JSS ACADEMY OF TECHNICAL EDUCATION, BANGALORE-560060

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2020-2021(ODD SEM)

SUBJECT: Big Data Analytics[18CS72]

SL no.	Name	USN	Signature
1.	Anjali Sharma	1JS18CS017	
2.	Ankita Chowdhury	1JS18CS019	
3.	Hrunmana B N	1JS19CS402	

Under the Guidance of:

Mr. Manjunath B T

INTRODUCTION

What is Apache Spark?

Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

Features:

- Lightning fast real-time processing framework, Written in Scala (JVM).
- For python interface, Py4J library is used
- in-memory computations, Lazy execution and Parallel Processing
- Apache Hadoop MapReduce was performing batch processing only and lacked a real-time processing feature. Spark does Batch and Real time processing
- It leverages Apache Hadoop for both storage and processing. It uses HDFS (Hadoop Distributed File system) for storage and it can run Spark applications on YARN as well.
- Spark can load data directly from disk, memory and other data storage technologies such as Amazon S3, Hadoop Distributed File System (HDFS), HBase, Cassandra

Difference Between Spark and Map Reduce:

MAP REDUCE	SPARK
Computing Framework Engine, open source managed by Apache	Computing Framework Engine, open source managed by Apache
Yes , Map Reduce is Faster than traditional system but it does not leverage the memory of hadoop cluster to the maximum	spark has been proved to execute the batch processing jobs 10 to 100 times faster
Map Reduce is disk Oriented completely. Higher latency. No caching support.	Spark ensures lower latency computations by caching the partials results across its memory of distributed hardware. Stores data in memory
MapReduce is a cheaper option available while comparing it in terms of cost.	As spark requires a lot of RAM to run in-memory. Thus, increases the cluster, and also its cost.
Writing Map reduce pipelines is complex and lengthy as it is purely Java	Writing Spark code is always easy and we can write in 4 languages
Batch Processing	Batch/Iterative/ Real Time /Interactive Processing
Fault Tolerance and Highly Scalable and Cross platform	Fault Tolerance and Highly Scalable and Cross platform
Map Reduce has been tested on 15000 nodes	Spark has been tested on 8000 nodes
it has not inbuilt support to various things like SQL,ML,RT	it has in built support to various things like SQL,ML,RT
It is basic data processing engine.	It is data analytics engine. Hence, it is a choice for Data Scientist.
MapReduce runs very well on commodity hardware.	Spark needs mid to high-level hardware.

PySpark:

PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

- Apache Spark community released a tool, PySpark.
- It lets us use the power of Apache Spark in order to tame Big Data.
- It is because of a library called Py4j that they are able to achieve this.
- To use PySpark you will have to install python and Apache spark on your machine.
- Using PySpark, we can work with RDDs in Python programming language also.

Installing Spark in Local Environment

1. Install Java 8

Run the executable, and JAVA by default will be installed in:

C:\Program Files\Java\jdk1.8.0_201

Add the following environment variable:

```
JAVA_HOME = C:\Program Files\Java\jdk1.8.0_201
```

Add to PATH variable the following directory:

```
C:\Program Files\Java\jdk1.8.0_201\bin
```

2. Download and Install Spark

Extract the file to your chosen directory (7z can open tgz). In my case, it was C:\spark. There is another compressed directory in the tar, extract it (into here) as well.

Setup the environment variables

```
SPARK_HOME  = C:\spark\spark-2.3.2-bin-hadoop2.7
HADOOP_HOME = C:\spark\spark-2.3.2-bin-hadoop2.7
```

Add the following path to PATH environment variable:

```
C:\spark\spark-2.3.2-bin-hadoop2.7\bin
```

