**BDA Mini-Project (Phase-II)**

**Topic : Stroke Prediction using Apache Spark**

**Abstract:**

Stroke is the second leading cause of death worldwide and remains an important health burden both for the individuals and for the national healthcare systems. Potentially modifiable risk factors for stroke include hypertension, cardiac disease, diabetes, and dysregulation of glucose metabolism, atrial fibrillation, and lifestyle factors. The primary factors are age, gender, work conditions, bmi, etc. Data for stroke prediction is available here : https://www.kaggle.com/asaumya/healthcare-dataset-stroke-data. Using big data techniques, one can successfully process and predict the probability whether a person will get a stroke or not.

**About the dataset:**

The goal is to predict the stroke probability using the given information of patients. The primary factors are age, gender, work conditions, BMI, etc. It is a classification problem, A healthcare dataset containing stroke information has been used. The dataset schema is shown below.

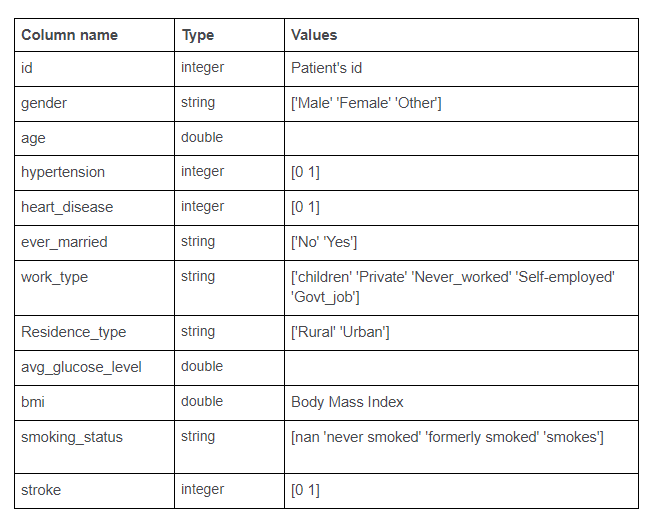


Fig 1: Dataset schema

The size of the train dataset is (43400 X 12) and that of the test dataset is (18601 X 11).

**Apache Spark**

Spark is a general-purpose distributed data processing engine that is suitable for use in a wide range of circumstances. On top of the Spark core data processing engine, there are libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application. Programming languages supported by Spark include: Java, Python, Scala, and R. Application developers and data scientists incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, IoT, or financial systems, and machine learning tasks.

Since its release, Apache Spark, the unified analytics engine, has seen rapid adoption by enterprises across a wide range of industries. Internet powerhouses such as Netflix, Yahoo, and eBay have deployed Spark at massive scale, collectively processing multiple petabytes of data on clusters of over 8,000 nodes. It has quickly become the largest open source community in big data, with over 1000 contributors from 250+ organizations.

**Benefits of Using Spark:**

**Speed**

Engineered from the bottom-up for performance, Spark can be 100x faster than Hadoop for large scale data processing by exploiting in memory computing and other optimizations. Spark is also fast when data is stored on disk, and currently holds the world record for large-scale on-disk sorting.

**Ease of Use**

Spark has easy-to-use APIs for operating on large datasets. This includes a collection of over 100 operators for transforming data and familiar data frame APIs for manipulating semi-structured data.

**A Unified Engine**

Spark comes packaged with higher-level libraries, including support for SQL queries, streaming data, machine learning and graph processing. These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows.

**Resilient Distributed Datasets (RDDs):**

RDDs are the building blocks of any Spark application. RDDs Stands for:

Resilient: It is fault tolerant and is capable of rebuilding data on failure.

Distributed: Data is distributed among the multiple nodes in a cluster.

Dataset: Collection of partitioned data with values.

It is a layer of abstracted data over the distributed collection. It is immutable in nature and follows lazy transformations.

With RDDs, you can perform two types of operations:

Transformations: These operations are applied to create a new RDD.

Actions: These operations are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

**pySpark**

PySpark is the collaboration of Apache Spark and Python.

It is a Python API for Spark that lets you harness the simplicity of Python and the power of Apache Spark in order to tame Big Data.

Dataframe in PySpark is the distributed collection of structured or semi-structured data. This data in Dataframe is stored in rows under named columns which is similar to the relational database tables or excel sheets.

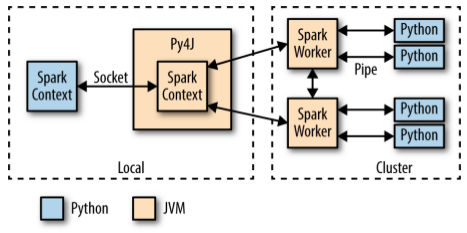


Fig 2: Pyspark Architecture

**Mllib :**

In PySpark, machine learning is facilitated by a Python library called MLlib (Machine Learning Library). It is nothing but a wrapper over PySpark Core that performs data analysis using machine-learning algorithms like classification, clustering, linear regression and few more.

