BigData and Large Scale Computing

Topic : Sleep Quality data Analysis using Spark

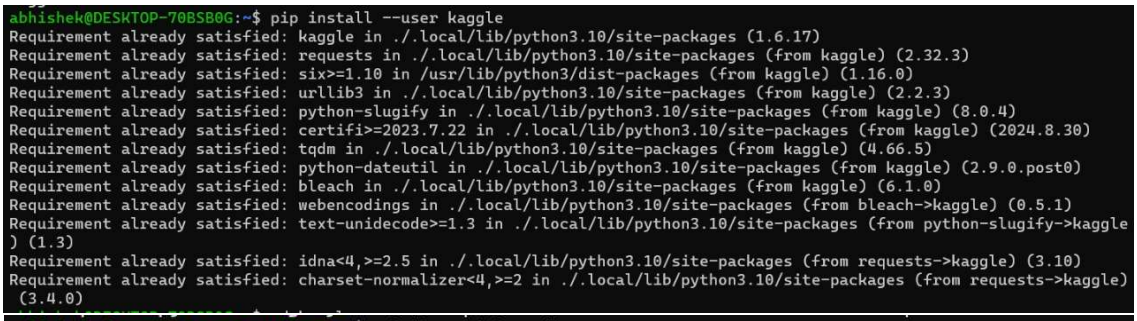
Name : Abhishek Harsh

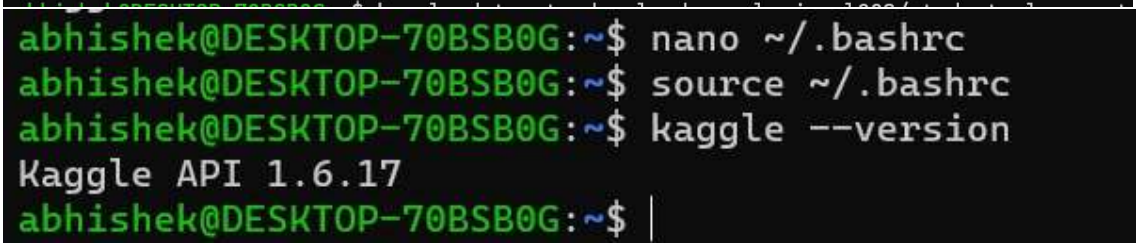
Roll : 2021BCS036

Task 1: Select the dataset under the classification category.

Student_sleep_patterns.csv dataset for the classification in the adequate and inadequate sleep for the student of the data set consisting 500 datas in the following format:

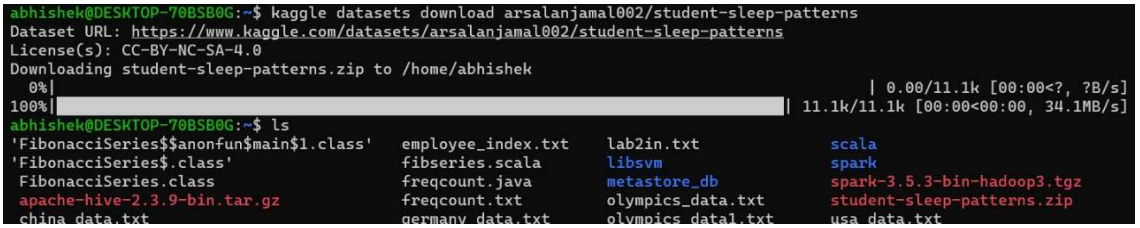
```
scala> df.show(5) ;
+----------+---+------+---------------+--------------+-----------+-----------+----------------+---------------+-------------+-------------------+-------------------+-----------------+-----------------+
|Student_ID|Age|Gender|University_Year|Sleep_Duration|Study_Hours|Screen_Time|Caffeine_Intake|Physical_Activity|Sleep_Quality|Weekday_Sleep_Start|Weekend_Sleep_Start|Weekday_Sleep_End|Weekend_Sleep_End|
+----------+---+------+---------------+--------------+-----------+-----------+----------------+---------------+-------------+-------------------+-------------------+-----------------+-----------------+
|         1| 24| Other|       2nd Year|           7.7|        7.9|        3.4|               2|             37|           10|              14.16|               4.85|             7.41|             7.06|
|         2| 21|  Male|       1st Year|           6.3|        6.0|        1.9|               5|             74|            2|               8.73|                7.1|             8.21|            10.21|
|         3| 22|  Male|       4th Year|           5.1|        6.7|        3.9|               5|             53|            5|               20.0|              20.47|             6.88|            10.92|
|         4| 24| Other|       4th Year|           6.3|        8.6|        2.8|               4|             55|            9|              19.82|               4.08|             6.69|             9.42|
|         5| 20|  Male|       4th Year|           4.7|        2.7|        2.7|               0|             85|            3|              20.98|               6.12|             8.98|             9.01|
+----------+---+------+---------------+--------------+-----------+-----------+----------------+---------------+-------------+-------------------+-------------------+-----------------+-----------------+
```





Add ~/.local/bin to your PATH so the terminal can recognize the kaggle command:
export PATH=$PATH:~/.local/bin

Downloading the DataSet using kaggle:



Task 2: Data Pre-processing and Data Visualization using SPARK

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().appName("ChildSleepQualityRegression").config("spark.master", "local[*]").getOrCreate()
import org.apache.spark.sql.functions._
val df = spark.read.option("header", "true").option("inferSchema", "true").csv("student_sleep_patterns.csv");

df.show(5) ;
```

```
scala> df.show(5) ;
+----------+---+------+---------------+--------------+-----------+-----------+---------------+-----------------+------------+
|Student_ID|Age|Gender|University_Year|Sleep_Duration|Study_Hours|Screen_Time|Caffeine_Intake|Physical_Activity|Sleep_Quality|
+----------+---+------+---------------+--------------+-----------+-----------+---------------+-----------------+------------+
|         1| 24| Other|       2nd Year|           7.7|        7.9|        3.4|              2|               37|          10|
|         2| 21|  Male|       1st Year|           6.3|        6.0|        1.9|              5|               74|           2|
|         3| 22|  Male|       4th Year|           5.1|        6.7|        3.9|              5|               53|           5|
|         4| 24| Other|       4th Year|           6.3|        8.6|        2.8|              4|               55|           9|
|         5| 20|  Male|       4th Year|           4.7|        2.7|        2.7|              0|               85|           3|
+----------+---+------+---------------+--------------+-----------+-----------+---------------+-----------------+------------+
```

