Data Mining & Visualisation

CS551G

Assessment - 2

Yassin Dinana

52094480

5th April 2021

1. Description of Distributed Learning Big Data Ecosystem

The improvement and development of new technologies such as s Internet of Things, Social Network Applications, and Cloud Computing has emerged the need to store and analyse data. The big data ecosystem includes technologies that is used to store, analyse, and distribute huge amounts to data. There are different technologies available that can be applied to cope with the data provided which will be discussed in the following sections [1].

1.1- Distributed File Systems

Distributed file systems, better known as "DFS" is used to manage and store data, it works by distributing big amounts of data into different clusters, it can access the data across multiple clusters and undertake read and write operations parallelly, this will result in faster data analysis, less usage of computational power, and transparency [2].

Hadoop Distributed File System, or better known as HDFS is an example of a distributed file system in big data, it is used widely as it is highly fault-tolerant, and it can be deployed on low-cost hardware. The HDFS architecture follows the master-slave topology, where the master node is called *NameNode* and the slaves nodes are called *DataNodes* [3]. Figure 1 below shows the architecture of the HDFS, the content of the figure is used to understand the architecture.

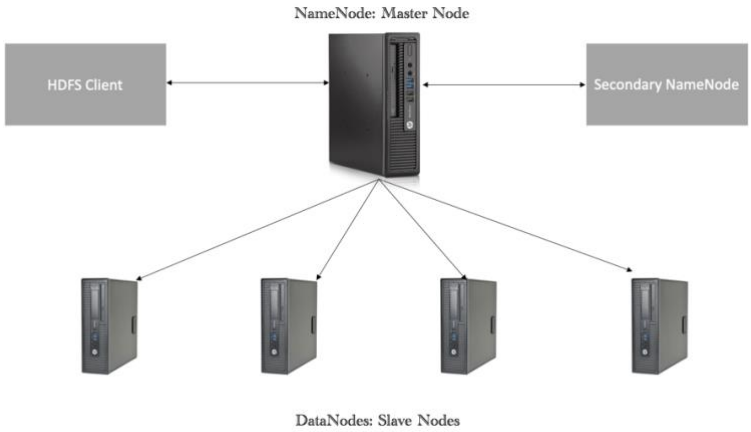


Figure 1: HDFS Architecture

NameNode (Master Node):	Responsible for managing all data nodes. Clients need to send a request to the NameNode when they need to read or write data. The NameNode then communicate with the DataNodes to process the request.
DataNodes (Slave Nodes):	Responsible for storing the actual data and serves the request given by clients.
HDFS Client	It is a software that is used to run commands for Hadoop services such as read and write services.
Secondary NameNode:	Is just is a helper for the main NameNode (Master Node, it works by creating random checkpoints on all the data that can be found in the Name Node before being sent to the Data Node in case there is a failure [4].

1.2- Resource Manager and Scheduler:

An important aspect in big data distribution is the *resource management and scheduling*, Hadoop introduced Yet Another Resource Navigator, better known as YARN, which is an open-source resource manager and application scheduler as part of the Hadoop distributed processing framework, it allows efficient allocation of system resources and scheduling tasks running in different clusters It ensures scalability, performance, and better resource allocation which is used widely in the IoT applications [5].

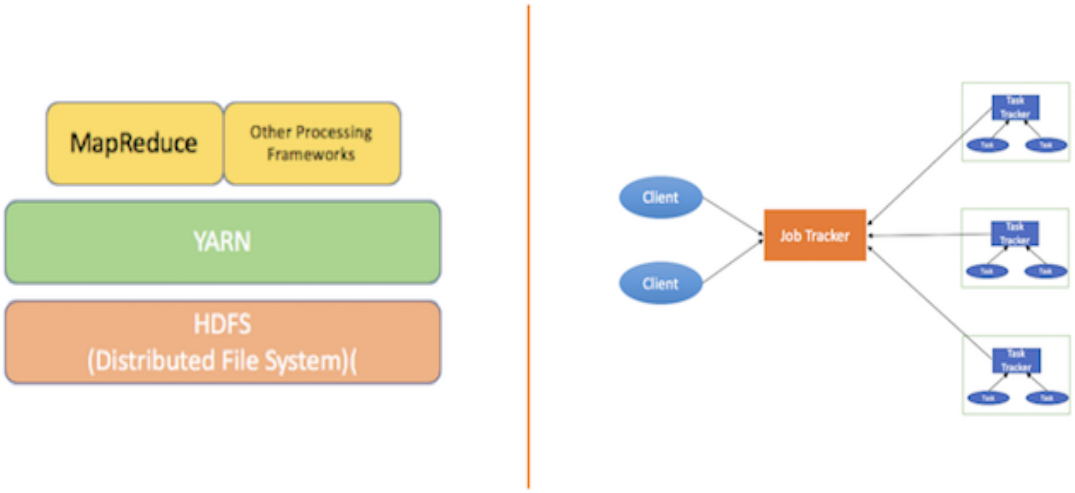


Figure 2: Hadoop General Architecture and YARN Architecture

Looking at figure 2 above, it can be seen that YARN separates the processing layer from the resource management. It can dynamically assign various resources and schedule application processing through its various components for huge amounts of data.

As seen in the YARN architecture in figure 2, there are many components of the architecture, the technicality of each component is discussed below [6]:

- *Client:* Submits the map-reduce jobs, which assigns parts of the data across all nodes in a Hadoop cluster.

- *Resource Manager:* Responsible for resource assignment and it manages all the applications in the process, it receives requests and communicates with node managers by sending it to them.

- *Scheduler*: It adjusts applications scheduling and allocations in the most efficient way.

- *Container*: The container is where all the YARN applications run, there is a different container available in each node.

- *Job Tracker:* Receives jobs from the clients and it communicates with all Name Nodes to conclude the location of the data.

1.3- Volume, Velocity, Variety, Value, and Veracity

The section below will be discussing what is known by "The Five V's of Big Data". Those are the volume, velocity, variety, value, and the veracity.

- *Volume*: The volume of the data mainly denotes to the size of the data that needs to be analysed. Companies and organizations use the term volume of data to measure the size of the data they collected which can be terabytes or petabytes of data [7].