One of the great features of machine learning with PySpark is that it works on distributed systems and is highly scalable.

MLlib exposes three core machine learning functionalities with PySpark:

Data Preparation: It provides various features like extraction, transformation, selection, hashing etc.

Machine Learning Algorithms: It avails some popular and advanced regression, classification, and clustering algorithms for machine learning.

Utilities: It has statistical methods such as chi-square testing, descriptive statistics, linear algebra and model evaluation methods.

**Data Preprocessing:**

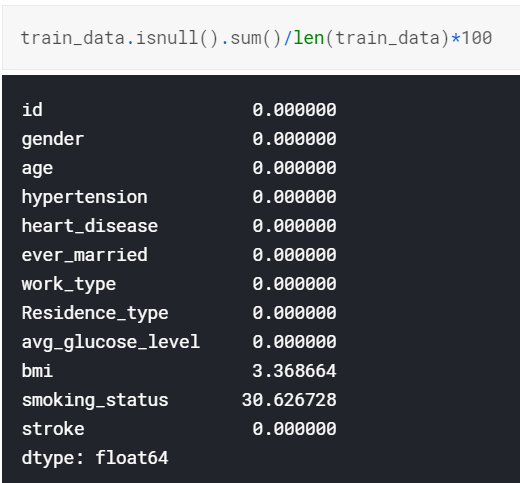


Fig 3 : Missing values in training data

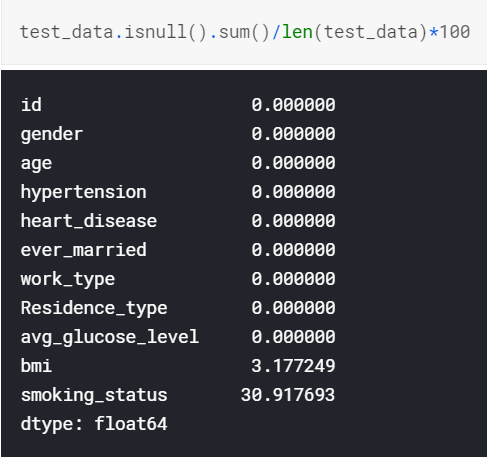
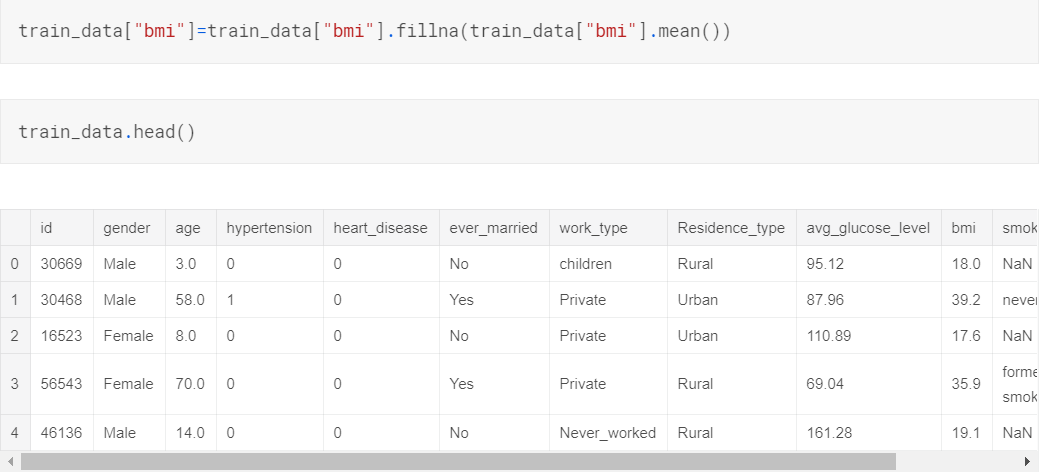
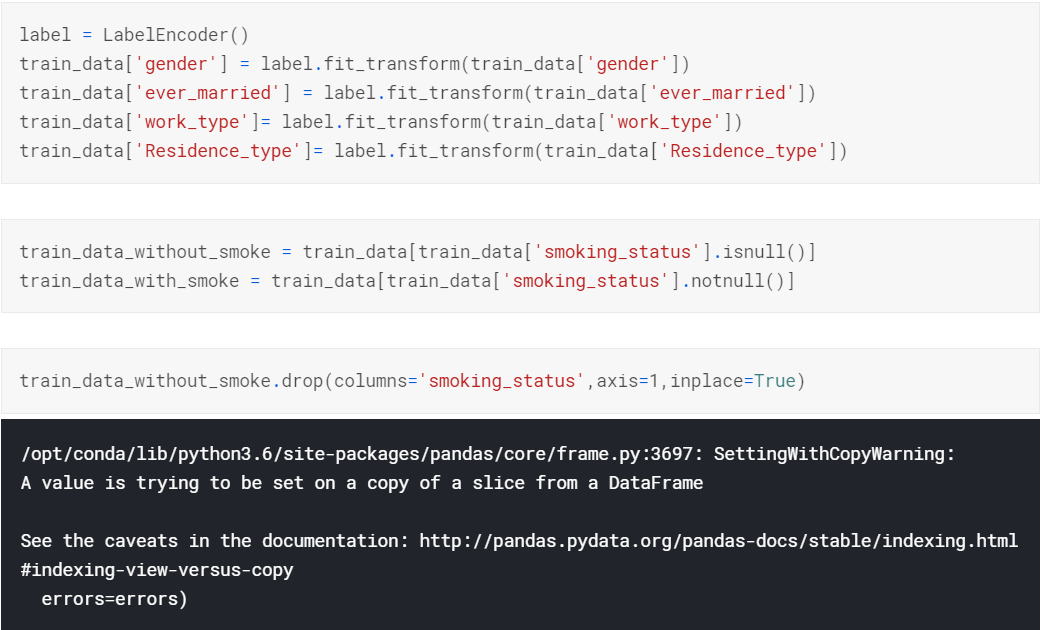