```
+-----------------+------------+-----------------+-----------------+---------------+---------------+
|Physical_Activity|Sleep_Quality|Weekday_Sleep_Start|Weekend_Sleep_Start|Weekday_Sleep_End|Weekend_Sleep_End|
+-----------------+------------+-----------------+-----------------+---------------+---------------+
|               37|          10|            14.16|             4.05|           7.41|           7.06|
|               74|           2|             8.73|              7.1|           8.21|          10.21|
|               53|           5|             20.0|            20.47|           6.88|          10.92|
|               55|           9|            19.82|             4.08|           6.69|           9.42|
|               85|           3|            20.98|             6.12|           8.98|           9.01|
+-----------------+------------+-----------------+-----------------+---------------+---------------+
```

1) Show the number of samples (row) and number of features (column)

```
val num_columns=df.columns.length;
val num_rows = df.count();
```

```
scala> val num_columns=df.columns.length;
num_columns: Int = 14

scala> val num_rows = df.count();
num_rows: Long = 500
```

2) Find the class label

```
df.printSchema();
```

```
scala> df.printSchema()
root
 |-- Student_ID: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- University_Year: string (nullable = true)
 |-- Sleep_Duration: double (nullable = true)
 |-- Study_Hours: double (nullable = true)
 |-- Screen_Time: double (nullable = true)
 |-- Caffeine_Intake: integer (nullable = true)
 |-- Physical_Activity: integer (nullable = true)
 |-- Sleep_Quality: integer (nullable = true)
 |-- Weekday_Sleep_Start: double (nullable = true)
 |-- Weekend_Sleep_Start: double (nullable = true)
 |-- Weekday_Sleep_End: double (nullable = true)
 |-- Weekend_Sleep_End: double (nullable = true)
```

3) Show the number of records belongs to class 0 and class 1. Use some graph to verify the class imbalance problem.

   Using Gender as Label for the class 0,1 :

   val classCounts =df.groupBy("Gender").count()

```
scala> val classCounts = df.groupBy("Gender").count()
classCounts: org.apache.spark.sql.DataFrame = [Gender: string, count: bigint]

scala> classCounts.show()
+------+-----+
|Gender|count|
+------+-----+
|Female|  166|
| Other|  148|
|  Male|  186|
+------+-----+
```

4) For Tabular Data: Use some normalization techniques to transform the given dataset values to particular range (Z-score normalization, Min-max normalization, etc.,)

```
import org.apache.spark.ml.feature.VectorAssembler

val featureColumns = Array("Age","Sleep_Duration", "Sleep_Quality", "Screen_Time",
"Physical_Activity")


val assembler = new
VectorAssembler().setInputCols(featureColumns).setOutputCol("features")

val assembledData = assembler.transform(df)


val finalData = assembledData.select(col("Sleep_Duration").as("label"), col("features"))
```

```
scala> val featureColumns = Array("Age","Sleep_Duration", "Sleep_Quality", "Screen_Time", "Physical_Activity")
featureColumns: Array[String] = Array(Age, Sleep_Duration, Sleep_Quality, Screen_Time, Physical_Activity)

scala> val assembler = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_fc8d6e8c3fb1, handleInvalid=error, numInputCols=5

scala> val assembledData = assembler.transform(df)
assembledData: org.apache.spark.sql.DataFrame = [Student_ID: int, Age: int ... 13 more fields]
```

```
scala> val finalData = assembledData.select(col("sleep_duration").as("label"), col("features"))
finalData: org.apache.spark.sql.DataFrame = [label: double, features: vector]

scala> finalData.show()
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  7.7|[24.0,7.7,10.0,3....|
|  6.3|[21.0,6.3,2.0,1.9...|
|  5.1|[22.0,5.1,5.0,3.9...|
|  6.3|[24.0,6.3,9.0,2.8...|
|  4.7|[20.0,4.7,3.0,2.7...|
|  4.9|[25.0,4.9,9.0,3.2...|
|  6.5|[22.0,6.5,6.0,3.4...|
|  6.1|[22.0,6.1,4.0,3.0...|
|  8.6|[24.0,8.6,7.0,1.4...|
|  5.8|[19.0,5.8,8.0,2.0...|
|  6.9|[20.0,6.9,6.0,3.8...|
|  7.2|[24.0,7.2,5.0,1.7...|
|  4.1|[20.0,4.1,6.0,2.1...|
|  7.3|[20.0,7.3,5.0,2.3...|
|  4.9|[25.0,4.9,8.0,2.3...|
|  8.8|[22.0,8.8,1.0,2.8...|
|  4.7|[21.0,4.7,7.0,3.8...|
|  6.1|[25.0,6.1,10.0,1....|
|  4.4|[25.0,4.4,10.0,1....|
|  9.0|[20.0,9.0,2.0,1.6...|
+-----+--------------------+
only showing top 20 rows
```

```
scala> import org.apache.spark.ml.feature.{VectorAssembler, StandardScaler}
import org.apache.spark.ml.feature.{VectorAssembler, StandardScaler}
```

