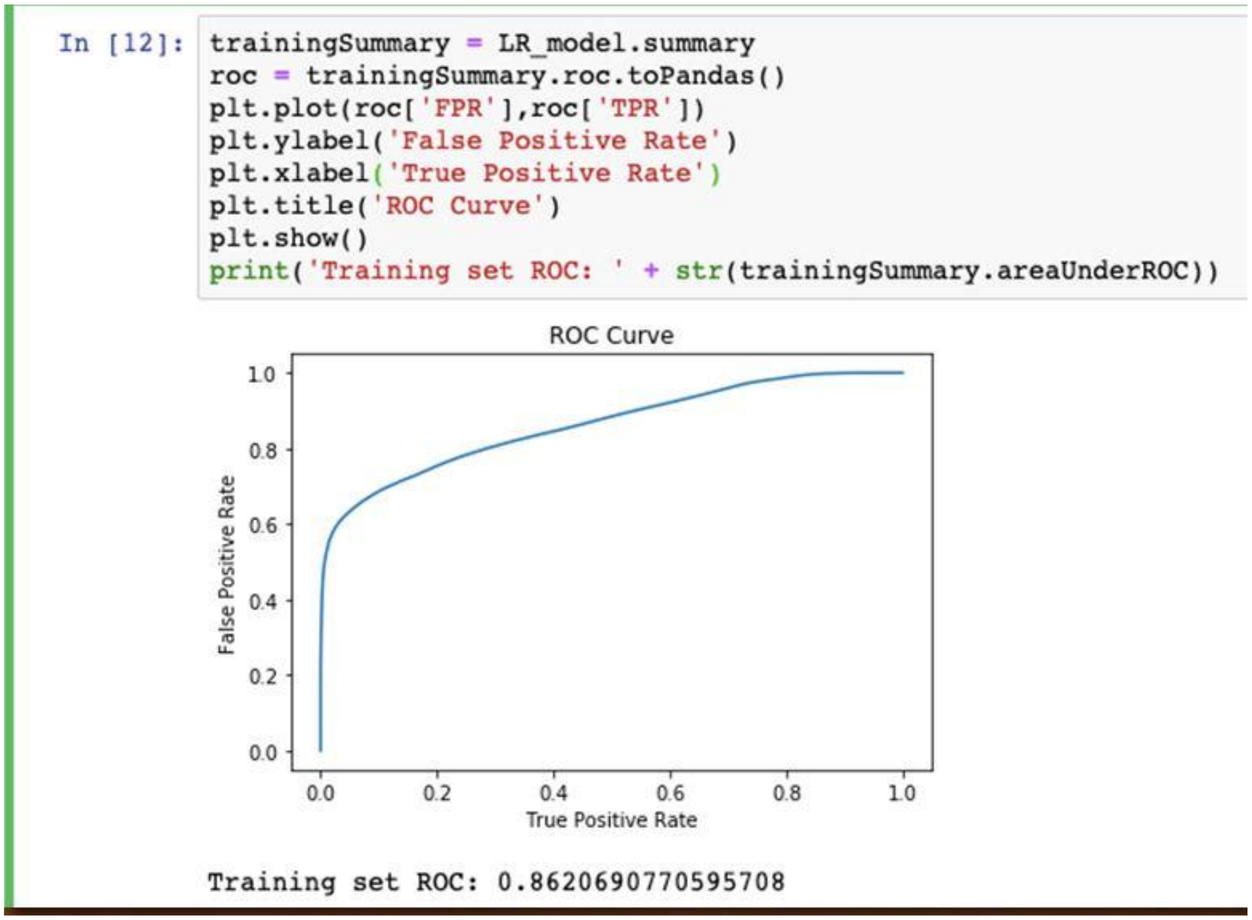
```
[ ]  evaluator = BinaryClassificationEvaluator()
     print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))

     Test Area Under ROC: 0.8484979121025327
```

- Conclusions from Analysis on local Standalone Mode:-
  - ➢ Test accuracy for both models in the local stand-alone mode is quite high.
  - ➢ Logistic Regression – 86%
  - ➢ Decision Tree Classifier – 84%
  - ➢ The main reason for this is the even distribution of rainfall in these set of records.
  - ➢ The following visualization would map the number of rainfall occurrences across the small data splice and we can infer that the rainfall is almost equally distributed over it.