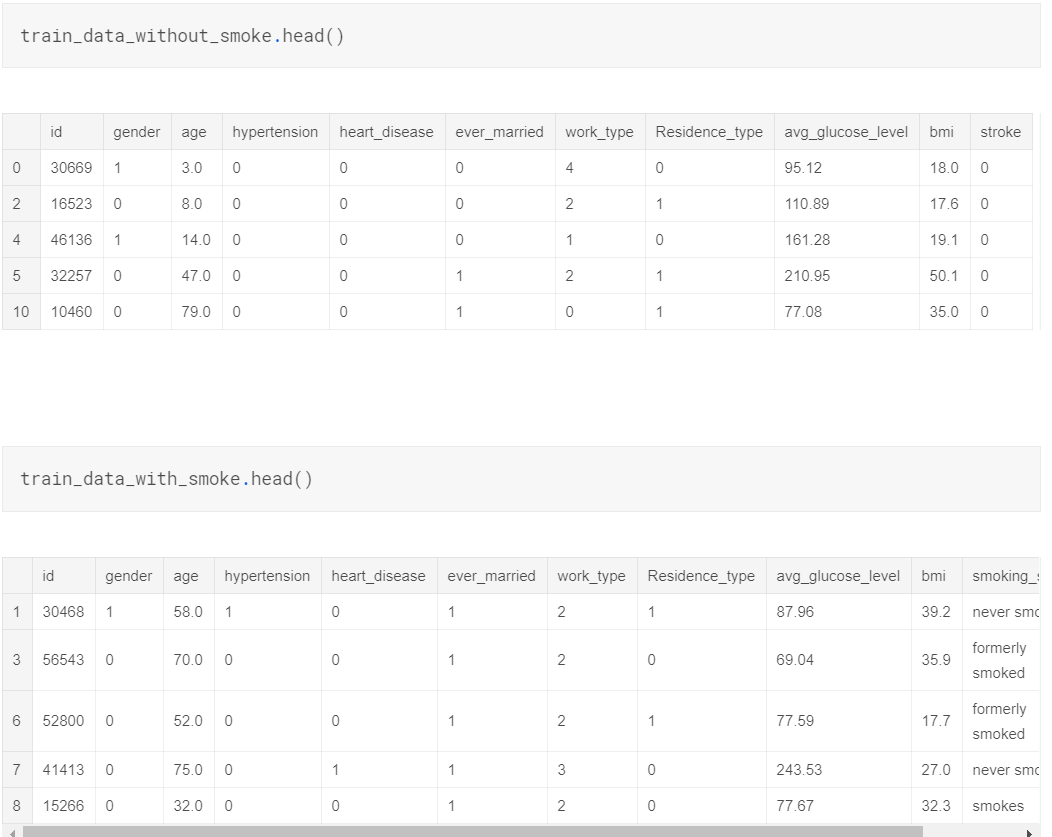
Fig 4 : Missing values in test data

This data is imbalanced and has considerable amount of missing values. The percent amount of missing data in both datasets is shown in the images above. The missing BMI values were filled with mean BMI value.

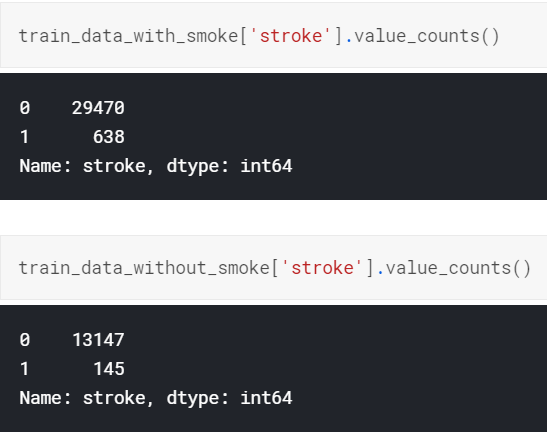


A large number of records do not contain value for ‘smoking\_status’ field. These records were separated from the datasets and thus 2 types of datasets were obtained, one containing value for ‘smoking\_status’ and the other dataset without ‘smoking\_status ’parameter.