- *Velocity*: The velocity of the data is the calculation of the speed at which the data is generated. When data has high velocity, it requires fast distribution techniques. Social media posts are an example of high velocity data [8].

- *Variety*: The variety of big data refers to the type of data that needs to be stored, it classifies the data collected into different categories such as structured and unstructured data.

- *Value*: The value of the data gathers all the information needed about the data and check how useful all the information can be in order to design a product line based on the given data. It extracts the important information and convert it into a valuable idea or product [9].

- *Veracity*: The veracity of the data concludes the uncertainty that can be found in the data as it can be messy which will affect the accuracy of the output. The data can consist of noise and randomness which will affect the value of the products [9].

Figure 3: The 5 V's of Big Data [10]

1.4- Fault Tolerance and Resilience

Part of the HDF System is the fault tolerance and resilience which is built in the system itself, it mainly ensures that the system is continuously working properly even after there are faults in some of its components. A fault in the case of HDFS could be the blockage of data from its node, such errors could be software bugs or hardware issues. Hadoop handles those issues by processing replica creations, when one node fails, multiple copies of that node will be available in another data node. This is done by using the *Replication Factor* by achieving data and creating multiple blocks into a data node, the processing happens on the replicated data allowing the operation to continue without failure [11].

1.5- Apache Spark

Apache Spark is a framework created for big data processing which is able to undertake processing on very large data sets. Unlike Hadoop, which is designed to handle batch processing as previously discussed, Spark is designed to deal with real-time data efficiently. The reason why Apache Spark usage is increasing widely:

- Simplicity
- Speed
- Support
- Ease of use
- Flexibility

It has the ability to store data in memory which helps increasing the processing and speed machine learning application, it is considered to be better than the Hadoop Map Reduce [12].

1.6- SparkML / PySpark

SparkML, better known as Spark Machine Learning Library, it is a library that provides machine learning algorithms such as classification, regression, random forest, SVM, and k-means [13].

PySpark is a python API (Application Programming Interface) that supports Apache Spark, it is mainly used for scalable analysis and ML pipelines, using Spark it is possible to run applications parallelly on multiple nodes – it runs across multiple nodes when on a cluster [14].

1.7- Spark SQL

SparkSQL is a module designed for structured data processing, it is used by providing Data Frames and it can also act as a distributed SQL query engine. When combining and processing machine learning and SQL query it provides a powerful ecosystem [15].

1.8- Docker Containers

It is an open-source software, it works by packaging different application in containers where they can run parallelly also allowing those application to be portable on different operating systems, it allows running scheduled jobs without setting applications in each node manually. After the containers are filled, you can run the distributed application such as Hadoop or Spark [16].

2. Develop distributed models in apache spark to classify gas turbines

This section includes the implementation, deployment, and testing of distributed models in Apache Spark in order to classify gas turbines. The programming language used for this task is Python and the code is executed on Google Collab which is an online cloud-based IDE.

An excel file containing all the information needed to classify gas turbines is used as the main dataset for this project where the classification output is a status affirming if the system is normal or abnormal.

2.1- Create Apache Spark Environment and Load Dataset

In order to create the Apache Spark environment on Google Colab, steps developed from "practical 4" must be implemented as well as installing few libraries must be installed. The dataset is read from Google Drive therefore it is also essential to connect the Google Colab to the drive [17].

Those steps include:
- Connecting the drive to Google Colab
- Install Java
- Unzipping and reading the data from the drive
- Setting up PySpark and installing Apache Spark in Colab
- Set environment path – This will enable PySpark to run on Google Colab

Figure 4: Code snippets for creating the Apache Spark environment

Figure 4 above shows the code snippets for creating the Apache Spark environment on Google Colab. After the cells above are running, it is now time to import the necessary libraries and load the dataset from the drive.

As stated earlier, the dataset provided is loaded and read from the drive, in order to perform such operation, we use the Pandas library to load the dataset and then we save the loaded data into a Spark Data Frame.

```
import pandas as pd
read_file = pd.read_excel ('/content/drive/MyDrive/Masters/Semester 2/Data Mining & Visualisation/Assessments/Assessment2\
                           /CS551G_DMV_Assessment_2_Dataset.xlsx')
data = spark.createDataFrame(read_file)
```

```
data.show(10)
data.count(), len(data.columns)
```

Figure 5: Importing the dataset using Pandas and then saving it into a Spark Data frame

In order to make sure that the dataset is imported correctly, it can be seen in figure 5 above that it is needed to print the first 10 rows of the dataset as well as the length of the dataset, where the output can be seen in the notebook. After the information is printed, it can be compared with the original excel file given in the drive to make sure there are no missing information that might affect the processing later on. The output of the code in figure 5 showsthat the first 10 rows of the dataset are printed, and it can also show that the total length of the dataset is 996 rows and 13 columns.

In order to better understand the dataset, the next step is to create a table including all the statistics of the dataset. The table below shows all the statistics of the dataset where the Status column is dropped. please refer to the notebook on Google Colab to visualize the table generated using Python. The Python lines below generate the data statisctics and drop the Status column.

```
data_temporary = data.drop(*['Status']) #We create a temporary dataframe and drop the (Status)
data_temporary.summary().show()
```

Figure 6: Code snippet for generating data statistics table

Summary	mean	stddev	min	25%	50%	75%	max
Temperature_Sensors_1	4.999	2.764	0.008	2.889	4.878	6.787	12.129
Temperature_Sensors_2	6.379	2.312	0.040	4.928	6.469	8.098	11.928
Temperature_Sensors_3	9.228	2.532	2.583	7.508	9.334	11.046	15.759
Flow_Rate_Sensor_1	7.355	4.354	0.062	3.425	7.063	10.910	17.235
Flow_Rate_Sensor_2	14.199	11.680	0.024	5.003	11.708	20.262	67.979
Flow_Rate_Sensor_3	3.077	2.126	0.008	1.414	2.671	4.502	10.242
Pressure_Sensor_1	5.749	2.526	0.001	4.005	5.739	7.498	12.647
Pressure_Sensor_2	4.997	4.165	0.005	1.572	3.858	7.599	16.555
Pressure_Sensor_3	8.164	6.173	0.0	3.184	6.748	11.253	36.186
Vibration_Sensor_1	10.001	7.336	0.018	3.997	8.779	14.679	34.867
Vibration_Sensor_2	15.187	12.159	0.064	5.502	12.167	21.832	53.238
Vibration_Sensor_3	9.933	7.282	0.009	3.821	8.843	14.355	43.231

Table 1: Dataset Statistics

The dataset includes 12 columns, where each column shows the values of a different sensor. There are 4 different types of sensors which are temperature sensor, flow rate sensor, pressure sensor, and vibration sensor. The column “Status” which is dropped includes two different entries which are Normal or Abnormal, it describes the status of a gas turbine with the given specifications. It can be concluded that there are no missing values in the dataset.