As we can see the number of occurrences of rainfall (approx. 400000) and none occurrences (approx. 350000) are close.

- Visualization on the training Data on local node:-

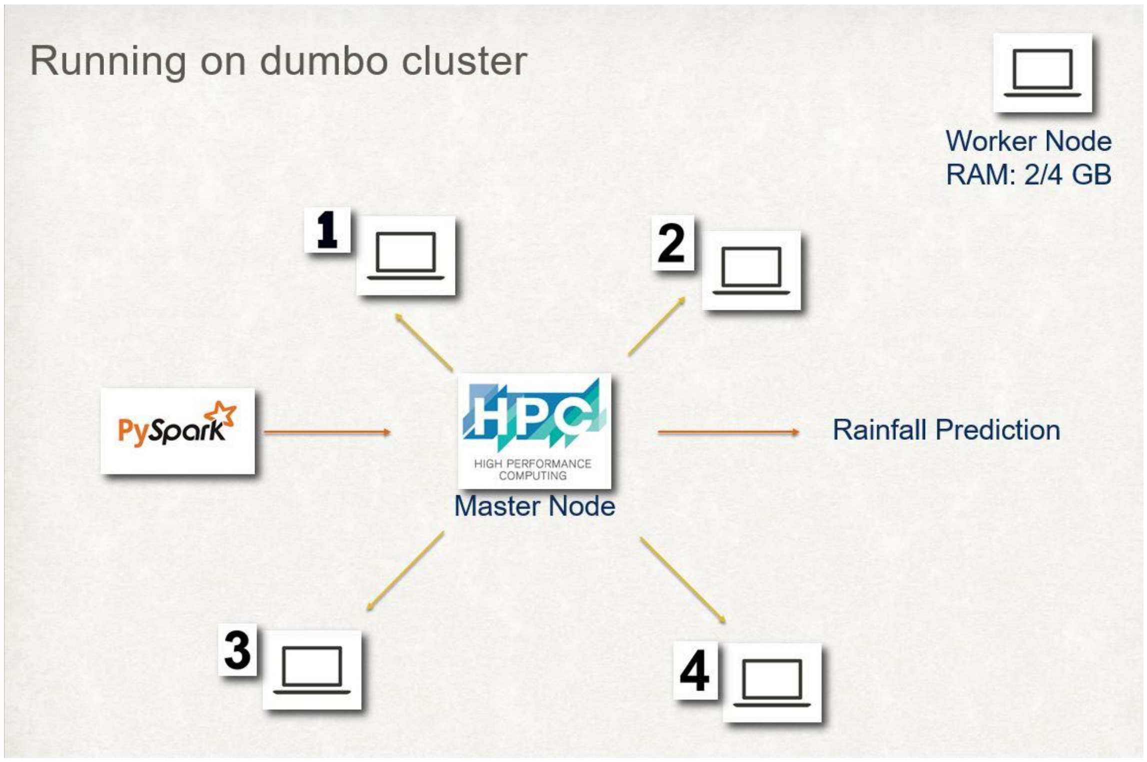


```
In [12]: trainingSummary = LR_model.summary
         roc = trainingSummary.roc.toPandas()
         plt.plot(roc['FPR'],roc['TPR'])
         plt.ylabel('False Positive Rate')
         plt.xlabel('True Positive Rate')
         plt.title('ROC Curve')
         plt.show()
         print('Training set ROC: ' + str(trainingSummary.areaUnderROC))
```