Moreover, the dataset is highly imbalanced. It can be seen in the image below.



There is a high possibility that the classifier will predict no stroke for all data. So ROSE method is used to make the data more balanced by generating artificial data to make the set more balanced.



**ML Algorithms:**

There are a number of ML algorithms to choose from. Based on the properties of data we have chosen the following algorithms :

Decision Tree Classifier, Random Forest Classifier and Gradient Boosted Tree Classifier.

These are tree based algorithms which work well with categorical data and do not require feature scaling.

**Decision Tree:**

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

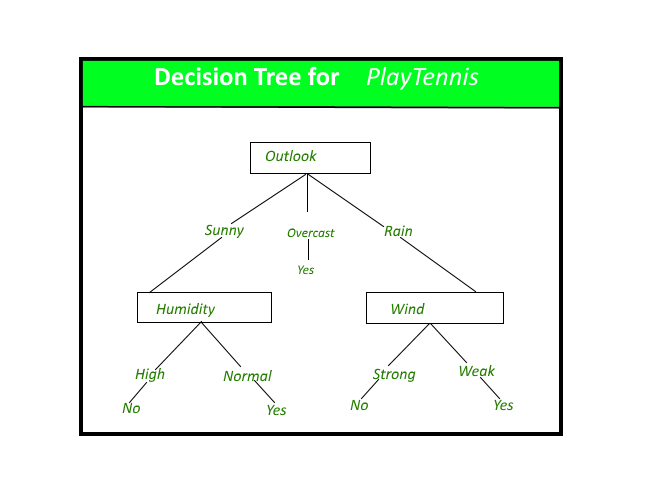


Fig 5 : Example of Decision Tree Classifier

**Random Forest :**

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction (see figure below).

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

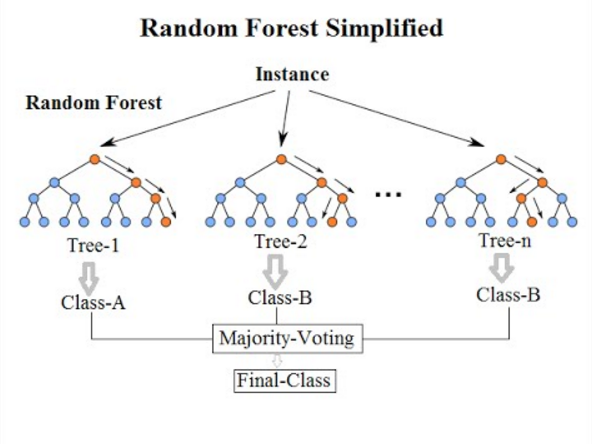


Fig 6: Example of Random Forest Classifier

**Gradient Boosted Tree Classifier:**

Boosting is an ensemble technique in which the predictors are not made independently, but sequentially.

This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the bootstrap process, but based on the error). The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc.

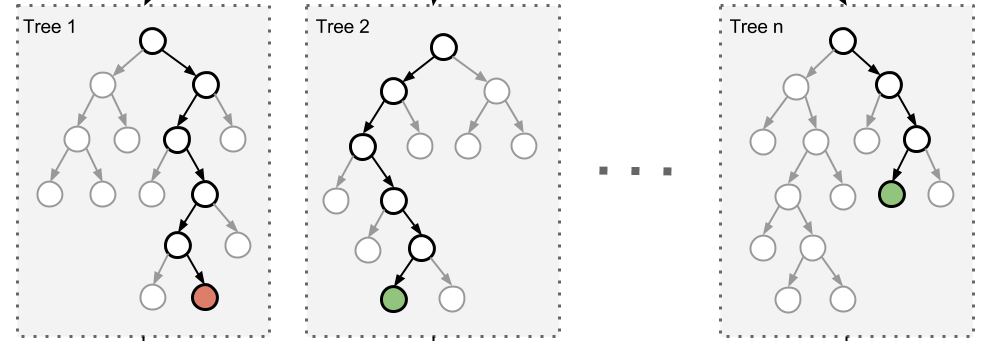


Fig 7 : Gradient Boosted Tree Classification

**Code and output:**

**a. Loading file to hdfs :**

hdfs dfs -mkdir /user

hdfs dfs -mkdir /user/hemal

hdfs dfs -copyFromLocal ~/Desktop/stroke\ data/\* /user/hemal

hdfs dfs -copyFromLocal ~/Desktop/stroke\_data/\* /user/hemal

hdfs dfs –chmod –R 755 /user/hemal

hdfs dfs -chmod -R 755 /user/hemal

**b. Code file: trial.py**

from pyspark.ml.feature import VectorAssembler

from pyspark.sql import SparkSession

from pyspark.ml.evaluation import BinaryClassificationEvaluator

from pyspark.ml.classification import GBTClassifier, RandomForestClassifier, DecisionTreeClassifier

spark = SparkSession.builder.appName('stroke').getOrCreate()

train\_data = spark.read.csv("/user/hemal/train.csv",header=True,inferSchema=True)

test\_data = spark.read.csv("/user/hemal/test.csv",header=True,inferSchema=True)

assembler=VectorAssembler(

inputCols=[

'gender','age','hypertension',

'heart\_disease','ever\_married','work\_type',

'residence\_type', 'avg\_glucose\_level', 'bmi', 'smoking\_status'