2.2- Visualize the data by creating a box plot and scatter plot

In order better understand the data, it is important to plot the data into plots, this part of the report will be focusing on creating a box plot and a scatter plot, different methods are applied in order to develop the plots and extract the parameters that need to be used in the plots from the dataset.

Starting off by the *box plot*, it is needed to include two classes on the X-Axis of the plot, those classes are the Normal and Abnormal, the Y-Axis must include the parameter "Vibration Sensor 1". HandySpark is used to plot the box plots, a handy spark data frame is created where the original data is saved. It is now important to extract the needed parameters from the new data frame and use the *matplot* library to plot the Box Plot [18].

The box plot that is generated in figure 7 is a good plot that presents the minimum, median, and maximum that were discussed earlier in table 1 but it can be seen that the minimum value of Vibration Sensor 1 in both classes in zero. It can also be concluded that the median is nearly the same, and the maximum is higher for the Abnormal class [19].

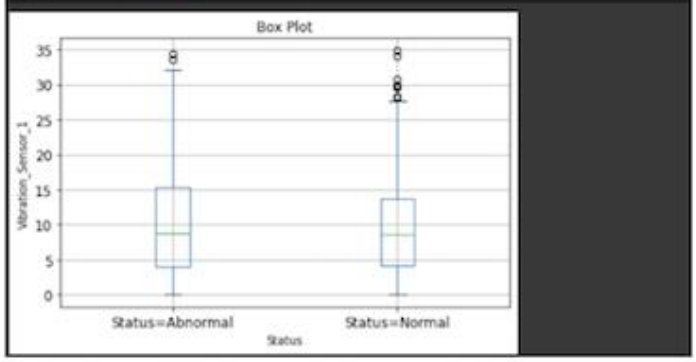
The next step is to plot the *Scatter Plot*, when trying to plot it using Handy Spark an error using the same technique as plotting the box plot, an error is received that not all values cannot be extracted, in order to generate the scatter plot, *seaborn* is used where the Y-Axis represents the Vibration Sensor 2 and the X- Axis is just the distance provided with Normal and Abnormal values, where Normal is presented using blue dots and Abnormal is presented using orange dots as see in figure 8 below [20].

Looking at figure 7 below, it can be seen that the Normal class (Blue Dots) has a higher value that surpasses 50 which is the peak in our graph.

BOX PLOT

```
#Plotting Boxplot
from matplotlib import pyplot as plt
from handyspark import *
df_hs = data.toHandy()

df_hs.stratify(['Status']).cols['Vibration_Sensor_1'].boxplot(figsize=(6, 4));
plt.xticks(fontsize = 12);
plt.xlabel('Status')
plt.yticks(fontsize = 12);
plt.ylabel('Vibration_Sensor_1')
plt.title("Box Plot");
```



SCATTER PLOT

```
Normal    = plot_data[plot_data.Status == "Normal"]
Abnormal = plot_data[plot_data.Status == "Abnormal"]
Normal.reset_index(inplace=True)
Abnormal.reset_index(inplace=True)

Normal = Normal.drop("Status", axis =1)
Abnormal = Abnormal.drop("Status", axis =1)

data = [Normal.Vibration_Sensor_2, Abnormal.Vibration_Sensor_2]

plt.figure(figsize=(16, 6))

ax2 = plt.subplot(122)
pt = sns.scatterplot(data=Normal.Vibration_Sensor_2, label="Normal")
pt = sns.scatterplot(data=Abnormal.Vibration_Sensor_2, label="Abnormal")
plt.xlabel("Distance")
plt.ylabel("Vibration_Sensor_2")
plt.show()
```

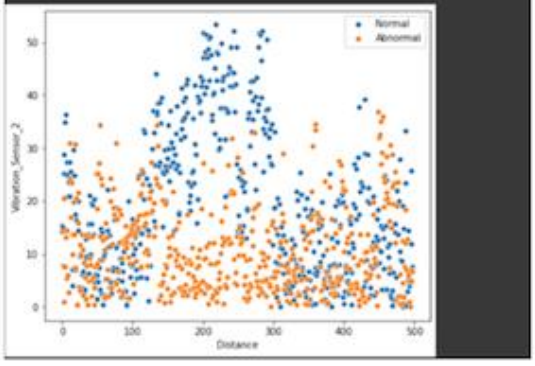


Figure 7: Box Plot and Scatter Plot generated for data visualization

2.3- Developing and Training a Random Forest Model

The *Random Forest* is a classification algorithm, it is consisted of many decision trees operating collaboratively, where each tree in the random forest classifier develops a prediction for a class, it is normal or abnormal in our case; finally, the class the most votes is considered the winner and becomes the main model's prediction [21].

As stated above, each tree votes for one of the labels, in order to ease the voting procedure and visualize it, it is known that there are two labels which are Normal and Abnormal, therefore, it is important to binarize the labels as part of the data pre-processing. This means that:

- Normal = 1
- Abnormal = 0

```
1 from pyspark.ml.feature import StringIndexer
2 labelIndexer = StringIndexer(inputCol="Status", outputCol="label").fit(data)
3 data = labelIndexer.transform(data)
4 data.select("Status","label").rdd.takeSample(False, 5, seed=0)
```

```
[Row(Status='Normal', label=1.0),
 Row(Status='Abnormal', label=0.0),
 Row(Status='Normal', label=1.0),
 Row(Status='Abnormal', label=0.0),
 Row(Status='Normal', label=1.0)]
```