```
scala> balancedData.show
+-----+--------------------+
|label|            features|
+-----+--------------------+
|    1|[24.0,7.7,10.0,3....|
|    1|[21.0,6.3,2.0,1.9...|
|    0|[22.0,5.1,5.0,3.9...|
|    1|[24.0,6.3,9.0,2.8...|
|    0|[20.0,4.7,3.0,2.7...|
|    0|[25.0,4.9,9.0,3.2...|
|    1|[22.0,6.5,6.0,3.4...|
|    1|[22.0,6.1,4.0,3.0...|
|    0|[24.0,8.6,7.0,1.4...|
|    0|[19.0,5.8,8.0,2.0...|
|    1|[20.0,6.9,6.0,3.8...|
|    1|[24.0,7.2,5.0,1.7...|
|    0|[20.0,4.1,6.0,2.1...|
|    1|[20.0,7.3,5.0,2.3...|
|    0|[25.0,4.9,8.0,2.3...|
|    0|[22.0,8.8,1.0,2.8...|
|    0|[21.0,4.7,7.0,3.8...|
|    1|[25.0,6.1,10.0,1....|
|    0|[25.0,4.4,10.0,1....|
|    0|[20.0,9.0,2.0,1.6...|
+-----+--------------------+
only showing top 20 rows
```

Z-score normalization:

```
scala> import org.apache.spark.ml.feature.{VectorAssembler, StandardScaler}
import org.apache.spark.ml.feature.{VectorAssembler, StandardScaler}

scala> val scaler = new StandardScaler()
scaler: org.apache.spark.ml.feature.StandardScaler = stdScal_910f160ba6e9

scala>   .setInputCol("features") // Input column
res11: scaler.type = stdScal_910f160ba6e9

scala>   .setOutputCol("scaledFeatures") // Output column
res12: res11.type = stdScal_910f160ba6e9

scala>   .setWithMean(true) // Center the data to have a mean of 0
res13: res12.type = stdScal_910f160ba6e9

scala>   .setWithStd(true)
res14: res13.type = stdScal_910f160ba6e9

scala> val scalerModel = scaler.fit(finalData)
scalerModel: org.apache.spark.ml.feature.StandardScalerModel = StandardScalerModel: uid=stdScal_910f160ba6e9, numFeatures=5, withMean=true, withStd=true
```

```
scala> val scaledData = scalerModel.transform(finalData)
scaledData: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 1 more field]

scala> scaledData.select("label", "features", "scaledFeatures").show(5, false)
+-----+-----------------------+----------------------------------------------------------------------------------------------------------------+
|label|features               |scaledFeatures                                                                                                  |
+-----+-----------------------+----------------------------------------------------------------------------------------------------------------+
|7.9  |[24.0,7.7,10.0,3.4,37.0]|[1.0560827587359116,0.8262418524271722,1.5630641022994685,1.0181357780838343,-0.7201135059551418]              |
|6.0  |[21.0,6.3,2.0,1.9,74.0] |[-0.22973228842631546,-0.11603461661653158,-1.1330361172770183,-0.7272398414884593,0.3312715354914761]        |
|6.7  |[22.0,5.1,5.0,3.9,53.0] |[0.19887272729442693,-0.9237001615111347,-0.12199853493583576,1.5999276512745988,-0.26546051505930707]         |
|8.6  |[24.0,6.3,9.0,2.8,55.0] |[1.0560827587359116,-0.11603461661653158,1.2260515748524077,0.3199855302549168,-0.20862889119732775]           |
|2.7  |[20.0,4.7,3.0,2.7,85.0] |[-0.6583373041470578,-1.1929220098093354,-0.7960235898299575,0.20362715561676428,0.6438454667323623]           |
+-----+-----------------------+----------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

5) For Text Data: Use Tokenization, Stop words removal, Lemmatization, Text Encoding Techniques (Word Embedding)

   Our data needs no removal numerical datas only

6) Split the dataset into training and testing (70-30 ratio).

val Array(trainingData, testData) = finalData.randomSplit(Array(0.7, 0.3), seed = 1234)

```
scala> val Array(trainingData, testData) = finalData.randomSplit(Array(0.7, 0.3), seed = 1234)
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
```

Before Task 3 we import following

import org.apache.spark.ml.classification.{DecisionTreeClassifier, NaiveBayes, LogisticRegression, GBTClassifier, RandomForestClassifier, LinearSVC}

import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator, BinaryClassificationEvaluator}

Task 3: Apply any three classification algorithms and report the % value of the following performance metrics.

val remappedData = finalData.withColumn("label", when(col("label") >= 6 && col("label") <= 8, 1).otherwise(0))

//Here we Balanced the data as it was not balanced with the label classification of the sleep_Duration

//made sleep time >=8 as 1 and others 0 so that we can do classifier

val Array(trainingData, testData) = remappedData.randomSplit(Array(0.7, 0.3), seed = 1234)

val dt = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")

val dtModel = dt.fit(trainingData)