],

outputCol='features')

train=assembler.transform(train\_data)

test= assembler.transform(test\_data)

train.printSchema()

test.printSchema()

final\_train= train.select('features','stroke')

final\_test = test.select('features', 'stroke')

evaluator = BinaryClassificationEvaluator(labelCol = 'stroke')

dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'stroke', maxDepth = 10)

dtModel = dt.fit(final\_train)

DTpredictions = dtModel.transform(final\_test)

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'stroke', numTrees=50)

rfModel = rf.fit(train)

RFpredictions = rfModel.transform(test)

gbt = GBTClassifier(featuresCol = 'features', maxIter=50, labelCol = 'stroke')

gbtModel = gbt.fit(train)

GBTpredictions = gbtModel.transform(test)

sample = open('samplefile2.txt', 'w')

print("1. Evaluations For Decision Tree Classifier\n", file=sample)

DTpredictions.select('stroke', 'rawPrediction', 'prediction', 'probability').show(10)

print("Test Area Under ROC: " + str(evaluator.evaluate(DTpredictions, {evaluator.metricName: "areaUnderROC"})), file=sample)

print("Test Area Under Precision Recall Curve: " + str(evaluator.evaluate(DTpredictions, {evaluator.metricName: 'areaUnderPR'})), file=sample)

print(", file=sample")

print("2. Evaluations For Random Forest Classifier\n", file=sample)

RFpredictions.select('id', 'stroke', 'rawPrediction', 'prediction', 'probability').show(10)

print("Test Area Under ROC: " + str(evaluator.evaluate(RFpredictions, {evaluator.metricName: "areaUnderROC"})), file=sample)

print("Test Area Under Precision Recall Curve: " + str(evaluator.evaluate(RFpredictions, {evaluator.metricName: 'areaUnderPR'})), file=sample)

print("", file=sample)

print("3. Evaluations For Gradient Boosted Tree\n", file=sample)

GBTpredictions.select('id', 'stroke', 'rawPrediction', 'prediction', 'probability').show(10)

print("Test Area Under ROC: " + str(evaluator.evaluate(GBTpredictions, {evaluator.metricName: "areaUnderROC"})),file=sample)

print("Test Area Under Precision Recall Curve: " + str(evaluator.evaluate(GBTpredictions, {evaluator.metricName: 'areaUnderPR'})), file=sample)

print("", file=sample)

sample.close()

**c. Execution command :**

$SPARK\_HOME/bin/spark-submit trial.py

**d. Output:**

"""

root

|-- id: integer (nullable = true)

|-- gender: integer (nullable = true)

|-- age: integer (nullable = true)

|-- hypertension: integer (nullable = true)

|-- heart\_disease: integer (nullable = true)

|-- ever\_married: integer (nullable = true)

|-- work\_type: integer (nullable = true)

|-- residence\_type: integer (nullable = true)

|-- avg\_glucose\_level: double (nullable = true)

|-- bmi: double (nullable = true)

|-- smoking\_status: integer (nullable = true)

|-- stroke: integer (nullable = true)

|-- features: vector (nullable = true)

root

|-- id: integer (nullable = true)

|-- gender: integer (nullable = true)

|-- age: integer (nullable = true)

|-- hypertension: integer (nullable = true)

|-- heart\_disease: integer (nullable = true)

|-- ever\_married: integer (nullable = true)

|-- work\_type: integer (nullable = true)

|-- residence\_type: integer (nullable = true)

|-- avg\_glucose\_level: double (nullable = true)

|-- bmi: double (nullable = true)

|-- smoking\_status: integer (nullable = true)

|-- stroke: integer (nullable = true)

|-- features: vector (nullable = true)

1. Evaluations For Decision Tree Classifier (depth 5)

Test Area Under ROC: 0.768139651656334

Test Area Under Precision Recall Curve: 0.7461299883929062

2. Evaluations For Random Forest Classifier

Test Area Under ROC: 0.8417353412425762 (10 trees)

Test Area Under Precision Recall Curve: 0.8022116726992913

3. Evaluations For Gradient Boosted Tree (15 iterations)

Test Area Under ROC: 0.8749491766436319

Test Area Under Precision Recall Curve: 0.8383484936234599

1. Evaluations For Decision Tree Classifier (depth 10)

Test Area Under ROC: 0.8436204375123288

Test Area Under Precision Recall Curve: 0.7738773601201463

2. Evaluations For Random Forest Classifier (50 trees)

Test Area Under ROC: 0.8442934321841468

Test Area Under Precision Recall Curve: 0.8087893423366714

3. Evaluations For Gradient Boosted Tree (50 iterations)

Test Area Under ROC: 0.9087341656505779

Test Area Under Precision Recall Curve: 0.8756167110974958

"""