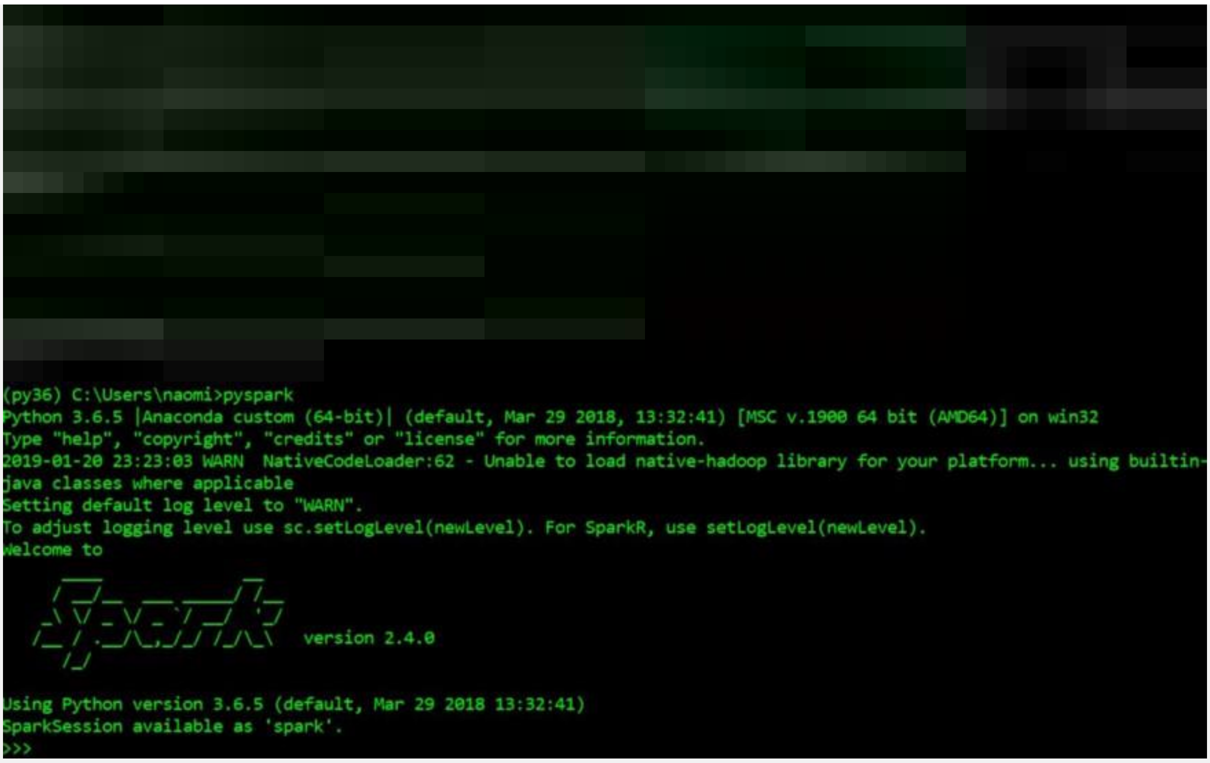
3. Download and setup winutils.exe

In hadoop binaries repository, https://github.com/steveloughran/winutils choose your hadoop version, then goto bin, and download the winutils.exe file. In my case: https://github.com/steveloughran/winutils/blob/master/hadoop-2.7.1/bin/winutils.exe

Save winutils.exe in to bin directory of your spark installation

4. Check PySpark installation

In your anaconda prompt,or any python supporting cmd, type pyspark, to enter pyspark shell. To be prepared, best to check it in the python environment from which you run jupyter notebook. You supposed to see the following:



5. PySpark with Jupyter notebook

Install conda findspark, to access spark instance from jupyter notebook. Check current installation in Anaconda cloud. In time of writing:

```
conda install -c conda-forge findspark
```

Open your python jupyter notebook, and write inside:

```
import findspark
findspark.init()findspark.find()
import pyspark
findspark.find()
```

Last line will output SPARK_HOME path. It's just for test, you can delete it.

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSessionconf = 
pyspark.SparkConf().setAppName('appName').setMaster('lo
cal')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
```

pyspark.sql.SparkSession : Main entry point for DataFrame and SQL functionality.

DATA SETS

Data Set Link: https://www.kaggle.com/mathchi/diabetes-data-set?select=diabetes.csv

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Content

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Number of Instances: 768
Number of Attributes: 8 plus class
For Each Attribute: (all numeric-valued)

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)

6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Missing Attribute Values: Yes
Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Problem Statement:

Diabetes:

According to WHO, Diabetes is a chronic disease that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces. Insulin is a hormone that regulates blood sugar. Hyperglycaemia, or raised blood sugar, is a common effect of uncontrolled diabetes and over time leads to serious damage to many of the body's systems, especially the nerves and blood vessels.

Using various ML Algorithms, we are classifying the datasets and checking which gives better accuracy.

ScreenShots:

In [10]:
```python
import findspark
findspark.init()
findspark.find()
from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import RandomForestClassifier

from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [11]:
```python
spark = SparkSession.builder.appName("Classification with Spark").getOrCreate()
```

In [12]:
```python
dataset = spark.read.csv("diabetes.csv",header=True)
```

In [13]:
```python
dataset.show()
```

```
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin| BMI|DiabetesPedigreeFunction|Age|Outcome|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
|          6|    148|           72|           35|      0|33.6|                   0.627| 50|      1|
|          1|     85|           66|           29|      0|26.6|                   0.351| 31|      0|
|          8|    183|           64|            0|      0|23.3|                   0.672| 32|      1|
|          1|     89|           66|           23|     94|28.1|                   0.167| 21|      0|
|          0|    137|           40|           35|    168|43.1|                   2.288| 33|      1|
|          5|    116|           74|            0|      0|25.6|                   0.201| 30|      0|
|          3|     78|           50|           32|     88|  31|                   0.248| 26|      1|
|         10|    115|            0|            0|      0|35.3|                   0.134| 29|      0|
|          2|    197|           70|           45|    543|30.5|                   0.158| 53|      1|
|          8|    125|           96|            0|      0|   0|                   0.232| 54|      1|
|          4|    110|           92|            0|      0|37.6|                   0.191| 30|      0|
|         10|    168|           74|            0|      0|  38|                   0.537| 34|      1|
|         10|    139|           80|            0|      0|27.1|                   1.441| 57|      0|
|          1|    189|           60|           23|    846|30.1|                   0.398| 59|      1|
|          5|    166|           72|           19|    175|25.8|                   0.587| 51|      1|
|          7|    100|            0|            0|      0|  30|                   0.484| 32|      1|
|          0|    118|           84|           47|    230|45.8|                   0.551| 31|      1|
|          7|    107|           74|            0|      0|29.6|                   0.254| 31|      1|
|          1|    103|           30|           38|     83|43.3|                   0.183| 33|      0|
|          1|    115|           70|           30|     96|34.6|                   0.529| 32|      1|
+-----------+-------+-------------+-------------+-------+----+------------------------+---+-------+
only showing top 20 rows
```