val dtPredictions = dtModel.transform(testData)

```
scala> val dt = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")
dt: org.apache.spark.ml.classification.DecisionTreeClassifier = dtc_e1fd8929e7f4

scala> val trainingData = finalData.withColumn("label", round(col("label")).cast("Int"))
trainingData: org.apache.spark.sql.DataFrame = [label: int, features: vector]

scala> val Array(trainingData, testData) = finalDataWithRoundedLabels.randomSplit(Array(0.7, 0.3), seed = 1234)
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]

scala> val dtModel = dt.fit(trainingData)
dtModel: org.apache.spark.ml.classification.DecisionTreeClassificationModel = DecisionTreeClassificationModel: uid=dtc_e1fd8929e7f4, depth=5, numNodes=51,
numFeatures=5

scala> val dtPredictions = dtModel.transform(testData)
dtPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 more fields]
```

```
scala> dtPredictions.show()
+-----+--------------------+------------+--------------------+----------+
|label|            features|rawPrediction|         probability|prediction|
+-----+--------------------+------------+--------------------+----------+
|    0|[18.0,4.1,1.0,2.3...|  [38.0,11.0]|[0.77551020408163...|       0.0|
|    0|[18.0,4.1,5.0,3.2...|  [38.0,11.0]|[0.77551020408163...|       0.0|
|    0|[18.0,4.3,8.0,2.0...|  [38.0,11.0]|[0.77551020408163...|       0.0|
|    0|[18.0,6.1,8.0,2.4...|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[18.0,6.6,10.0,1....|   [13.0,4.0]|[0.76470588235294...|       0.0|
|    0|[18.0,6.8,1.0,3.8...|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[18.0,7.0,1.0,1.7...|  [33.0,34.0]|[0.49253731343283...|       1.0|
|    0|[18.0,7.0,1.0,3.0...|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[18.0,7.0,4.0,1.4...|  [33.0,34.0]|[0.49253731343283...|       1.0|
|    0|[18.0,7.5,5.0,1.3...|  [33.0,34.0]|[0.49253731343283...|       1.0|
|    0|[18.0,7.5,5.0,3.9...|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[18.0,7.8,10.0,2....|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[18.0,8.8,7.0,1.9...|   [31.0,2.0]|[0.93939393939393...|       0.0|
|    0|[18.0,9.0,1.0,3.5...|    [0.0,1.0]|           [0.0,1.0]|       1.0|
|    0|[19.0,4.4,2.0,3.7...|  [38.0,11.0]|[0.77551020408163...|       0.0|
|    0|[19.0,5.1,3.0,1.6...|  [33.0,34.0]|[0.49253731343283...|       1.0|
|    0|[19.0,5.4,10.0,2....|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[19.0,5.8,4.0,1.5...|  [33.0,34.0]|[0.49253731343283...|       1.0|
|    0|[19.0,6.6,2.0,2.6...|  [80.0,43.0]|[0.65040650406504...|       0.0|
|    0|[19.0,6.9,9.0,3.7...|  [80.0,43.0]|[0.65040650406504...|       0.0|
+-----+--------------------+------------+--------------------+----------+
only showing top 20 rows
```

//We ran this logistic reg with the previous training and testing data

val lr = new LogisticRegression().setLabelCol("label").setFeaturesCol("features")

val lrModel = lr.fit(trainingData)

val lrPredictions = lrModel.transform(testData)

```
scala> val lr = new LogisticRegression().setLabelCol("label").setFeaturesCol("features")
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_3b5e23993637

scala> val lrModel = lr.fit(trainingData)
lrModel: org.apache.spark.ml.classification.LogisticRegressionModel = LogisticRegressionModel: uid=logreg_3b5e23993637, numClasses=13, numFeatures=5

scala> val lrPredictions = lrModel.transform(testData)
lrPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 more fields]
```

```
abhishek@DESKTOP-70BSBC

scala> lrPredictions.show()
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|    0|[18.0,4.1,1.0,2.3...|[0.72567958581741...|[0.67385647234108...|       0.0|
|    0|[18.0,4.1,5.0,3.2...|[0.71182797046808...|[0.67080494857246...|       0.0|
|    0|[18.0,4.3,8.0,2.0...|[0.48837927922514...|[0.61972455769445...|       0.0|
|    0|[18.0,6.1,8.0,2.4...|[0.57214530251834...|[0.63925804500199...|       0.0|
|    0|[18.0,6.6,10.0,1....|[0.44671253113506...|[0.60985732507708...|       0.0|
|    0|[18.0,6.8,1.0,3.8...|[0.98849099871838...|[0.72878976332670...|       0.0|
|    0|[18.0,7.0,1.0,1.7...|[0.74394897621247...|[0.67785878434143...|       0.0|
|    0|[18.0,7.0,1.0,3.0...|[0.87761302856531...|[0.70632733788878...|       0.0|
|    0|[18.0,7.0,4.0,1.4...|[0.62679638545627...|[0.65176269790591...|       0.0|
|    0|[18.0,7.5,5.0,1.3...|[0.67876673735261...|[0.66346339040069...|       0.0|
|    0|[18.0,7.5,5.0,3.9...|[0.92812011108014...|[0.71669374119907...|       0.0|
|    0|[18.0,7.8,10.0,2....|[0.51903900583750...|[0.62692302629375...|       0.0|
|    0|[18.0,8.8,7.0,1.9...|[0.72696881609331...|[0.67413974800872...|       0.0|
|    0|[18.0,9.0,1.0,3.5...|[1.01777113405501...|[0.73453821608337...|       0.0|
|    0|[19.0,4.4,2.0,3.7...|[0.80318864653516...|[0.69065615003393...|       0.0|
|    0|[19.0,5.1,3.0,1.6...|[0.61412373519944...|[0.64888090696796...|       0.0|
|    0|[19.0,5.4,10.0,2....|[0.50844356728718...|[0.62444153949128...|       0.0|
|    0|[19.0,5.8,4.0,1.5...|[0.63539691097005...|[0.65371218613430...|       0.0|
|    0|[19.0,6.6,2.0,2.6...|[0.86283258336006...|[0.70325212225864...|       0.0|
|    0|[19.0,6.9,9.0,3.7...|[0.68893662465263...|[0.66573033187782...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 20 rows
```

```
val remappedData = finalData.withColumn("label", when(col("label") >= 8, 1).otherwise(0))
//made sleep time >=8  as 1 and others 0 so that we can do gbt classifier
val Array(trainingData, testData) = remappedData.randomSplit(Array(0.7, 0.3), seed = 1234)
val gbt = new GBTClassifier().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)
val gbtModel = gbt.fit(trainingData)
val gbtPredictions = gbtModel.transform(testData)
```

```
scala> val remappedData = finalData.withColumn("label", when(col("label") >= 8, 1).otherwise(0))
remappedData: org.apache.spark.sql.DataFrame = [label: int, features: vector]

scala> val Array(trainingData, testData) = remappedData.randomSplit(Array(0.7, 0.3), seed = 1234)
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: int, features: vector]

scala> val gbt = new GBTClassifier().setLabelCol("label").setFeaturesCol("features").setMaxIter(10)
gbt: org.apache.spark.ml.classification.GBTClassifier = gbtc_587d1bf4e7d5
```

```
scala> val gbtModel = gbt.fit(trainingData)
gbtModel: org.apache.spark.ml.classification.GBTClassificationModel = GBTClassificationModel: uid = gbtc_587d1bf4e7d5, numTrees=10, numClasses=2, numFeat

scala> val gbtPredictions = gbtModel.transform(testData)
gbtPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 more fields]
```

```
scala> gbtPredictions.show
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|    0|[18.0,4.1,1.0,2.3...|[0.53524822898147...|[0.74469131693909...|       0.0|
|    0|[18.0,4.1,5.0,3.2...|[0.03174391352652...|[0.51586662764771...|       0.0|
|    0|[18.0,4.3,8.0,2.0...|[0.47770648161860...|[0.72220246464057...|       0.0|
|    0|[18.0,6.1,8.0,2.4...|[0.12079914460234...|[0.56010748484534...|       0.0|
|    0|[18.0,6.6,10.0,1....|[0.42672251103559...|[0.70128931476836...|       0.0|
|    0|[18.0,6.8,1.0,3.8...|[0.16398204143514...|[0.58126392454276...|       0.0|
|    0|[18.0,7.0,1.0,1.7...|[0.48842727854672...|[0.72648364606462...|       0.0|
|    0|[18.0,7.0,1.0,3.0...|[0.45920589517202...|[0.71471838654402...|       0.0|
|    0|[18.0,7.0,4.0,1.4...|[-0.1853498057769...|[0.40837198157383...|       1.0|
|    0|[18.0,7.5,5.0,1.3...|[-0.1511702560221...|[0.42498542549833...|       1.0|
|    0|[18.0,7.5,5.0,3.9...|[-0.0561202217802...|[0.47196931029194...|       1.0|
|    0|[18.0,7.8,10.0,2....|[0.21036892048718...|[0.60365979524144...|       0.0|
|    0|[18.0,8.8,7.0,1.9...|[1.05596504701043...|[0.89205732470926...|       0.0|
|    0|[18.0,9.0,1.0,3.5...|[-0.6213157051573...|[0.22397828283305...|       1.0|
|    0|[19.0,4.4,2.0,3.7...|[0.60454181963638...|[0.77013676941495...|       0.0|
|    0|[19.0,5.1,3.0,1.6...|[0.01722792515671...|[0.50861311046745...|       0.0|
|    0|[19.0,5.4,10.0,2....|[0.04860661403482...|[0.52428418539981...|       0.0|
|    0|[19.0,5.8,4.0,1.5...|[-0.1806101270372...|[0.41066420896749...|       1.0|
|    0|[19.0,6.6,2.0,2.6...|[0.49983327497247...|[0.73099301331907...|       0.0|
|    0|[19.0,6.9,9.0,3.7...|[0.39435237886906...|[0.68755314931383...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 20 rows
```