**Screenshots:**

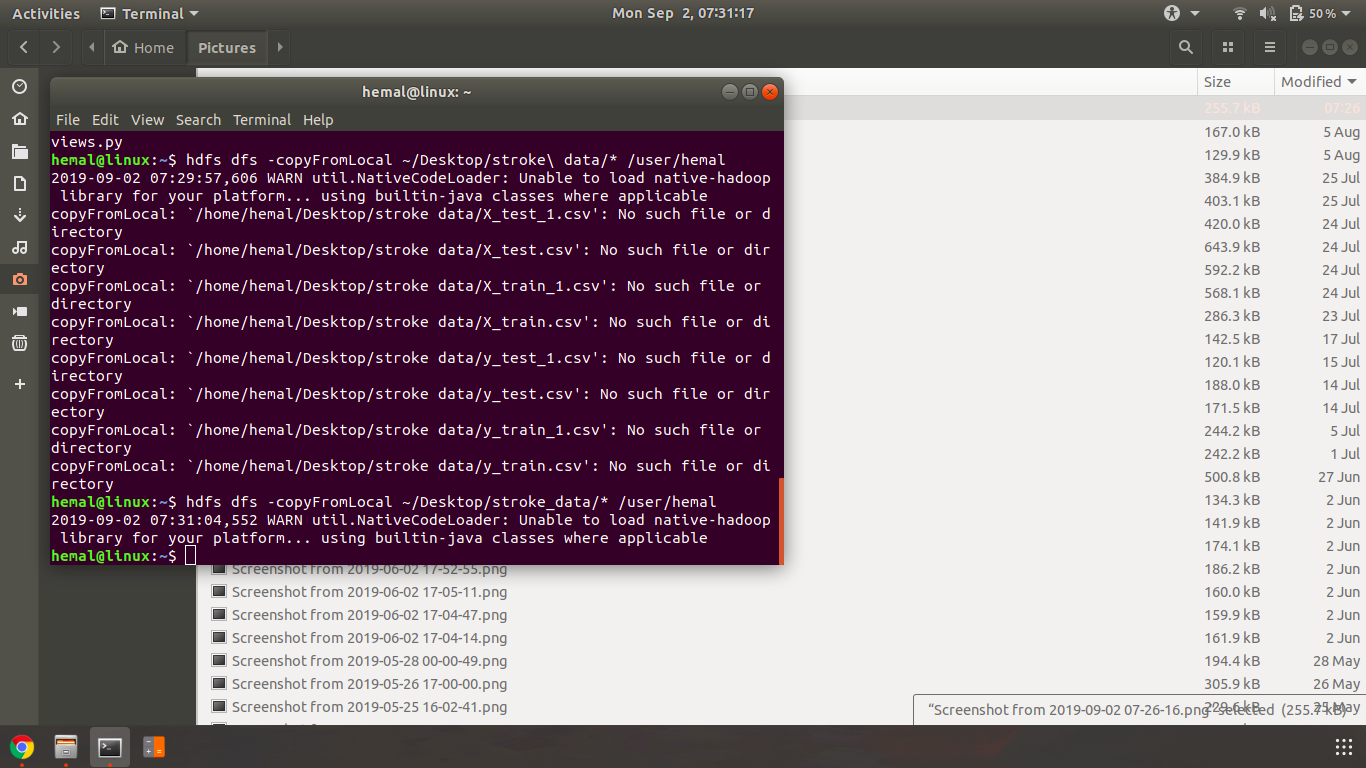


Fig 8 : loading files to hdfs using hdfs commands

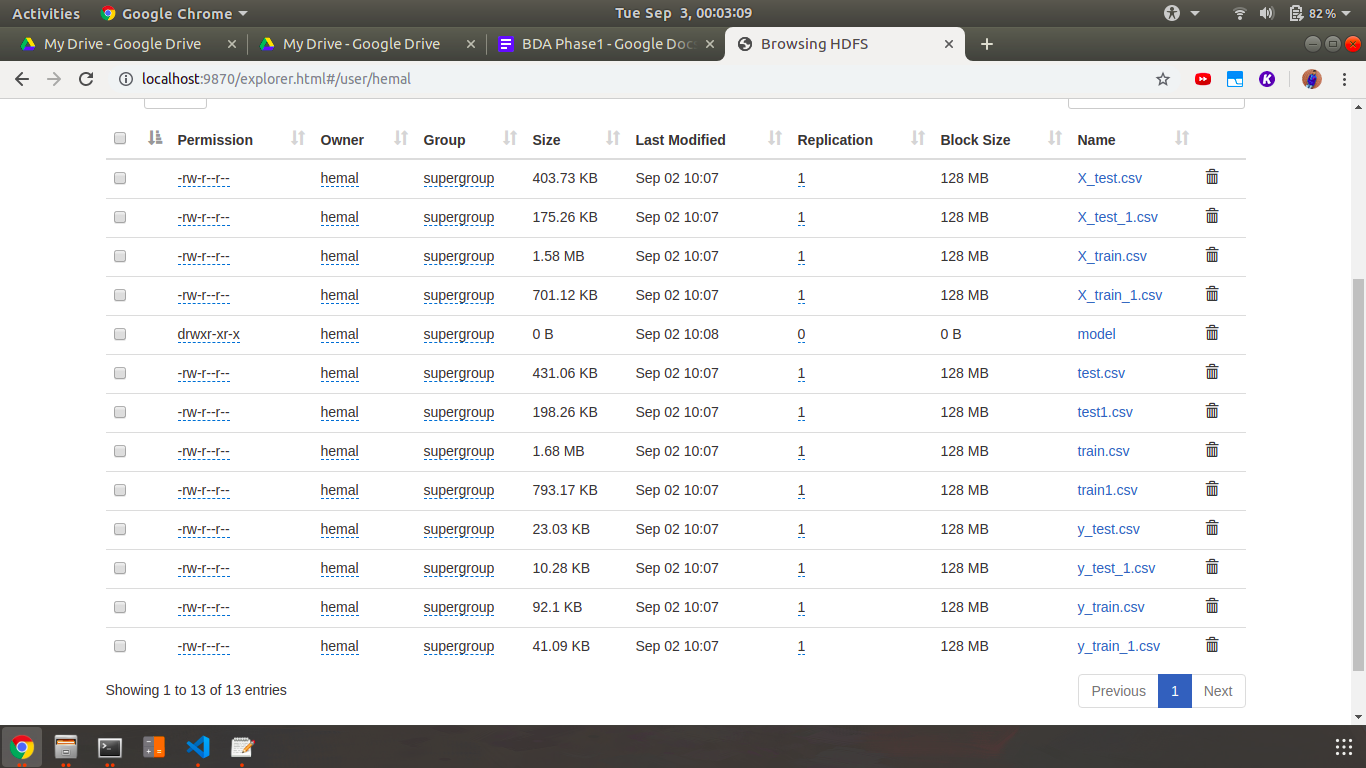


Fig 9. : Training and testing files loaded in hdfs

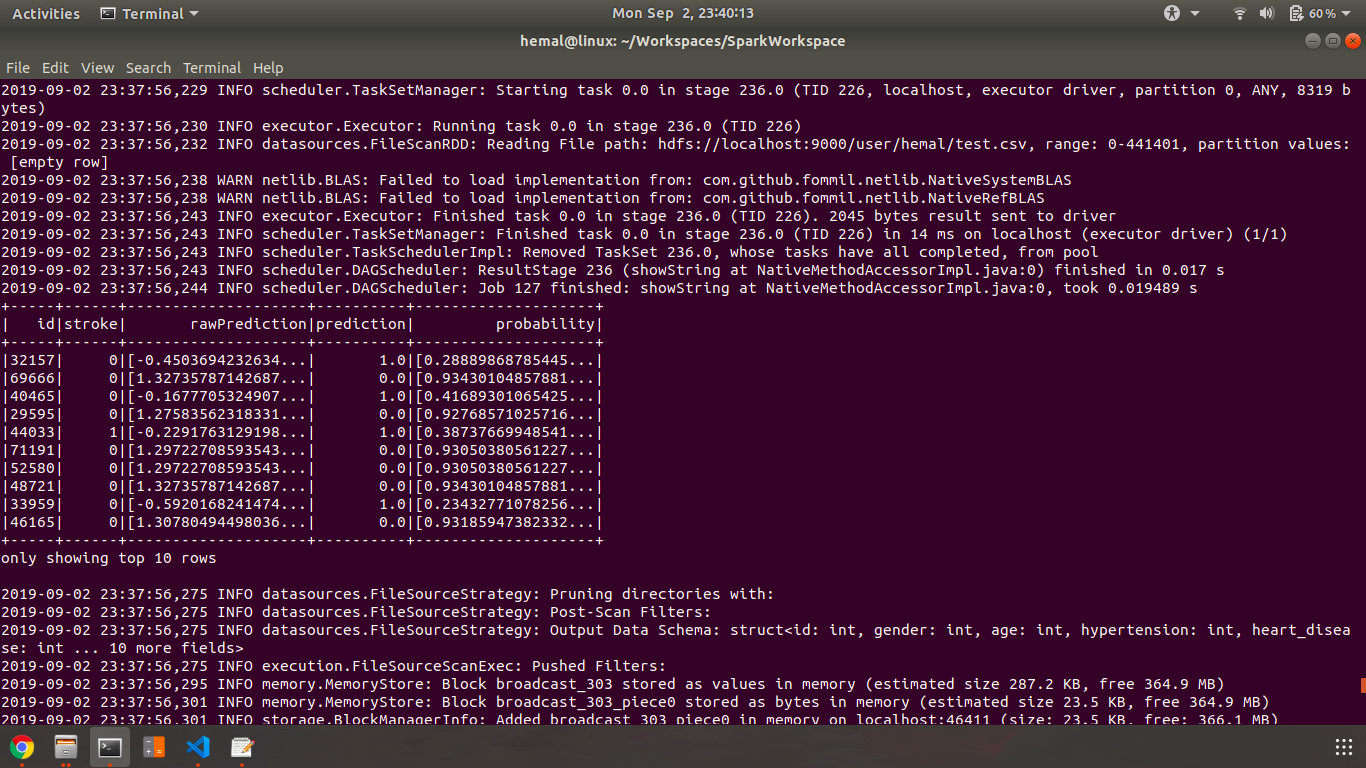


Fig 10. : Sample output for Decision Tree Classifier