In [14]:
```python
dataset.printSchema()
```

```
root
 |-- Pregnancies: string (nullable = true)
 |-- Glucose: string (nullable = true)
 |-- BloodPressure: string (nullable = true)
 |-- SkinThickness: string (nullable = true)
 |-- Insulin: string (nullable = true)
 |-- BMI: string (nullable = true)
 |-- DiabetesPedigreeFunction: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Outcome: string (nullable = true)
```

In [15]:
```python
from pyspark.sql.functions import col
new_data = dataset.select(*(col(c).cast("float").alias(c) for c in dataset.columns))
```

In [16]:
```python
new_data.printSchema()
```

```
root
 |-- Pregnancies: float (nullable = true)
 |-- Glucose: float (nullable = true)
 |-- BloodPressure: float (nullable = true)
 |-- SkinThickness: float (nullable = true)
 |-- Insulin: float (nullable = true)
 |-- BMI: float (nullable = true)
 |-- DiabetesPedigreeFunction: float (nullable = true)
 |-- Age: float (nullable = true)
 |-- Outcome: float (nullable = true)
```

In [17]:

```
from pyspark.sql.functions import col, count, isnan, when
#checking for null ir nan type values in our columns
new_data.select([count(when(col(c).isNull(), c)).alias(c) for c in new_data.columns]).show()
```

```
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|BMI|DiabetesPedigreeFunction|Age|Outcome|
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
|          0|      0|            0|            0|      0|  0|                       0|  0|      0|
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
```

In [18]:

```
cols=new_data.columns
cols.remove("Outcome")
assembler = VectorAssembler(inputCols=cols,outputCol="features")

# Now let us use the transform method to transform our dataset
data=assembler.transform(new_data)

data.select("features",'Outcome').show(truncate=False)
```

```
+-----------------------------------------------------------------------+-------+
|features                                                               |Outcome|
+-----------------------------------------------------------------------+-------+
|[6.0,148.0,72.0,35.0,0.0,33.599998474121094,0.6269999742507935,50.0]   |1.0    |
|[1.0,85.0,66.0,29.0,0.0,26.600000381469727,0.3510000109672546 4,31.0]  |0.0    |
|[8.0,183.0,64.0,0.0,0.0,23.299999237060547,0.671999990940094,32.0]     |1.0    |
|[1.0,89.0,66.0,23.0,94.0,28.100000381469727,0.16699999570846558,21.0]  |0.0    |
|[0.0,137.0,40.0,35.0,168.0,43.099998474121094,2.2880001068115234,33.0] |1.0    |
|[5.0,116.0,74.0,0.0,0.0,25.600000381469727,0.20100000500679016,30.0]   |0.0    |
|[3.0,78.0,50.0,32.0,88.0,31.0,0.24799999594688416,26.0]                |1.0    |
|[10.0,115.0,0.0,0.0,0.0,35.29999923706055,0.13400000333786 01,29.0]    |0.0    |
|[2.0,197.0,70.0,45.0,543.0,30.5,0.15800000727176666,53.0]              |1.0    |
|[8.0,125.0,96.0,0.0,0.0,0.0,0.23199999332427979,54.0]                  |1.0    |
|[4.0,110.0,92.0,0.0,0.0,37.599998474121094,0.19099999964237213,30.0]   |0.0    |
|[10.0,168.0,74.0,0.0,0.0,38.0,0.5370000004768372,34.0]                 |1.0    |
|[10.0,139.0,80.0,0.0,0.0,27.100000381469727,1.440999984741211,57.0]    |0.0    |
|[1.0,189.0,60.0,23.0,846.0,30.100000381469727,0.39800000190734863,59.0]|1.0    |
|[5.0,166.0,72.0,19.0,175.0,25.799999237060547,0.5870000123977661,51.0] |1.0    |
|[7.0,100.0,0.0,0.0,0.0,30.0,0.4839999973773956 3,32.0]                 |1.0    |
|[0.0,118.0,84.0,47.0,230.0,45.79999923706055,0.5509999990463257,31.0]  |1.0    |
|[7.0,107.0,74.0,0.0,0.0,29.600000381469727,0.2540000081062317,31.0]    |1.0    |
|[1.0,103.0,30.0,38.0,83.0,43.29999923706055,0.18299999833106995,33.0]  |0.0    |
|[1.0,115.0,70.0,30.0,96.0,34.599998474121094,0.5289999842643738,32.0]  |1.0    |
+-----------------------------------------------------------------------+-------+
only showing top 20 rows
```

In [19]:

```
standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
data=standardscaler.fit(data).transform(data)
```