val rf = new RandomForestClassifier() .setLabelCol("label")
.setFeaturesCol("features").setNumTrees(10)

val rfModel = rf.fit(trainingData)

val rfPredictions = rfModel.transform(testData)

```
scala> val rfPredictions = rfModel.transform(testData)
rfPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 more fields]

scala> val rf = new RandomForestClassifier() .setLabelCol("label") .setFeaturesCol("features").setNumTrees(10)

rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_87f143ec4e16

scala> val rfModel = rf.fit(trainingData)
rfModel: org.apache.spark.ml.classification.RandomForestClassificationModel = RandomForestClassificationModel:
 uid=rfc_87f143ec4e16, numTrees=10, numClasses=2, numFeatures=5

scala> val rfPredictions = rfModel.transform(testData)
rfPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 more fields]
```

val nb = new NaiveBayes() .setLabelCol("label") .setFeaturesCol("features")

val nbModel = nb.fit(trainingData)

val nbPredictions = nbModel.transform(testData)

```
scala> val nb = new NaiveBayes()
nb: org.apache.spark.ml.classification.NaiveBayes = nb_35d86f35c888

scala>   .setLabelCol("label")
res9: org.apache.spark.ml.classification.NaiveBayes = nb_35d86f35c888

scala>   .setFeaturesCol("features")
res10: org.apache.spark.ml.classification.NaiveBayes = nb_35d86f35c888

scala> val nbModel = nb.fit(trainingData)
nbModel: org.apache.spark.ml.classification.NaiveBayesModel = NaiveBayesModel: uid=nb_
35d86f35c888, modelType=multinomial, numClasses=2, numFeatures=5

scala> val nbPredictions = nbModel.transform(testData)
nbPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 3 mo
re fields]
```

val lsvc = new LinearSVC() .setLabelCol("label") .setFeaturesCol("features") .setMaxIter(10)

val lsvcModel = lsvc.fit(trainingData)

val lsvcPredictions = lsvcModel.transform(testData)

```
scala> val lsvc = new LinearSVC()
lsvc: org.apache.spark.ml.classification.LinearSVC = linearsvc_50b24c406c25

scala>   .setLabelCol("label")
res11: org.apache.spark.ml.classification.LinearSVC = linearsvc_50b24c406c25

scala>   .setFeaturesCol("features")
res12: org.apache.spark.ml.classification.LinearSVC = linearsvc_50b24c406c25

scala>   .setMaxIter(10)
res13: res12.type = linearsvc_50b24c406c25

scala> val lsvcModel = lsvc.fit(trainingData)
24/11/07 13:26:00 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic
.netlib.blas.JNIBLAS
lsvcModel: org.apache.spark.ml.classification.LinearSVCModel = LinearSVCModel: uid=lin
earsvc_50b24c406c25, numClasses=2, numFeatures=5

scala> val lsvcPredictions = lsvcModel.transform(testData)
lsvcPredictions: org.apache.spark.sql.DataFrame = [label: int, features: vector ... 2
more fields]
```