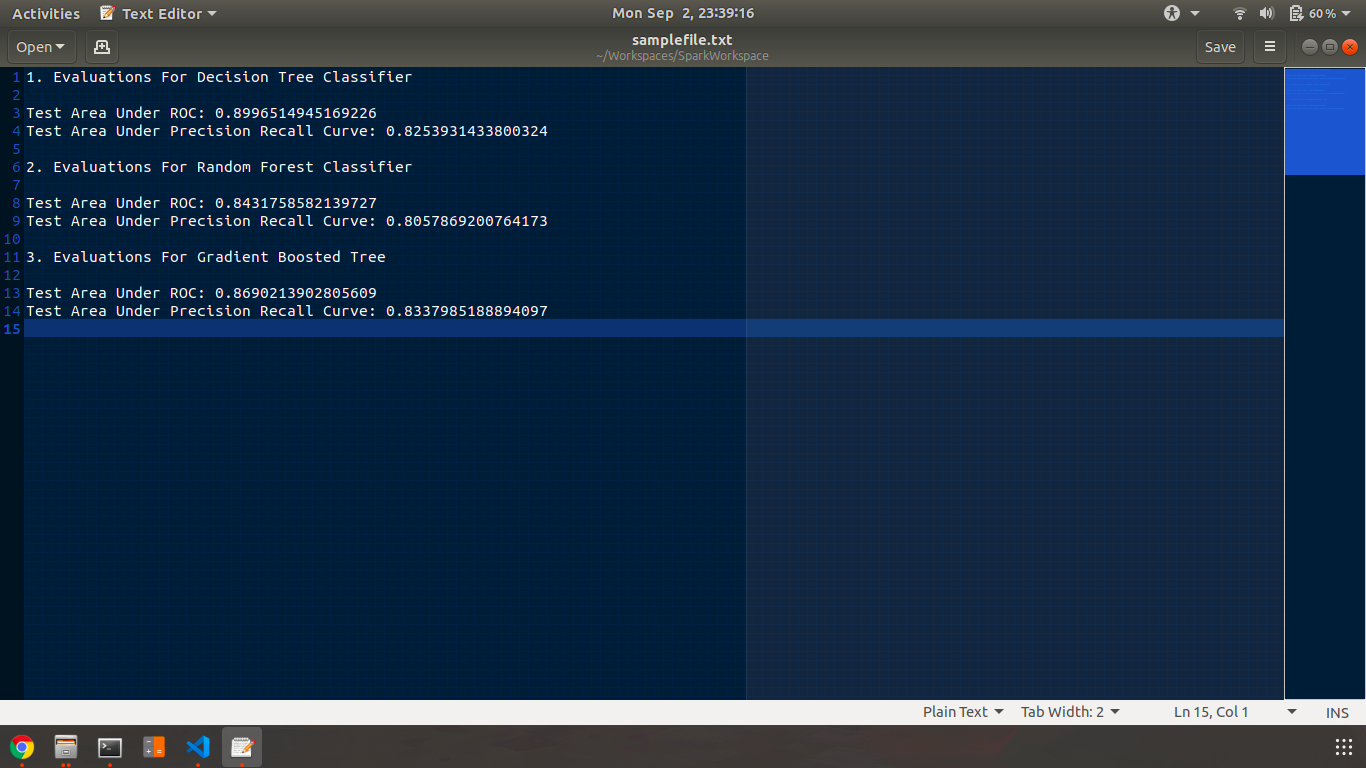


Fig 11. : ROC and Precision recall curves in file : samplefile.txt

**Conclusion :**

In this project we understood Apache Spark architecture. We used HDFS to store the training and testing files after preprocessing. We used pySpark and Mllib to apply Machine Learning Algorithms like Decision Tree, Random Forests and Gradient boosted trees on our dataset. We found the area under the ROC curve and PR curve in order to evaluate our models. Thus we were able to predict the stroke with over 90% accuracy.