In [20]:
```python
data.select("features",'Outcome','Scaled_features').show(truncate=False)
```

```
+------------------------------------------------------------------------+-------+-----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------+
|features                                                                |Outcome|Scaled_features
|
+------------------------------------------------------------------------+-------+-----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------+
|[6.0,148.0,72.0,35.0,0.0,33.599998474121094,0.6269999742507935,50.0]    |1.0    |[1.7806383732194306,4.628960915766174,3.719813
8711154307,2.1940523222807116,0.0,4.261709202425419,1.8923810993699686,4.251616970894646]                     |
|[1.0,85.0,66.0,29.0,0.0,26.600000381469727,0.3510000109672546,31.0]     |0.0    |[0.29677306220323846,2.658524850271114,3.40982
93818558116,1.8179290670325896,0.0,3.373853320188119,1.0593713140527197,2.6360025219546803]                   |
|[8.0,183.0,64.0,0.0,0.0,23.299999237060547,0.671999990940094,32.0]      |1.0    |[2.374184497625907,5.723647618818986,3.306501
218769272,0.0,0.0,2.955292430788826,2.028197980632078,2.721034861372573]                                      |
|[1.0,89.0,66.0,23.0,94.0,28.100000381469727,0.16699999570846558,21.0]   |0.0    |[0.29677306220323846,2.78363190204857 8,3.40982
93818558116,1.4418058117844677,0.8156606685129459,3.564108203936454,0.5040313372439763,1.785679127775751]     |
|[0.0,137.0,40.0,35.0,168.0,43.099998474121094,2.2880001068115234,33.0]  |1.0    |[0.0,4.284916523378148,2.0665632617307947,2.19
40523222807116,1.4577765139380312,5.466656799498205,6.905531635244907,2.806067200790466]                      |
|[5.0,116.0,74.0,0.0,0.0,25.600000381469727,0.20100000500679016,30.0]    |0.0    |[1.4838653110161923,3.628104501546461,3.823142
034201971,0.0,0.0,3.247016731022563,0.6066485264255773,2.5509701825367874]                                    |
|[3.0,78.0,50.0,32.0,88.0,31.0,0.24799999594688416,26.0]                 |1.0    |[0.890319186609715 3,2.4395875096605515,2.58320
40771634937,2.0059906946566506,0.7635972215865877,3.9319342641322486,0.7485016335678398,2.210840824865216]    |
|[10.0,115.0,0.0,0.0,0.0,35.29999923706055,0.13400000333786 01,29.0]     |0.0    |[2.9677306220323847,3.596827738602095,0.0,0.0,
0.0,4.477331500775503,0.4044323509503849,2.4659378431188945]                                                  |
|[2.0,197.0,70.0,45.0,543.0,30.5,0.15800000727176666,53.0]               |1.0    |[0.5935461244064769,6.161522300040111,3.616485
708028891,2.8209244143609147,4.711741946835422,3.8685159695494704,0.4768680059655209,4.5067139891483246]      |
|[8.0,125.0,96.0,0.0,0.0,0.0,0.23199999332427979,54.0]                   |1.0    |[2.374184497625907 7,3.9095953680457556,4.95975
1828153908,0.0,0.0,0.0,0.0,0.7002111968910825,4.5917463285662174]                                             |
|[4.0,110.0,92.0,0.0,0.0,37.599998474121094,0.19099999964237213,30.0]    |0.0    |[1.1870922488129538,3.440443923880265,4.753095
5019808285,0.0,0.0,4.7690555590876444,0.5764669922591124,2.5509701825367874]                                  |
|[10.0,168.0,74.0,0.0,0.0,38.0,0.5370000004768372,34.0]                  |1.0    |[2.9677306220323847,5.254496174653495,3.823142
034201971,0.0,0.0,4.8197903882911435,1.6207475167416163,2.891099540208359]                                    |
|[10.0,139.0,80.0,0.0,0.0,27.100000381469727,1.440999984741211,57.0]     |0.0    |[2.9677306220323847,4.3474700492668 8,4.1331265
234615895,0.0,0.0,3.4372716147708973,4.349156694264776,4.846843346819896]                                     |
|[1.0,189.0,60.0,23.0,846.0,30.100000381469727,0.3980000019073486 3,59.0]|1.0    |[0.29677306220323846,5.911308196485182,3.09984
48925961926,1.4418058117844677,7.340946016616514,3.8177813822675666,1.201224421194982,5.016908025655682]      |
|[5.0,166.0,72.0,19.0,175.0,25.799999237060547,0.5870000123977661,51.0]  |1.0    |[1.4838653110161923,5.191942648764763,3.719813
8711154307,1.1910569749523863,1.5185172020187825,3.272383903702717,1.7716551425999747,4.336649310312539]      |
|[7.0,100.0,0.0,0.0,0.0,30.0,0.48399999737739563,32.0]                   |1.0    |[2.07741143542266 9,3.127676294436604,0.0,0.0,
0.0,3.8050976749666923,1.4607854621150949,2.721034861372573]                                                   |
|[0.0,118.0,84.0,47.0,230.0,45.79999923706055,0.5509999990463257,31.0]   |1.0    |[0.0,3.6906580274351932,4.33978284963467,2.946
2988327769555,1.9957654655103998,5.809115687013845,1.6630016375902872,2.6360025219546803]                     |
|[7.0,107.0,74.0,0.0,0.0,29.600000381469727,0.2540000081062317,31.0]     |1.0    |[2.07741143542266 9,3.3466136350471665,3.823142
034201971,0.0,0.0,3.7543630876847884,0.7666105810520987,2.6360025219546803]                                   |
|[1.0,103.0,30.0,38.0,83.0,43.29999923706055,0.18299999833106995,33.0]   |0.0    |[0.29677306220323846,3.221506583269702 5,1.5499
224462980963,2.3821139499047725,0.7202110158146225,5.4920242140999544,0.5523217739207337,2.806067200790466]|
|[1.0,115.0,70.0,30.0,96.0,34.599998474121094,0.5289999842643738,32.0]   |1.0    |[0.29677306220323846,3.596827738602095,3.61648
5708028891,1.88061627624061,0.833015150821732,4.388545791590976,1.596602253429271,2.721034861372573]          |
+------------------------------------------------------------------------+-------+-----------------------------------------------------------
-------------------------------------------------------------------------------------------------------------+
only showing top 20 rows
```