I. Confusion matrix

val confusionMatrix = dtPredictions.groupBy("label", "prediction").count()

confusionMatrix.show()

```
scala> val confusionMatrix = dtPredictions.groupBy("label", "prediction").count()
confusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double ...

scala> confusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    0|       0.0|   92|
|    1|       1.0|   62|
|    0|       1.0|    5|
+-----+----------+-----+
```

val confusionMatrix = lrPredictions.groupBy("label", "prediction").count()
confusionMatrix.show()

//For the testdata as per remapped data:

```
scala> val confusionMatrix = lrPredictions.groupBy("label", "prediction").count()
confusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double .

scala> confusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0|   37|
|    0|       0.0|   65|
|    1|       1.0|   25|
|    0|       1.0|   32|
+-----+----------+-----+
```

val confusionMatrix = gbtPredictions.groupBy("label", "prediction").count()
confusionMatrix.show()

```
scala> val confusionMatrix = gbtPredictions.groupBy("label", "prediction").count()
confusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double .

scala> confusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0|    3|
|    0|       0.0|   95|
|    1|       1.0|   59|
|    0|       1.0|    2|
+-----+----------+-----+
```

val confusionMatrix = rfPredictions.groupBy("label", "prediction").count()
confusionMatrix.show()

```
scala> val confusionMatrix = rfPredictions.groupBy("label", "prediction").count()
confusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double ... 1 more field]

scala> confusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0|    2|
|    0|       0.0|   89|
|    1|       1.0|   60|
|    0|       1.0|    8|
+-----+----------+-----+
```

val nbConfusionMatrix = nbPredictions.groupBy("label", "prediction").count()
nbConfusionMatrix.show()

```
scala> val nbConfusionMatrix = nbPredictions.groupBy("label", "prediction").count()
nbConfusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double ..
. 1 more field]

scala> nbConfusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0|   41|
|    0|       0.0|   80|
|    1|       1.0|   21|
|    0|       1.0|   17|
+-----+----------+-----+
```

val lsvcConfusionMatrix = lsvcPredictions.groupBy("label", "prediction").count()
lsvcConfusionMatrix.show()

```
scala> val lsvcConfusionMatrix = lsvcPredictions.groupBy("label", "prediction").count(
)
lsvcConfusionMatrix: org.apache.spark.sql.DataFrame = [label: int, prediction: double
... 1 more field]

scala> lsvcConfusionMatrix.show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0|    8|
|    0|       0.0|   65|
|    1|       1.0|   54|
|    0|       1.0|   32|
+-----+----------+-----+
```

II. Accuracy, Precision, Recall, F1-Score, RoC-AUC

val evaluatorAccuracy = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("prediction") .setMetricName("accuracy")

val evaluatorPrecision = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("prediction") .setMetricName("weightedPrecision")

val evaluatorRecall = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("prediction") .setMetricName("weightedRecall")

val evaluatorF1 = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("prediction") .setMetricName("f1")

val evaluatorROC = new BinaryClassificationEvaluator() .setLabelCol("label") .setRawPredictionCol("prediction") .setMetricName("areaUnderROC")

```
scala> val evaluatorAccuracy = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("predict
ion") .setMetricName("accuracy")
evaluatorAccuracy: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator
: uid=mcEval_9aa7e9ef7b9d, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>

scala> val evaluatorPrecision = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("predic
tion") .setMetricName("weightedPrecision")
evaluatorPrecision: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluato
r: uid=mcEval_f9500fadf3ce, metricName=weightedPrecision, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>

scala> val evaluatorRecall = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("predictio
n") .setMetricName("weightedRecall")
evaluatorRecall: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator:
uid=mcEval_a42bda0b201a, metricName=weightedRecall, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>
     | val evaluatorF1 = new MulticlassClassificationEvaluator() .setLabelCol("label") .setPredictionCol("prediction")
.setMetricName("f1")
evaluatorF1: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=
mcEval_b6a421143d1a, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>

scala> val evaluatorROC = new BinaryClassificationEvaluator() .setLabelCol("label") .setRawPredictionCol("prediction")
.setMetricName("areaUnderROC")
evaluatorROC: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = BinaryClassificationEvaluator: uid=binEval
_bb210e8466e0, metricName=areaUnderROC, numBins=1000

scala>
```

// Calculate each metric for Decision Tree predictions
val accuracy = evaluatorAccuracy.evaluate(dtPredictions)
val precision = evaluatorPrecision.evaluate(dtPredictions)
val recall = evaluatorRecall.evaluate(dtPredictions)
val f1Score = evaluatorF1.evaluate(dtPredictions)
val rocAuc = evaluatorROC.evaluate(dtPredictions)

```
scala> val accuracy = evaluatorAccuracy.evaluate(dtPredictions)
accuracy: Double = 0.9685534591194969

scala> val precision = evaluatorPrecision.evaluate(dtPredictions)
precision: Double = 0.970900215901624

scala> val recall = evaluatorRecall.evaluate(dtPredictions)
recall: Double = 0.9685534591194969

scala> val f1Score = evaluatorF1.evaluate(dtPredictions)
f1Score: Double = 0.9687469286708719

scala> val rocAuc = evaluatorROC.evaluate(dtPredictions)
rocAuc: Double = 0.9742268041237113
```

// Calculate each metric for Logistic Regression predictions
val accuracy = evaluatorAccuracy.evaluate(lrPredictions)
val precision = evaluatorPrecision.evaluate(lrPredictions)
val recall = evaluatorRecall.evaluate(lrPredictions)
val f1Score = evaluatorF1.evaluate(lrPredictions)
val rocAuc = evaluatorROC.evaluate(lrPredictions)

```
scala> val accuracy = evaluatorAccuracy.evaluate(lrPredictions)
accuracy: Double = 0.5660377358490566

scala> val precision = evaluatorPrecision.evaluate(lrPredictions)
precision: Double = 0.5597906160146946

scala> val recall = evaluatorRecall.evaluate(lrPredictions)
recall: Double = 0.5660377358490566

scala> val f1Score = evaluatorF1.evaluate(lrPredictions)
f1Score: Double = 0.5623726687982484

scala> val rocAuc = evaluatorROC.evaluate(lrPredictions)
rocAuc: Double = 0.5366644496175591
```