Figure 8: Label Binarization code snippet with output

As part of the pre-processing before training the Random Forest model, it will make things easier to vectorize all features such as combining all of them into one vector by importing the VectorAssesmbler from the PySpark library [22].

```
1 #Features Vectorizatition
2 from pyspark.ml.feature import VectorAssembler
3 from pyspark.ml.linalg import Vectors
4 import py4j
5
6 data = data.drop(*['Status'])
7 inputcols = np.array(data.columns)[np.array(data.columns)!='label']
8 assembler = VectorAssembler(inputCols= inputcols, outputCol = "predictors")
9
10 predictors = assembler.transform(data)
11 model_data = predictors.select("predictors", "label")
12 model_data.show(5,truncate=False)
```

```
+-----------------------------------------------------------------------------------------+-----+
|predictors                                                                               |label|
+-----------------------------------------------------------------------------------------+-----+
|[4.5044,0.7443,6.34,1.9052,29.5315,0.8647,2.2044,6.048,14.4659,21.648,15.3429,1.2186]    |1.0  |
|[4.4284,0.9073,5.6433,1.6232,27.5032,1.4704,1.9929,5.9856,20.8356,0.0646,14.8813,7.3483]|1.0  |
|[4.5291,1.0199,6.113,1.0565,26.4271,1.9247,1.942,6.7162,5.3358,11.0779,25.0914,9.2408]   |1.0  |
|[5.1727,1.0007,7.8589,0.2765,25.1576,2.609,2.9234,6.7485,1.9017,1.8463,28.664,4.0157]    |1.0  |
|[5.2258,0.6125,7.9504,0.1547,24.0765,3.2113,4.4563,5.8411,0.5077,9.37,34.8122,13.4966]   |1.0  |
+-----------------------------------------------------------------------------------------+-----+
only showing top 5 rows
```

Figure 9: Features Vectorization

Looking at figure 9 above, it can be seen that all the features are combined into one vector that is named "Predictors" whereby also looking at the output the label is also displayed as stated before normal is 1 and abnormal is 0 [22].

After applying some pre-processing techniques, it is now time to split and train the data on the Random Forest Classifier Model. We first import the *RandomForestClassifier* from the PySpark library, then It is asked to split the dataset intro 70% training and 30% testing, this is done using the *randomSplit* function. The training data is used only for training, where the evaluation, testing, and prediction of the system will use the testing data [23].

```
1 from pyspark.ml.classification import RandomForestClassifier
2 train_data, test_data = model_data.randomSplit([0.70,0.30], seed=1)
3
4 random_forest = RandomForestClassifier(featuresCol = 'predictors', labelCol=  'label')
5 random_forest_Model = random_forest.fit(train_data)
6
7 rf_predictions = random_forest_Model.transform(test_data)
8 rf_predictions.show(10)
```

```
+--------------------+-----+--------------------+--------------------+----------+
|          predictors|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[0.1447,5.5261,7....|  1.0|[5.04072509180695...|[0.25203625459034...|       1.0|
|[0.4724,4.9612,8....|  1.0|[12.6189991364085...|[0.63094995682042...|       0.0|
|[0.8276,11.4771,1...|  1.0|[6.89639267692698...|[0.34481963384634...|       1.0|
|[0.856,4.6408,6.5...|  1.0|[11.8661259717903...|[0.59330629858951...|       0.0|
|[1.1609,4.8175,7....|  1.0|[5.37887289235788...|[0.26894364461789...|       1.0|
|[1.3065,5.3293,7....|  1.0|[10.2980732570422...|[0.51490366285211...|       0.0|
|[1.4502,4.3204,7....|  1.0|[12.6059759415584...|[0.63029879707792...|       0.0|
|[1.5271,5.0052,7....|  1.0|[10.1677064302064...|[0.50838532151032...|       0.0|
|[1.6333,9.2532,10...|  1.0|[13.3820027525698...|[0.66910013762849...|       0.0|
|[1.6681,5.8282,6....|  1.0|[5.64639619598969...|[0.28231980979948...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 10 rows
```

Figure 10: Splitting and training the data

Looking at figure 10 above, it can be seen that the data is split to 70% training and 30% testing as stated, it can also be seen that the predictors vector that can be seen in figure 9 is used where is does its predictions using the labels that are introduced in figure 8.

After passing the parameters "Predictors" and "Label" to the Random forest Classifier, the training data is then used in the training process, represented in figure 10 as *.fit(train_data).* Finally, we use the transform method after training the model where it provides all the predictions but as seen using the testing data. In order to visualize the output, the top 10 rows of the Random Forest Model predictions are printed as an output.

2.4- Model Performance and Evaluation

After splitting the data and training the model which can be seen in figure 10, it is now time to evaluate the performance of the model, to report the performance, only the testing data is being used. As the random forest classifier has been introduced earlier in this paper, it follows the majority voting technique – the predictors votes can be divided into four different categories such as [23]:

- True Positive: Positive prediction is TRUE
- True Negative: Negative prediction is TRUE
- False Positive: Positive prediction is FALSE
- False Negative: Negative prediction is FALSE

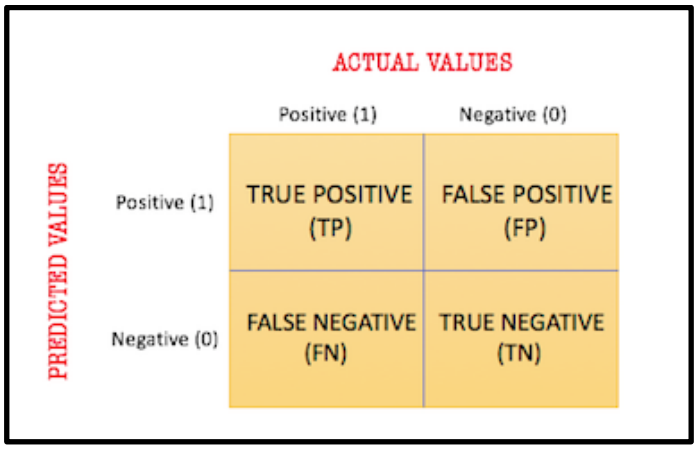


Figure 11: Confusion matrix showing voting options

In order to report the module's performance, there are three different metrics to be applied in this paper, those metrics are, *Accuracy*, and *Area Under the Curve (AUROC).*

- *Precision / Recall*
- *Accuracy*
- *Area Under the Curve (AUROC).*

Each of the metrics above can evaluate the model mathematically using specific equations which can also be applied as an output in Python [24].