In [21]:
```python
assembled_data = data.select("Scaled_features","Outcome")
assembled_data.show()
```

```
+--------------------+-------+
|     Scaled_features|Outcome|
+--------------------+-------+
|[1.78063837321943...|    1.0|
|[0.29677306220323...|    0.0|
|[2.37418449762590...|    1.0|
|[0.29677306220323...|    0.0|
|[0.0,4.2849165233...|    1.0|
|[1.48386531101619...|    0.0|
|[0.89031918660971...|    1.0|
|[2.96773062203238...|    0.0|
|[0.59354612440647...|    1.0|
|[2.37418449762590...|    1.0|
|[1.18709224881295...|    0.0|
|[2.96773062203238...|    1.0|
|[2.96773062203238...|    0.0|
|[0.29677306220323...|    1.0|
|[1.48386531101619...|    1.0|
|[2.07741143542266...|    1.0|
|[0.0,3.6906580274...|    1.0|
|[2.07741143542266...|    1.0|
|[0.29677306220323...|    0.0|
|[0.29677306220323...|    1.0|
+--------------------+-------+
only showing top 20 rows
```

In [22]:
```python
train, test = assembled_data.randomSplit([0.7, 0.3])
```

In [23]:
```python
train.show()
```

```
+--------------------+-------+
|     Scaled_features|Outcome|
+--------------------+-------+
|(8,[0,1,6,7],[0.5...|    0.0|
|(8,[0,1,6,7],[0.5...|    0.0|
|(8,[0,1,6,7],[0.8...|    0.0|
|(8,[0,1,6,7],[1.7...|    0.0|
|(8,[0,1,6,7],[2.9...|    1.0|
|(8,[1,5,6,7],[3.0...|    0.0|
|(8,[1,5,6,7],[3.6...|    0.0|
|(8,[1,5,6,7],[4.3...|    1.0|
|(8,[1,5,6,7],[4.4...|    1.0|
|(8,[1,5,6,7],[4.5...|    1.0|
|(8,[1,5,6,7],[5.2...|    1.0|
|(8,[1,6,7],[2.940...|    0.0|
|[0.0,1.7827754878...|    0.0|
|[0.0,2.3144804578...|    0.0|
|[0.0,2.4395875096...|    0.0|
|[0.0,2.6272480873...|    0.0|
|[0.0,2.6898016132...|    0.0|
|[0.0,2.8461854279...|    0.0|
|[0.0,2.8461854279...|    0.0|
|[0.0,2.9087389538...|    0.0|
+--------------------+-------+
only showing top 20 rows
```

In [24]:
```python
test.show()
```

```
+--------------------+-------+
|     Scaled_features|Outcome|
+--------------------+-------+
|(8,[0,1,6,7],[2.0...|    0.0|
|(8,[1,5,6,7],[2.2...|    0.0|
|(8,[1,5,6,7],[3.7...|    1.0|
|(8,[1,5,6,7],[4.0...|    1.0|
|[0.0,2.0955431172...|    0.0|
|[0.0,2.6272480873...|    0.0|
|[0.0,2.9087389538...|    0.0|
|[0.0,2.9400157167...|    0.0|
|[0.0,3.1276762944...|    0.0|
|[0.0,3.1589530573...|    0.0|
|[0.0,3.1902298203...|    0.0|
|[0.0,3.1902298203...|    0.0|
|[0.0,3.1902298203...|    0.0|
|[0.0,3.2840601091...|    1.0|
|[0.0,3.2840601091...|    0.0|
|[0.0,3.3153368721...|    0.0|
|[0.0,3.3466136350...|    0.0|
|[0.0,3.5342742127...|    0.0|
|[0.0,3.5655509756...|    0.0|
|[0.0,3.7219347903...|    0.0|
+--------------------+-------+
only showing top 20 rows
```

Logistic Regression

In [25]:
```python
log_reg = LogisticRegression(labelCol="Outcome", featuresCol="Scaled_features",maxIter=40)
model=log_reg.fit(train)
```