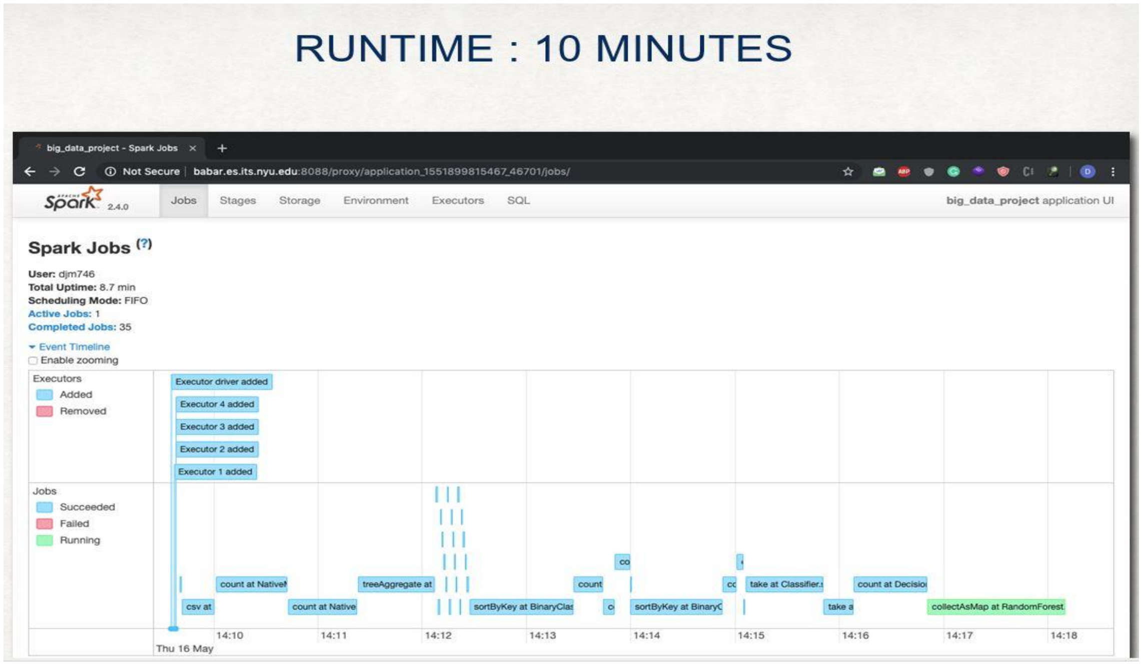
```
Training set ROC: 0.8620690770595708
```

6. Machine Learning – SparkML (NYU HPC Dumbo Cluster)

- In order to scale the model to a much bigger Data set and procure more efficiency, we performed the analysis on a data set of 20 million records on Nyu's high performance Computing Dumbo Cluster.



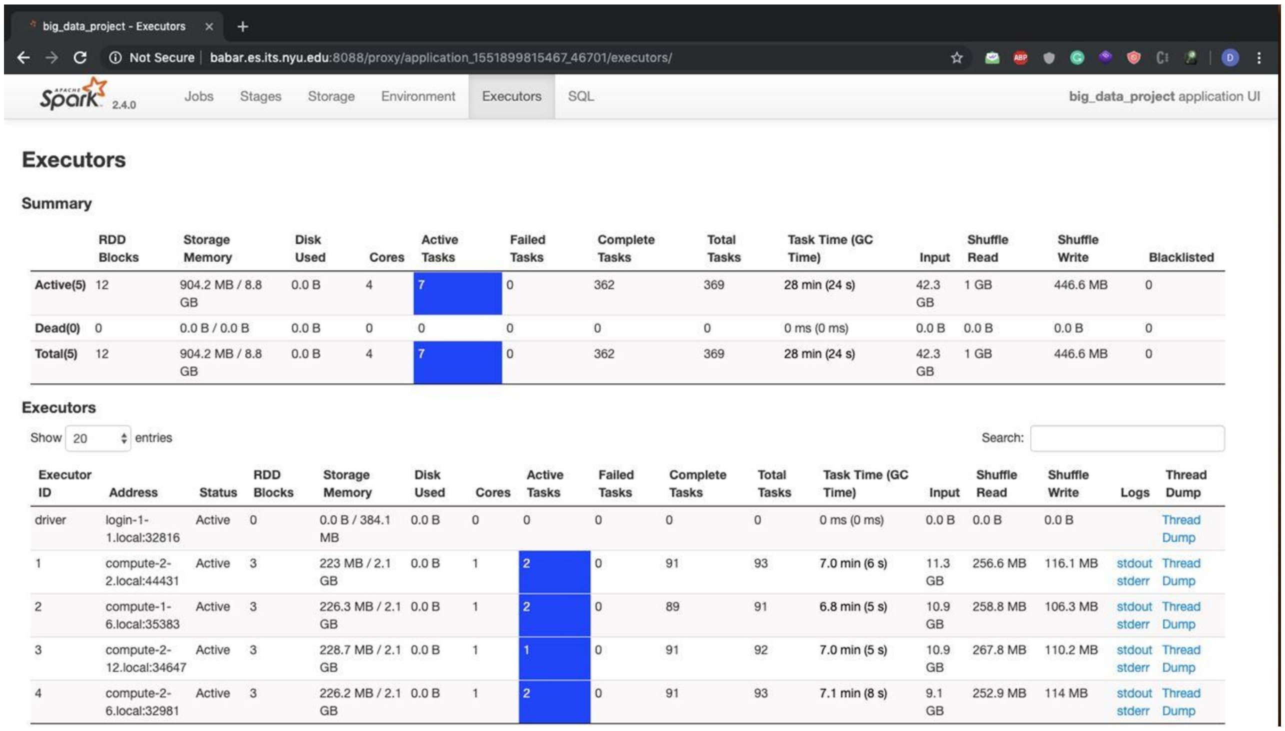
- **Spark Job Dashboard :- (from 14:10 to 14:20)**