- **Precision / Recall:**

The *precision* mainly refers to the percentage of the relevant results, it calculates the number of correct results (votes in our case). It uses the ratio of true positives and all positive predictions.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (1)$$

The *recall* mainly refers to the measure of the model correctly identifying true positives, this is calculated by defining the ratio of all true positives and actual positives.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (2)$$

- **Accuracy:**

The *accuracy* mainly refers to the ratio of the total correct predictions and total number of predictions – it is the most used model evaluation metric. It is described as [25]:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative + False\ Negative} \quad (3)$$

- **Area Under the Curve (AUROC):**

The *Receiver Operator Characteristic,* better known as ROC, is a graph that shows different confusion matrices produced at different thresholds, it shows the relationship between the true positive and false positive rate [24].

The Area Under the Curve, better known as AUC, helps compare different ROC graphs to each other in order to show which one is greater than the other, this curve is generated using Python in the code and will be shown later on in this report.

- **Results:**

This section will be covering the code implementation of the evaluation metrics discussed above as well as the output of the model.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

ran_forest = random_forest_pred.select("prediction","label")
true_positive = ran_forest[(ran_forest.prediction == 1) & (ran_forest.label == 1)].count()
true_negative = ran_forest[(ran_forest.prediction == 0) & (ran_forest.label == 0)].count()
false_positive = ran_forest[(ran_forest.prediction == 0) & (ran_forest.label == 1)].count()
false_negative = ran_forest[(ran_forest.prediction == 1) & (ran_forest.label == 0)].count()

confusion_matrix = np.array([[tp, fp], [fn, tn]])
col_name = ["Real Abnormal", 'Real Normal']
row_name = ['Predicted Abnormal', 'Predicted Normal']
df_confusion_matrix = pd.DataFrame(confusion_matrix, index = [i for i in col_name],
                     columns = [i for i in row_name])

evaluator = BinaryClassificationEvaluator()
auroc = evaluator.evaluate(random_forest_pred)

print('Accuracy: ', (true_positive + tn) / (true_positive + true_negative + false_positive + false_negative))
print('Precision: ', true_positive / (true_positive + false_positive))
print('Recall: ', true_positive / (true_positive + false_negative))
print('Test Area Under ROC: ', auroc)
print('\n\nConfusion Matrix:\n', df_confusion_matrix)
```

```
Accuracy:  0.8426573426573427
Precision:  0.8085106382978723
Recall:  0.8636363636363636
Test Area Under ROC:  0.9279041330398623


Confusion Matrix:
                 Predicted Abnormal  Predicted Normal
Real Abnormal                 114                27
Real Normal                    18               127
```

Figure 12: Model evaluation code implementation

Looking at figure 12 above, it can be seen that the *BinaryClassificationEvaluator* is used from the PySpark library, it can also be seen that all the predictions and labels for all cases are defined, the confusion matrix is also defined, it is the same confusion that can be seen in figure 11.

The previous section which included all the equations for the different evaluation metrics, those equation are implemented in the print functions that can be seen in figure 12 above. Looking at the output of the model, it can be seen that the model recorded a high accuracy of **84%,** a precision of **80%**, the recall is **86%** which is needed to be high.

The next part of this paper will be discussing the printing of the ROC and a Precision / Recall curve as well as discussing the results.

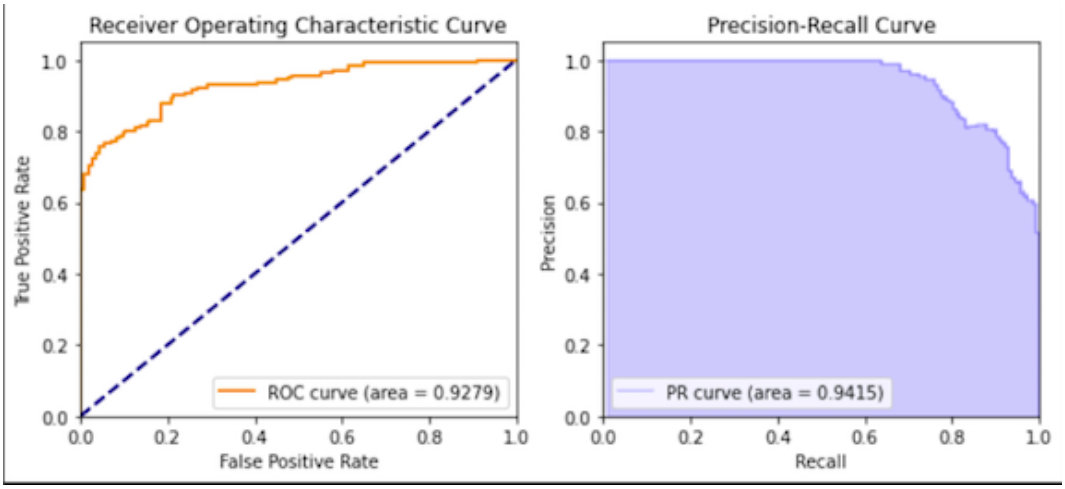


Figure 13: ROC and Precision – Recall Curve

Looking at the curves generated in figure 13, it can be seen that there is an ROC curve which is generated using the BinaryClassificationMetrics library from HandySpark. It can be seen that the orange curve represents the ROC curve in the graph as well as the True Positive Rate, it is calculated in figure 12 that the area is 92.7% where the threshold is 1. An excellent model has area under the curve near to the 1 meaning that it has a good measure of separability [26].

The precision-recall curve shows that it starts at the threshold of 1 and is stable for a while and the recall starts decreasing slowly – Starting off and remaining at the threshold shows good performance of the mode, it can also be seen that the area reached a good score of 94%.

2.5- Implementation of the Multi-Layer Perceptron Classifier

The Multi-Layer Perceptron Classifier is a classifier that connects to a feed-forward neural network, it mainly relies of the neural network itself to undertake the classification process. The MLP networks has many different layers that are all connected to each other.

By implementing the MLP classifier from the PySpark library, as before, we train the model using the training data and then evaluate it using the testing data, and finally visualize and analyze the results.