In [26]:
```python
prediction_test=model.transform(test)
```

In [27]:
```python
prediction_test.show()
```

```
+--------------------+-------+--------------------+--------------------+----------+
|     Scaled_features|Outcome|       rawPrediction|         probability|prediction|
+--------------------+-------+--------------------+--------------------+----------+
|(8,[0,1,6,7],[2.0...|    0.0|[3.00934626047817...|[0.95299457813441...|       0.0|
|(8,[1,5,6,7],[2.2...|    0.0|[2.86249672911400...|[0.94596107132278...|       0.0|
|(8,[1,5,6,7],[3.7...|    1.0|[0.43302150915964...|[0.60659494563541...|       0.0|
|(8,[1,5,6,7],[4.0...|    1.0|[-1.0873447563720...|[0.25211860776150...|       1.0|
|[0.0,2.0955431172...|    0.0|[2.02100635901620...|[0.88298502870374...|       0.0|
|[0.0,2.6272480873...|    0.0|[2.21764386312331...|[0.90182278440525...|       0.0|
|[0.0,2.9087389538...|    0.0|[1.23263301888761...|[0.77427908187886...|       0.0|
|[0.0,2.9400157167...|    0.0|[1.47187569539789...|[0.81334231509700...|       0.0|
|[0.0,3.1276762944...|    0.0|[0.65635090074451...|[0.65844019316943...|       0.0|
|[0.0,3.1589530573...|    0.0|[2.85982177280004...|[0.94582416773803...|       0.0|
|[0.0,3.1902298203...|    0.0|[1.26717365409503...|[0.78025854021615...|       0.0|
|[0.0,3.1902298203...|    0.0|[2.19723997633928...|[0.90000138590173...|       0.0|
|[0.0,3.1902298203...|    0.0|[2.26711075930569...|[0.90611628955997...|       0.0|
|[0.0,3.2840601091...|    1.0|[1.57367137141972...|[0.82830636161183...|       0.0|
|[0.0,3.2840601091...|    0.0|[2.28199514129786...|[0.90737486630932...|       0.0|
|[0.0,3.3153368721...|    0.0|[1.18340640009529...|[0.76555973069936...|       0.0|
|[0.0,3.3466136350...|    0.0|[0.54445468871403...|[0.63284808202811...|       0.0|
|[0.0,3.5342742127...|    0.0|[1.58300582535699...|[0.82962979500114...|       0.0|
|[0.0,3.5655509756...|    0.0|[0.92099615367912...|[0.71524503539201...|       0.0|
|[0.0,3.7219347903...|    0.0|[1.00434322689320...|[0.73191165143244...|       0.0|
+--------------------+-------+--------------------+--------------------+----------+
only showing top 20 rows
```

In [19]:
```
prediction_test.select("Outcome","prediction").show(10)
```

```
+-------+----------+
|Outcome|prediction|
+-------+----------+
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       1.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    1.0|       0.0|
+-------+----------+
only showing top 10 rows
```

In [20]:
```
# Compute raw scores on the test set
predictionAndLabels = prediction_test.select("Outcome","prediction").rdd
```

In [21]:
```
predictionAndLabels.collect()
```

Out[21]:
```
[Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=1.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=1.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=1.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=1.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
 Row(Outcome=1.0, prediction=0.0),
 Row(Outcome=0.0, prediction=0.0),
```

In [22]:
```
metrics = BinaryClassificationMetrics(predictionAndLabels)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
C:\Users\anjal\anaconda3\lib\site-packages\pyspark\sql\context.py:125: FutureWarning: Deprecated in 3.0.0. Use SparkSession.bui
lder.getOrCreate() instead.
  warnings.warn(
```

```
Area under ROC = 0.7630208333333334
```

In [ ]:

In [28]:
```
evaluator = MulticlassClassificationEvaluator(labelCol="Outcome", predictionCol="prediction", metricName="accuracy")
accuracy_LR = evaluator.evaluate(prediction_test)
print ("Accuracy = " ,accuracy_LR)
```

```
Accuracy =  0.7860082304526749
```

NaiveBayes

In [29]:
```
naive_bayes = NaiveBayes(featuresCol='Scaled_features',labelCol='Outcome',smoothing=1.0)
```

In [30]:
```
model = naive_bayes.fit(train)
```

In [31]:
```
# select example rows to display.
prediction_test = model.transform(test)
```