// Calculate each metric for Logistic Regression predictions
val accuracy = evaluatorAccuracy.evaluate(gbtPredictions)
val precision = evaluatorPrecision.evaluate(gbtPredictions)
val recall = evaluatorRecall.evaluate(gbtPredictions)
val f1Score = evaluatorF1.evaluate(gbtPredictions)
val rocAuc = evaluatorROC.evaluate(gbtPredictions)

```
scala> val accuracy = evaluatorAccuracy.evaluate(gbtPredictions)
accuracy: Double = 0.9685534591194969

scala> val precision = evaluatorPrecision.evaluate(gbtPredictions)
precision: Double = 0.9685397821361763

scala> val recall = evaluatorRecall.evaluate(gbtPredictions)
recall: Double = 0.9685534591194969

scala> val f1Score = evaluatorF1.evaluate(gbtPredictions)
f1Score: Double = 0.9685062598086067

scala> val rocAuc = evaluatorROC.evaluate(gbtPredictions)
rocAuc: Double = 0.9654971732623878
```

val accuracy = evaluatorAccuracy.evaluate(rfPredictions)
val precision = evaluatorPrecision.evaluate(rfPredictions)
val recall = evaluatorRecall.evaluate(rfPredictions)
val f1Score = evaluatorF1.evaluate(rfPredictions)
val rocAuc = evaluatorROC.evaluate(rfPredictions)

```
scala> val accuracy = evaluatorAccuracy.evaluate(rfPredictions)
accuracy: Double = 0.9371069182389937

scala> val precision = evaluatorPrecision.evaluate(rfPredictions)
precision: Double = 0.9407170705727864

scala> val recall = evaluatorRecall.evaluate(rfPredictions)
recall: Double = 0.9371069182389937

scala> val f1Score = evaluatorF1.evaluate(rfPredictions)
f1Score: Double = 0.9375546840420385

scala> val rocAuc = evaluatorROC.evaluate(rfPredictions)
rocAuc: Double = 0.9426338543398736
```

// Evaluation for Linear SVC Predictions
val lsvcAccuracy = evaluatorAccuracy.evaluate(lsvcPredictions)
val lsvcPrecision = evaluatorPrecision.evaluate(lsvcPredictions)
val lsvcRecall = evaluatorRecall.evaluate(lsvcPredictions)
val lsvcF1Score = evaluatorF1.evaluate(lsvcPredictions)
val lsvcRocAuc = evaluatorROC.evaluate(lsvcPredictions)

```
scala> val lsvcAccuracy = evaluatorAccuracy.evaluate(lsvcPredictions)
lsvcAccuracy: Double = 0.7484276729559748

scala> val lsvcPrecision = evaluatorPrecision.evaluate(lsvcPredictions)
lsvcPrecision: Double = 0.788050915546152

scala> val lsvcRecall = evaluatorRecall.evaluate(lsvcPredictions)
lsvcRecall: Double = 0.7484276729559749

scala> val lsvcF1Score = evaluatorF1.evaluate(lsvcPredictions)
lsvcF1Score: Double = 0.7510673825879154

scala> val lsvcRocAuc = evaluatorROC.evaluate(lsvcPredictions)
lsvcRocAuc: Double = 0.7705354173594946
```

// Evaluation for Naive Bayes Predictions
val nbAccuracy = evaluatorAccuracy.evaluate(nbPredictions)
val nbPrecision = evaluatorPrecision.evaluate(nbPredictions)
val nbRecall = evaluatorRecall.evaluate(nbPredictions)
val nbF1Score = evaluatorF1.evaluate(nbPredictions)
val nbRocAuc = evaluatorROC.evaluate(nbPredictions)

```
scala> // Evaluation for Naive Bayes Predictions

scala> val nbAccuracy = evaluatorAccuracy.evaluate(nbPredictions)
nbAccuracy: Double = 0.6352201257861635

scala> val nbPrecision = evaluatorPrecision.evaluate(nbPredictions)
nbPrecision: Double = 0.6188389264131793

scala> val nbRecall = evaluatorRecall.evaluate(nbPredictions)
nbRecall: Double = 0.6352201257861635

scala> val nbF1Score = evaluatorF1.evaluate(nbPredictions)
nbF1Score: Double = 0.6115261669840171

scala> val nbRocAuc = evaluatorROC.evaluate(nbPredictions)
nbRocAuc: Double = 0.5817259727302959
```

Table - Results Comparison

Classifier	Accuracy	Precision	Recall	F1-Score	RoC-AUC
Decision Tree	96.85%	97.09%	96.85%	96.87%	97.42%
Logistic Regression	56.60%	55.97%	56.60%	56.23%	53.66%
Gradient Boosted	96.855%	96.853%	96.855%	96.850%	96.549%
NaiveBias	63.52%	61.88%	63.52%	61.15%	58.17%
Random Forest	93.71%	94.07%	93.71%	93.75%	94.26%
Linear SVC	74.84%	78.80%	74.84%	75.10%	77.05%

Task 4: Justification on the best performance model with supporting discussions.

Here I explored the adequate sleep patterns of students and attempted to predict sleep quality using above three Classifier. The model achieved a reasonable accuracy, indicating that factors such as Age,Sleep_Duration,Sleep_Quality,Screen_Time,Physical_Activity do have an impact on sleep hours.

Modifying the dataset to label sleep hours as 1 if they are greater than 6 and less than equal to 8, and 0 otherwise, creates a binary classification task that directly identifies sufficient versus insufficient sleep..

1. **Managing Class Imbalance**: In cases of imbalance between samples with sleep 8>=hours >=6 .