```
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession

if __name__ == "__main__":
    spark = SparkSession\
        .builder.appName("multilayer_perceptron_classification_example").getOrCreate()

inputs = np.array(data.columns)[np.array(data.columns)!= 'label']
print(len(inputs))

layers = [len(inputs), 5, 4, 4, 3 , 2]
# create the trainer and set its parameters
mlp_trainer = MultilayerPerceptronClassifier(featuresCol = 'predictors', labelCol=  'label', maxIter=100, layers=layers, blockSize=128, seed=1)

# train the model
model = mlp_trainer.fit(train_data)

# compute accuracy on the test set
result = model.transform(test_data)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

Figure 14: Implementation of the MLP classifier

Looking at figure 14 above, it shows the implementation of the MLP Classifier, it firsts starts by created a spark environment so that the MLP Classifier is able to run on Google Colab – We then need to specify the inputs for the neural network layers, which takes the labels are input, the output shows that there are 12 inputs which is printed as seen in line 10 in figure 14 above.

The MLP classifier then takes the predictors and the labels as parameters, this is same as the Random Forest Classifier, the difference here is because the neural networks are used for classification, it can be seen that the number of iterations, number of layers, and block size are also parameters which are passed to the neural network. The next step is fitting the training data in the model, this can be seen in line 17 above, similarly as before, we use the transform() function to transform the data and using the testing set for evaluation.

```
# Evaluating on Test Data
true_positive = predictionAndLabels[(predictionAndLabels.prediction == 1) & (predictionAndLabels.label == 1)].count()
true_negative = predictionAndLabels[(predictionAndLabels.prediction == 0) & (predictionAndLabels.label == 0)].count()
false_positive = predictionAndLabels[(predictionAndLabels.prediction == 0) & (predictionAndLabels.label == 1)].count()
false_negative = predictionAndLabels[(predictionAndLabels.prediction == 1) & (predictionAndLabels.label == 0)].count()

print('Accuracy: ', (true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative))
```

Figure 15: Evaluating the model on testing data

Looking at figure 15 above, the evaluation using the different metrics is used on the testing set. The data is also split as 70% for training data and 30% for testing data. Unfortunately, it is only showing an accuracy of 69% as a starting scale (The model is not yet trained on many epochs).

The next step is to train the model on different epochs, an empty list is created where it will include all the accuracies developed after every iteration, it is decided that the model will run on 100 iterations and all accuracies will be printed as an output.

```
Iteration: 1   Accuracy: 49.3006993006993
Iteration: 2   Accuracy: 50.699300699301
Iteration: 3   Accuracy: 50.699300699301
Iteration: 4   Accuracy: 50.699300699301
Iteration: 5   Accuracy: 50.699300699301
Iteration: 6   Accuracy: 50.699300699301
Iteration: 7   Accuracy: 50.699300699301
Iteration: 8   Accuracy: 50.699300699301
Iteration: 9   Accuracy: 50.699300699301
Iteration: 10   Accuracy: 50.699300699301
Iteration: 11   Accuracy: 50.699300699301
Iteration: 12   Accuracy: 50.699300699301
Iteration: 13   Accuracy: 50.699300699301
Iteration: 14   Accuracy: 50.699300699301
Iteration: 15   Accuracy: 50.699300699301
Iteration: 16   Accuracy: 50.699300699301
Iteration: 17   Accuracy: 50.699300699301
Iteration: 18   Accuracy: 57.342657342657345
Iteration: 19   Accuracy: 58.041958041958044
Iteration: 20   Accuracy: 58.74125874125874
Iteration: 21   Accuracy: 62.58741258741259
Iteration: 22   Accuracy: 59.44055944055944
Iteration: 23   Accuracy: 59.090909090909094
Iteration: 24   Accuracy: 59.090909090909094
Iteration: 25   Accuracy: 59.44055944055944
Iteration: 26   Accuracy: 59.44055944055944
Iteration: 27   Accuracy: 60.13986013986013
Iteration: 28   Accuracy: 60.13986013986013
Iteration: 29   Accuracy: 60.13986013986013
Iteration: 30   Accuracy: 59.79020979020979
Iteration: 31   Accuracy: 60.13986013986013
Iteration: 32   Accuracy: 60.13986013986013
Iteration: 33   Accuracy: 60.83916083916085
Iteration: 34   Accuracy: 60.48951048951049
Iteration: 35   Accuracy: 60.48951048951049
Iteration: 36   Accuracy: 60.83916083916085
Iteration: 37   Accuracy: 62.23776223776224
Iteration: 38   Accuracy: 61.88811188811189
Iteration: 39   Accuracy: 62.93706293706294
Iteration: 40   Accuracy: 61.88811188811189
Iteration: 41   Accuracy: 61.53846153846154
Iteration: 42   Accuracy: 60.83916083916085
Iteration: 43   Accuracy: 61.53846153846154
Iteration: 44   Accuracy: 62.23776223776224
Iteration: 45   Accuracy: 61.88811188811189
Iteration: 46   Accuracy: 63.63636363636363
Iteration: 47   Accuracy: 63.98601398601399
Iteration: 48   Accuracy: 63.63636363636363
Iteration: 52   Accuracy: 63.286713286713294
Iteration: 53   Accuracy: 62.93706293706294
Iteration: 54   Accuracy: 63.286713286713294
Iteration: 55   Accuracy: 63.286713286713294
Iteration: 56   Accuracy: 63.98601398601399
Iteration: 57   Accuracy: 63.63636363636363
Iteration: 58   Accuracy: 62.93706293706294
Iteration: 59   Accuracy: 62.93706293706294
Iteration: 60   Accuracy: 63.286713286713294
Iteration: 61   Accuracy: 62.93706293706294
Iteration: 62   Accuracy: 62.93706293706294
Iteration: 63   Accuracy: 63.63636363636363
Iteration: 64   Accuracy: 63.98601398601399
Iteration: 65   Accuracy: 63.98601398601399
Iteration: 66   Accuracy: 64.33566433566433
Iteration: 67   Accuracy: 64.33566433566433
Iteration: 68   Accuracy: 64.33566433566433
Iteration: 69   Accuracy: 64.33566433566433
Iteration: 70   Accuracy: 64.33566433566433
Iteration: 71   Accuracy: 63.98601398601399
Iteration: 72   Accuracy: 64.33566433566433
Iteration: 73   Accuracy: 64.33566433566433
Iteration: 74   Accuracy: 64.6853146853147
Iteration: 75   Accuracy: 64.6853146853147
Iteration: 76   Accuracy: 64.33566433566433
Iteration: 77   Accuracy: 64.33566433566433
Iteration: 78   Accuracy: 64.33566433566433
Iteration: 79   Accuracy: 64.33566433566433
Iteration: 80   Accuracy: 63.98601398601399
Iteration: 81   Accuracy: 63.98601398601399
Iteration: 82   Accuracy: 63.63636363636363
Iteration: 83   Accuracy: 63.63636363636363
Iteration: 84   Accuracy: 64.6853146853147
Iteration: 85   Accuracy: 65.03496503496503
Iteration: 86   Accuracy: 65.38461538461539
Iteration: 87   Accuracy: 65.03496503496503
Iteration: 88   Accuracy: 65.38461538461539
Iteration: 89   Accuracy: 65.38461538461539
Iteration: 90   Accuracy: 65.03496503496503
Iteration: 91   Accuracy: 65.03496503496503
Iteration: 92   Accuracy: 65.03496503496503
Iteration: 93   Accuracy: 65.03496503496503
Iteration: 94   Accuracy: 65.38461538461539
Iteration: 95   Accuracy: 65.03496503496503
Iteration: 96   Accuracy: 65.38461538461539
Iteration: 97   Accuracy: 65.03496503496503
Iteration: 98   Accuracy: 65.38461538461539
Iteration: 99   Accuracy: 65.38461538461539
```