In [32]:
```
prediction_test.show()
```

```
+--------------------+-------+--------------------+--------------------+----------+
|     Scaled_features|Outcome|       rawPrediction|         probability|prediction|
+--------------------+-------+--------------------+--------------------+----------+
|(8,[0,1,6,7],[2.0...|    0.0|[-17.913465045042...|[0.51471712239396...|       0.0|
|(8,[1,5,6,7],[2.2...|    0.0|[-14.945093079062...|[0.61304772443910...|       0.0|
|(8,[1,5,6,7],[3.7...|    1.0|[-17.727518454082...|[0.60898156364314...|       0.0|
|(8,[1,5,6,7],[4.0...|    1.0|[-21.774920893756...|[0.60370782413439...|       0.0|
|[0.0,2.0955431172...|    0.0|[-27.815736937756...|[0.77140623381313...|       0.0|
|[0.0,2.6272480873...|    0.0|[-30.276441952212...|[0.69731740037420...|       0.0|
|[0.0,2.9087389538...|    0.0|[-44.210915011055...|[0.74817864098683...|       0.0|
|[0.0,2.9400157167...|    0.0|[-33.517912998034...|[0.69230045040051...|       0.0|
|[0.0,3.1276762944...|    0.0|[-47.780870074750...|[0.72812810729529...|       0.0|
|[0.0,3.1589530573...|    0.0|[-21.601004666834...|[0.70885996825772...|       0.0|
|[0.0,3.1902298203...|    0.0|[-36.109334899503...|[0.71123489140353...|       0.0|
|[0.0,3.1902298203...|    0.0|[-33.878582573975...|[0.73064702441743...|       0.0|
|[0.0,3.1902298203...|    0.0|[-34.225051895099...|[0.68586842051537...|       0.0|
|[0.0,3.2840601091...|    1.0|[-33.906667660597...|[0.73762889361350...|       0.0|
|[0.0,3.2840601091...|    0.0|[-27.874920063310...|[0.76761285064617...|       0.0|
|[0.0,3.3153368721...|    0.0|[-38.191889234909...|[0.65903992353788...|       0.0|
|[0.0,3.3466136350...|    0.0|[-30.217174573846...|[0.72486341052346...|       0.0|
|[0.0,3.5342742127...|    0.0|[-31.755486086323...|[0.72470730298884...|       0.0|
|[0.0,3.5655509756...|    0.0|[-41.303986218189...|[0.60598558671805...|       0.0|
|[0.0,3.7219347903...|    0.0|[-29.556753428990...|[0.74591685075410...|       0.0|
+--------------------+-------+--------------------+--------------------+----------+
only showing top 20 rows
```

In [28]:
```
prediction_test.select("Outcome","prediction").show(10)
```

```
+-------+----------+
|Outcome|prediction|
+-------+----------+
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    1.0|       0.0|
+-------+----------+
only showing top 10 rows
```

In [29]:
```
predictionAndLabels = prediction_test.select("Outcome","prediction").rdd
```

In [30]:
```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(labelCol="Outcome", predictionCol="prediction", metricName="accuracy")
accuracy_NB = evaluator.evaluate(prediction_test)
```

In [31]:
```
print ("Accuracy",accuracy_NB)
```

```
Accuracy 0.6637931034482759
```

In [32]:
```
metrics = BinaryClassificationMetrics(predictionAndLabels)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
Area under ROC = 0.665151515151515
```

GBTClassifier

In [33]:
```
gradient_boost_class = GBTClassifier(labelCol="Outcome", featuresCol="Scaled_features")
```

In [34]:
```
model = gradient_boost_class.fit(train)
```

In [35]:
```
prediction_test = model.transform(test)
```

In [36]:
```
prediction_test.show()
```

```
+--------------------+-------+--------------------+--------------------+----------+
|     Scaled_features|Outcome|       rawPrediction|         probability|prediction|
+--------------------+-------+--------------------+--------------------+----------+
|(8,[0,1,6,7],[0.5...|    0.0|[1.44568213655613...|[0.94741788737785...|       0.0|
|(8,[0,1,6,7],[0.8...|    0.0|[1.44568213655613...|[0.94741788737785...|       0.0|
|(8,[1,5,6,7],[3.6...|    0.0|[-0.2717930506072...|[0.36735375875999...|       1.0|
|(8,[1,6,7],[2.940...|    0.0|[1.50198599908887...|[0.95275324620783...|       0.0|
|[0.0,2.3144804578...|    0.0|[1.50522683353396...|[0.95304416054471...|       0.0|
|[0.0,2.6898016132...|    0.0|[1.09956867281198...|[0.90017201748161...|       0.0|
|[0.0,2.9087389538...|    0.0|[1.57134073008306...|[0.95861938047406...|       0.0|
|[0.0,2.9087389538...|    0.0|[1.35099731931943...|[0.93714424045724...|       0.0|
|[0.0,2.9712924797...|    0.0|[1.31419458311707...|[0.93266645995597...|       0.0|
|[0.0,2.9712924797...|    1.0|[1.04792819086640...|[0.89049978823396...|       0.0|
|[0.0,3.0651227685...|    0.0|[1.60420294495114...|[0.96114938322129...|       0.0|
|[0.0,3.2527833462...|    1.0|[1.15156570570070...|[0.90913604928283...|       0.0|
|[0.0,3.2840601091...|    0.0|[1.27112161713480...|[0.92705067680045...|       0.0|
|[0.0,3.2840601091...|    1.0|[0.46267033234851...|[0.71612905203861...|       0.0|
|[0.0,3.3153368721...|    0.0|[1.01285658080524...|[0.88347047119139...|       0.0|
|[0.0,3.3466136350...|    0.0|[1.60420294495114...|[0.96114938322129...|       0.0|
|[0.0,3.4717206868...|    0.0|[1.57771527148241...|[0.95912216785821...|       0.0|
|[0.0,3.5342742127...|    1.0|[1.36992895918271...|[0.93933800107683...|       0.0|
|[0.0,3.8470418421...|    0.0|[0.96996381183291...|[0.87434419197093...|       0.0|
|[0.0,4.0347024198...|    0.0|[-0.0345097970698...|[0.48275194797327...|       1.0|
+--------------------+-------+--------------------+--------------------+----------+
only showing top 20 rows
```