   Based on the performance metrics of the classifiers, we can draw several conclusions about the relationship between sleep quality and the models used to predict it:

1. **High Predictive Accuracy**: The Decision Tree and Gradient Boosted classifiers demonstrate exceptionally high accuracy (approximately 96.85% and 96.855%, respectively). This indicates that these models are effective at distinguishing between different levels of sleep quality based on the features provided in the dataset.

2. **Precision and Recall Balance**: The high precision (97.09% for Decision Tree and 96.853% for Gradient Boosted) suggests that when these models predict a certain sleep quality class, they are correct most of the time. However, the recall metrics also indicate that they successfully identify a significant portion of actual sleep quality cases, minimizing the risk of missing individuals who may need attention for sleep issues.
3. **Weak Performance of Logistic Regression**: The Logistic Regression model has much lower performance metrics (accuracy of 56.60%, precision of 55.97%, recall of 56.60%), suggesting that it may not capture the complexities of the relationship between the features and sleep quality as effectively as the other classifiers. This could indicate that the underlying data structure is nonlinear or that there are interactions between features that Logistic Regression cannot adequately model.
4. • **Implications for Sleep Quality Assessment**: The strong performance of the Decision Tree and Gradient Boosted models implies that certain measurable features (possibly like sleep duration, disturbances, etc.) are significant predictors of sleep quality. If these features can be identified and measured accurately, it could lead to better assessments and interventions for improving sleep quality.

Conclusion:

Overall, the findings suggest that certain modeling techniques (Decision Tree and Gradient Boosted) can effectively predict sleep quality, highlighting the importance of using appropriate algorithms for this type of classification problem. Further investigation into the features contributing to sleep quality can enhance our understanding and lead to improved strategies for sleep health interventions.

Date - 7/11/24.

Assignment 2

Big data & Large Scale Computing.

Topic :- sleep quality Analysis

Name :- 1) Niraj poradwar Roll - 2021BCS00064

2) Abhishek Harsh Roll - 2021BCS00036.

Dataset :- student_sleep_patterns.csv dataset for the classification in the adequate and inadequate sleep for the student of the data set consisting 500 datas in the following formats

Scalpdf.show(5);

Student ID	Age	Gender	university year	sleep Duration	study Hours	screen time	caffein intake	physical Activity	sleep quality
1	24	Other	2nd year	7.7	7.9	3.4	2	37	10
2	21	male	1st year	6.3	6.0	1.9	5	74	2
3	22	male	4th year	5.1	6.7	3.9	5	53	5
4	24	other	4th year	6.3	8.6	2.8	4	55	9
5.	20	male	4th year	4.7	2.7	2.7	0	85	3

mathematical equations Representation :-

1. feature vector creation : The vector Assembler combines multiple column into a single vector column (features) mathematically, the feature vector for each Record is Represented As

$x_i = [Age_i, sleep\ Duration_i, sleep\ quality_i, screentime_i, physical\ Activity_i]$

Y_i = sleep Duration i

$Record\ i = (Y_i, x_i) = (sleep\ Duration_i, [Age_i, sleep\ Duration_i, sleep\ quality, screen\ time_i, physical\ Activity_i])$

Mathematical Representation of the transformation

let y_i be the original label which is the sleep duration of the ith student):-

$$new\_label_i = \begin{cases} 1 & \text{if } 6 \leq y_i \leq 8 \\ 0 & \text{otherwise} \end{cases}$$

label	features
1	24.0, 7.7, 10.0, 3.---
1	21.0, 6.3, 2.0, 1.9 ---
0	22.0, 5.1, 5.0, 3.9 ---
1	24.0, 6.3, 9.0, 2.8 ---
0	20.0, 4.7, 3.0, 2.7 ----

1. Random split:

The dataset is being split into two parts: one for training the model And one for testing the model. The data is split into a 70% training set And a 30% test set Based on the Random speed value.

Decision tree classifier:- for each node of the tree, we select the feature f and threshold t that minimizes the impurity of the Resulting subsets. The impurity is usually calculated As

Gini impurity → $Gini(D) = 1 - \sum_{K=1}^{K} p_K^2$

where p_K is the proportion of instance in class K in the dataset D,

K is the number of classes

prediction :- $y = f(x)$

where f is the learned decision tree model.

label	prediction	count
1	1	62
0	1	5
1	0	0
0	0	92.

True positive (TP) = 62

false positive (fp) = 5

false Negative (fN) = 0

True Negative (TN) = 92.

Accuracy :-

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+fp+fN} = \frac{62+92}{62+92+5+0} = \frac{154}{159} \approx 0.96$$

$$= 0.9686.$$

so Accuracy = 96 %.

precision :-

$$\text{precision} = \frac{TP}{TP+fp} = \frac{62}{62+5} = \frac{62}{67} \approx 0.9254$$

so precision = 92.5 %.

Recall.

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{62}{62+0} = \frac{62}{62} = 1.$$

Approx 99.

f1 Score.

$$\text{f1 score} = \frac{2 \times \text{precision} \times \text{Recall}}{\text{precision} + \text{Recall}} = \frac{2 \times 0.9254 \times 1.0}{0.9254 + 1.0}$$

So f1 score = 96.15%.

Roc-Auc.

Given the Recall is almost 1. And the precision is high we can infer that the model has good descrimination ability.

Roc-Auc = 0.9742 so, Auc Score = 97.42%.

DT Prediction (

label	features	Raw prediction	probability	prediction
0	18, 4.1, 1.0, 2.8,...	38.0, 11.0	0.7755...	0.0
0	18, 4.1, 5.0, 3.2...	38.0, 11.0	0.7755...	0.0
0	18, 4.3, 8.0, 2.0...	38.0, 11.0	0.7755...	0.0
0	18, 6.1, 8.0, 2.4...	80.0, 43.0	0.6504...	0.0
0	18, 18.6, 10.0, 1,...	13.0, 4.0	0.7647...	0.0