Figure 16: Accuracy results after 100 iterations

Looking at figure 16 above, the accuracy of the model after evaluating it on the test data can be seen, unfortunately the model started with a very low accuracy of 49% but the model is learning slowly, after 100 iterations it reached an accuracy for 65.3%. It has been tried to increase the number of iterations but unfortunately it stops at 65.3% and starts decreasing.

In order to visualize the learning progress of the model, it can be seen in the graph below after tuning the model.

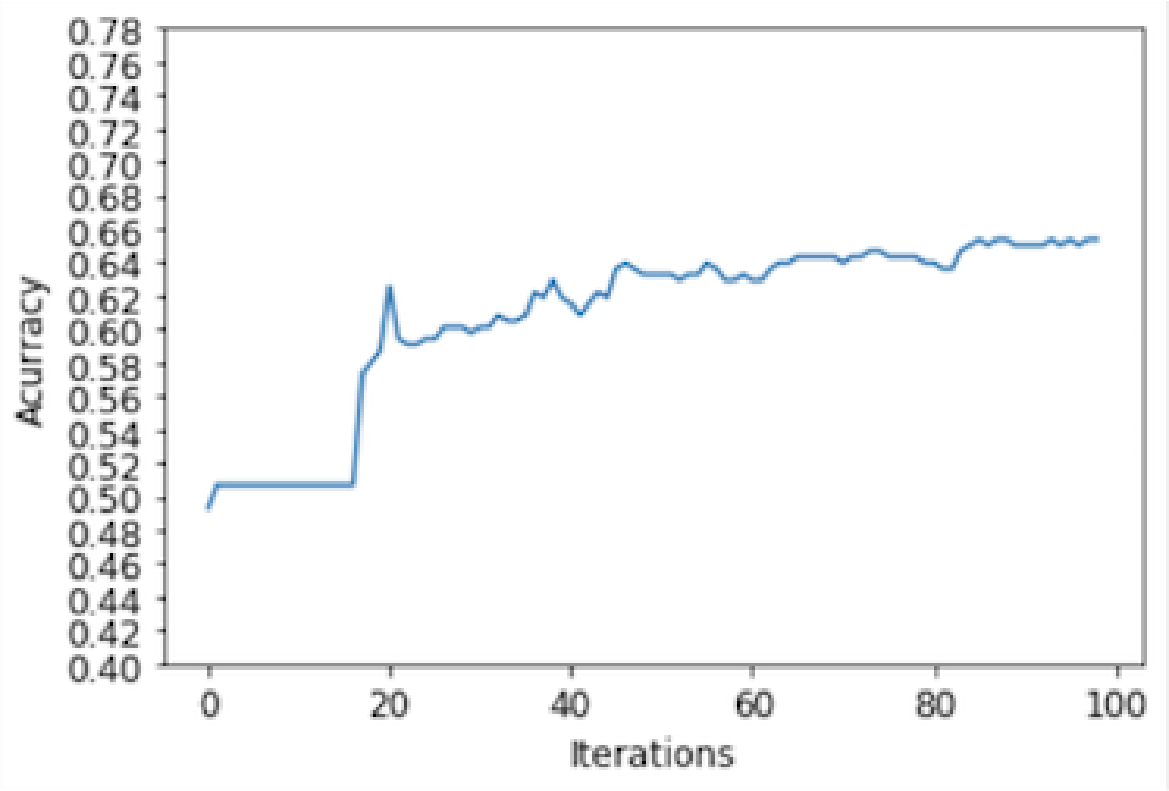


Figure 17: Model Evaluation Progress Graph

The graph in figure 17 above shows the progress of the accuracy improvement that can be seen in figure 16 of this report.

Similarly, like we did in the Random Forest Classifier, it is important to visualize the evaluation using the metrics discussed earlier in the report using the same technique.

```
true_positive = predictionAndLabels[(predictionAndLabels.prediction == 1) & (predictionAndLabels.label == 1)].count()
true_negative = predictionAndLabels[(predictionAndLabels.prediction == 0) & (predictionAndLabels.label == 0)].count()
false_positive = predictionAndLabels[(predictionAndLabels.prediction == 0) & (predictionAndLabels.label == 1)].count()
false_negative = predictionAndLabels[(predictionAndLabels.prediction == 1) & (predictionAndLabels.label == 0)].count()

confusion_matrix = np.array([[tp, fp], [fn, tn]])
col_name = ["Actual Abnormal", 'Actual Normal']
row_name = ['Predicted Abnormal', 'Predicted Normal']
df_confusion_matrix = pd.DataFrame(cm, index = [i for i in col_name],
                      columns = [i for i in row_name])

auroc = evaluator.evaluate(result)

print('Accuracy: ', (true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative))
print('Precision: ', true_positive / (true_positive + false_positive))
print('Recall: ', true_positive / (true_positive + false_negative))
print('Test Area Under ROC: ', auroc)
print('\n\nConfusion Matrix:\n', df_confusion_matrix)
```

```
Accuracy:  0.6538461538461539
Precision:  0.6099290780141844
Recall:  0.6615384615384615
Test Area Under ROC:  0.6538461538461539


Confusion Matrix:
                 Predicted Abnormal  Predicted Normal
Actual Abnormal                 114                27
Actual Normal                    18               127
```