In [37]:
```
prediction_test.select("Outcome","prediction").show(10)
```

```
+-------+----------+
|Outcome|prediction|
+-------+----------+
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       1.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    1.0|       0.0|
+-------+----------+
only showing top 10 rows
```

In [38]:
```
predictionAndLabels = prediction_test.select("Outcome","prediction").rdd
```

In [39]:
```
metrics = BinaryClassificationMetrics(predictionAndLabels)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
Area under ROC = 0.7170138888888888
```

In [40]:
```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator( labelCol="Outcome", predictionCol="prediction", metricName="accuracy")
accuracy_GBT = evaluator.evaluate(prediction_test)
```

In [41]:
```
print ("Accuracy",accuracy_GBT)
```

```
Accuracy 0.7413793103448276
```

RandomForestClassifier

In [42]:
```
random_forest_classifier = RandomForestClassifier(labelCol="Outcome", featuresCol="Scaled_features", numTrees=40)
```

In [43]:
```
model = random_forest_classifier.fit(train)
```

In [44]:
```
prediction_test = model.transform(test)
```

In [45]:
```
prediction_test.show()
```

```
+--------------------+-------+--------------------+--------------------+----------+
|     Scaled_features|Outcome|       rawPrediction|         probability|prediction|
+--------------------+-------+--------------------+--------------------+----------+
|(8,[0,1,6,7],[0.5...|    0.0|[38.4902255771088...|[0.96225563942772...|       0.0|
|(8,[0,1,6,7],[0.8...|    0.0|[38.2809672738907...|[0.95702418184726...|       0.0|
|(8,[1,5,6,7],[3.6...|    0.0|[19.6691676645421...|[0.49172919161355...|       1.0|
|(8,[1,6,7],[2.940...|    0.0|[38.1018706800663...|[0.95254676700165...|       0.0|
|[0.0,2.3144804578...|    0.0|[37.7177580420233...|[0.94294395105058...|       0.0|
|[0.0,2.6898016132...|    0.0|[34.5139232462476...|[0.86284808115619...|       0.0|
|[0.0,2.9087389538...|    0.0|[37.7177580420233...|[0.94294395105058...|       0.0|
|[0.0,2.9087389538...|    0.0|[25.2310398131261...|[0.63077599532815...|       0.0|
|[0.0,2.9712924797...|    0.0|[33.0527424559379...|[0.82631856139844...|       0.0|
|[0.0,2.9712924797...|    1.0|[35.9363678820332...|[0.89840919705083...|       0.0|
|[0.0,3.0651227685...|    0.0|[38.3197183345369...|[0.95799295836342...|       0.0|
|[0.0,3.2527833462...|    1.0|[35.3244895789060...|[0.88311223947265...|       0.0|
|[0.0,3.2840601091...|    0.0|[33.8725146472889...|[0.84681286618222...|       0.0|
|[0.0,3.2840601091...|    1.0|[32.8643056727631...|[0.82160764181907...|       0.0|
|[0.0,3.3153368721...|    0.0|[33.0552269501800...|[0.82638067375450...|       0.0|
|[0.0,3.3466136350...|    0.0|[38.2721347172117...|[0.95680336793029...|       0.0|
|[0.0,3.4717206868...|    0.0|[34.0701464003202...|[0.85175366000800...|       0.0|
|[0.0,3.5342742127...|    1.0|[34.3960493803103...|[0.85990123450776...|       0.0|
|[0.0,3.8470418421...|    0.0|[26.3864734907176...|[0.65966183726794...|       0.0|
|[0.0,4.0347024198...|    0.0|[17.2623736099806...|[0.43155934024951...|       1.0|
+--------------------+-------+--------------------+--------------------+----------+
only showing top 20 rows
```

In [46]:
```
prediction_test.select("Outcome","prediction").show(10)
```

```
+-------+----------+
|Outcome|prediction|
+-------+----------+
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       1.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    0.0|       0.0|
|    1.0|       0.0|
+-------+----------+
only showing top 10 rows
```

In [47]:
```
predictionAndLabels = prediction_test.select("Outcome","prediction").rdd
```

In [48]:
```
metrics = BinaryClassificationMetrics(predictionAndLabels)

# Area under ROC curve
print("Area under ROC = %s" % metrics.areaUnderROC)
```

```
Area under ROC = 0.735632183908046
```

In [49]:
```
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator( labelCol="Outcome", predictionCol="prediction", metricName="accuracy")
accuracy_RF= evaluator.evaluate(prediction_test)
```

In [50]:
```
print ("Accuracy",accuracy_RF)
```

```
Accuracy 0.75
```

In [51]:
```
print("Accuracy of GBT : ",accuracy_GBT)
print("Accuracy of LR : ",accuracy_LR)
print("Accuracy of NB : ",accuracy_NB)
print("Accuracy of RF : ",accuracy_RF)
```

```
Accuracy of GBT :  0.7413793103448276
Accuracy of LR :  0.7758620689655172
Accuracy of NB :  0.6637931034482759
Accuracy of RF :  0.75
```