RUNTIME : 10 MINUTES
big_data_project - Spark Jobs
Not Secure | babar.es.its.nyu.edu:8088/proxy/application_1551899815467_46701/jobs/
Spark 2.4.0
Jobs
Stages
Storage
Environment
Executors
SQL
big_data_project application UI

Spark Jobs (?)

User: djm746
Total Uptime: 8.7 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 35

- **Distributed Architecture:-**
  - 4 Executor Nodes are created, each with 3 RDD blocks.
  - The Driver node is the main node and therefore does not have any RDD’s allocated to it.


big_data_project - Executors
Not Secure | babar.es.its.nyu.edu:8088/proxy/application_1551899815467_46701/executors/
Spark 2.4.0
Jobs
Stages
Storage
Environment
Executors
SQL
big_data_project application UI

Executors

Summary

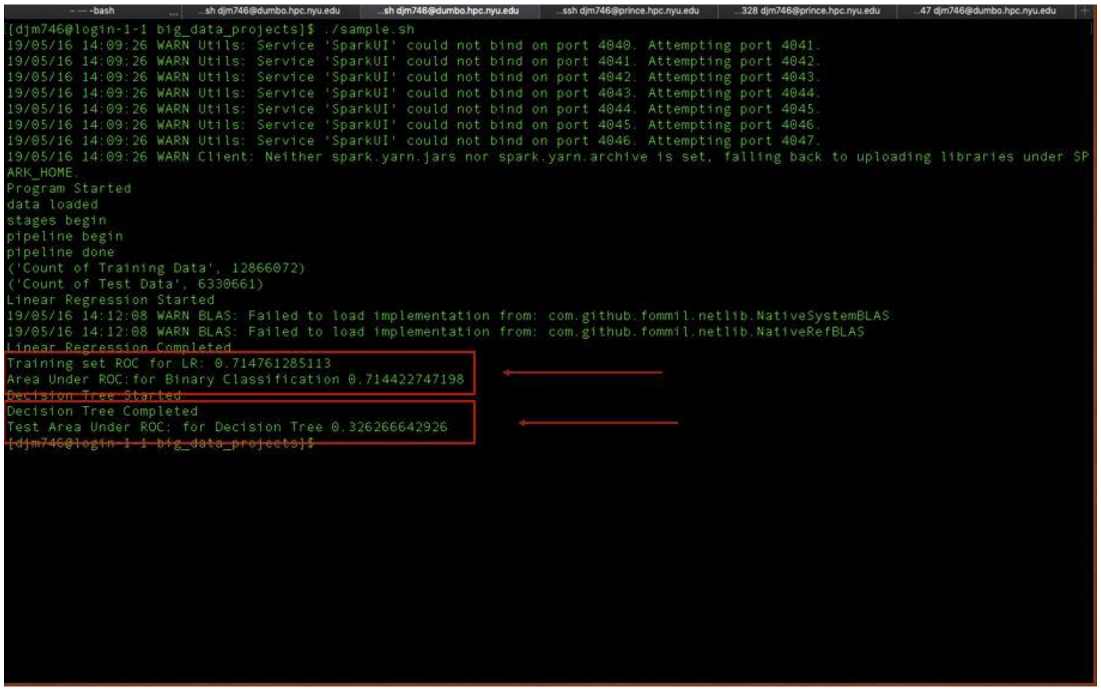
	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(5)	12	904.2 MB / 8.8 GB	0.0 B	4	7	0	362	369	28 min (24 s)	42.3 GB	1 GB	446.6 MB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(5)	12	904.2 MB / 8.8 GB	0.0 B	4	7	0	362	369	28 min (24 s)	42.3 GB	1 GB	446.6 MB	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	login-1-1.local:32816	Active	0	0.0 B / 384.1 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	compute-2-2.local:44431	Active	3	223 MB / 2.1 GB	0.0 B	1	2	0	91	93	7.0 min (6 s)	11.3 GB	256.6 MB	116.1 MB	stdout stderr	Thread Dump
2	compute-1-6.local:35383	Active	3	226.3 MB / 2.1 GB	0.0 B	1	2	0	89	91	6.8 min (5 s)	10.9 GB	258.8 MB	106.3 MB	stdout stderr	Thread Dump
3	compute-2-12.local:34647	Active	3	228.7 MB / 2.1 GB	0.0 B	1	1	0	91	92	7.0 min (5 s)	10.9 GB	267.8 MB	110.2 MB	stdout stderr	Thread Dump
4	compute-2-6.local:32981	Active	3	226.2 MB / 2.1 GB	0.0 B	1	2	0	91	93	7.1 min (8 s)	9.1 GB	252.9 MB	114 MB	stdout stderr	Thread Dump