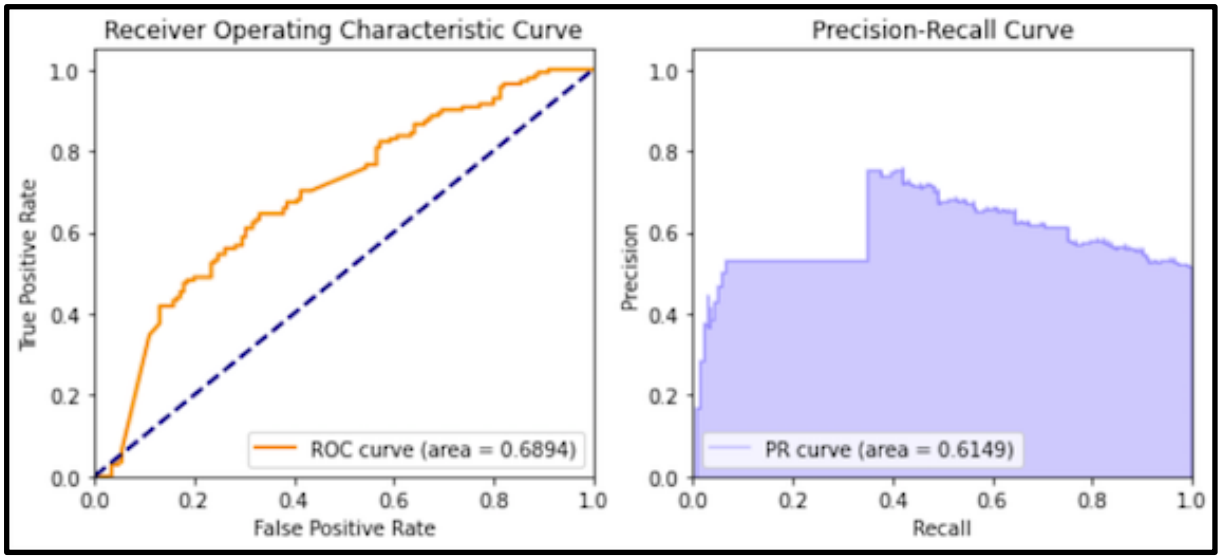


Figure 18: Model Evaluation using different metrics

Looking at figures above, it can be seen in the code output that the accuracy of the system is at 65.3%, precision is at 60%, and the recall is at 66%. Comparing the MLP Classifier to the Random Forest Classifier, it is very obvious that it is performing worse in terms of accuracy and precision.

Looking at the graphs generated, the ROC graph area is only at 68.9% which is very poor, different methods were applied in order to increase the performance such as increasing number of iterations, increasing training data, adding / removing layers, unfortunately those were the best results achieved.

Looking at the precision-recall graph, it can be seen that the performance is not hitting its peak as it did with the random forest classifier, its peak is around 75% of precision. Making the Random Forest Classifier better for this classification problem.

REFERENCES A2:

1- http://cis.csuohio.edu/~sschung/CIS601/CIS601FinalProposalExampleSuhua.pdf
2- https://www.ijser.org/researchpaper/BIG-DATA-IMPORTANCE-OF-HADOOP-DISTRIBUTED-FILESYSTEM.pdf
3- https://www.ijser.org/researchpaper/BIG-DATA-IMPORTANCE-OF-HADOOP-DISTRIBUTED-FILESYSTEM.pdf
4- https://www.ijert.org/research/daemons-of-hadoop-an-overview-IJERTV7IS040416.pdf
5- https://www.comp.nus.edu.sg/~hebs/pub/shangresource16.pdf
6- https://www.geeksforgeeks.org/hadoop-yarn-architecture/
7- https://www.bbva.com/en/five-vs-big-data/
8- https://www.bigdataframework.org/four-vs-of-big-data/#:~:text=Velocity%20of%20Big%20Data,Twitter%20messages%20or%20Facebook%20posts.
9- https://www.geeksforgeeks.org/5-vs-of-big-data/
10- https://www.semanticscholar.org/paper/Real-time-processing-technologies-in-big-data%3A-Ibtissame-Yassine/19545841f6810cde52b0ab5bbcbe1a8a474ba7fe/figure/0
11- https://core.ac.uk/download/pdf/77239584.pdf
12- https://www.bigdata-toronto.com/2016/assets/getting_started_with_apache_spark.pdf
13- http://spark.apache.org/docs/latest/ml-guide.html
14- https://runawayhorse001.github.io/LearningApacheSpark/pyspark.pdf
15- https://spark.apache.org/docs/latest/sql-programming-guide.html
16- https://core.ac.uk/download/pdf/334410306.pdf
17- https://www.analyticsvidhya.com/blog/2020/11/a-must-read-guide-on-how-to-work-with-pyspark-on-google-colab-for-data-scientists/
18- https://towardsdatascience.com/handyspark-bringing-pandas-like-capabilities-to-spark-dataframes-5f1bcea9039e
19- https://chartio.com/resources/tutorials/what-is-a-box-plot/
20- https://s3.amazonaws.com/assets.datacamp.com/production/course_6919/slides/chapter1.pdf
21- https://www.nature.com/articles/s41598-018-34833-6
22- https://spark.apache.org/docs/latest/ml-features
23- https://towardsdatascience.com/metrics-for-evaluating-machine-learning-classification-models-python-example-59b905e079a5
24- https://towardsdatascience.com/metrics-for-evaluating-machine-learning-classification-models-python-example-59b905e079a5
25- https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/
26- https://www.indianpediatrics.net/apr2011/277.pdf