- **Output Results:-**


-bash
...sh djm746@dumbo.hpc.nyu.edu
...sh djm746@dumbo.hpc.nyu.edu
...ssh djm746@prince.hpc.nyu.edu
...328 djm746@prince.hpc.nyu.edu
...47 djm746@dumbo.hpc.nyu.edu

```
[djm746@login-1-1 big_data_projects]$ ./sample.sh
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4042. Attempting port 4043.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4043. Attempting port 4044.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4044. Attempting port 4045.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4045. Attempting port 4046.
19/05/16 14:09:26 WARN Utils: Service 'SparkUI' could not bind on port 4046. Attempting port 4047.
19/05/16 14:09:26 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SP
ARK_HOME.
Program Started
data loaded
stages begin
pipeline begin
pipeline done
('Count of Training Data', 12866072)
('Count of Test Data', 6330661)
Linear Regression Started
19/05/16 14:12:08 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
19/05/16 14:12:08 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
Linear Regression Completed
Training set ROC for LR: 0.714761285113
Area Under ROC:for Binary Classification 0.714422747198
Decision Tree Started
Decision Tree Completed
Test Area Under ROC: for Decision Tree 0.326266642926
[djm746@login-1-1 big_data_projects]$
```

- **Output Results cont. :-**
  - The Accuracy of both the models have decreased, which is a testament to the fact that the Rainfall distribution over 20 million data records is much more sparse then on 0.75 million records.

7. Conclusion

In this project, we have determined the occurrence of Rainfall over the data accumulated from across the globe with an accuracy of 71%. The overall Accuracy came down as the data was spread across the globe and had uneven rainfall across it. The initial 86% for Logistic Regression and 84% for Decision Tree Classifier. But that accuracy was over a random data splice of 0.75 million records which had better Rainfall distribution. After getting sufficiently good results of our models we decided to scale this endeavor over the Cluster.

For prediction, we proposed a machine learning model based on a combination of Logistic regression and Decision Tree Classifier Model in order to compare each model's performance against each other. Moreover, we dealt with the overall process using Nyu Hpc Dumbo cluster as the platform and SparkML as the library for training the models.

We achieved Scalability and Runtime Efficiency in terms of Big Data in our project.

8. Future Scope

Based on available data, predict which conditions of weather would lead to rainfall, snowfall, tornado etc. which in technical terms can be known as Multi-class Classification.

Taking into account the feature vectors with precipitation, humidity, temperature increase, sea-level pressure etc. we can conform the suitable weather conditions for an above given class, which can have numerous socio-economic and forecasting applications.