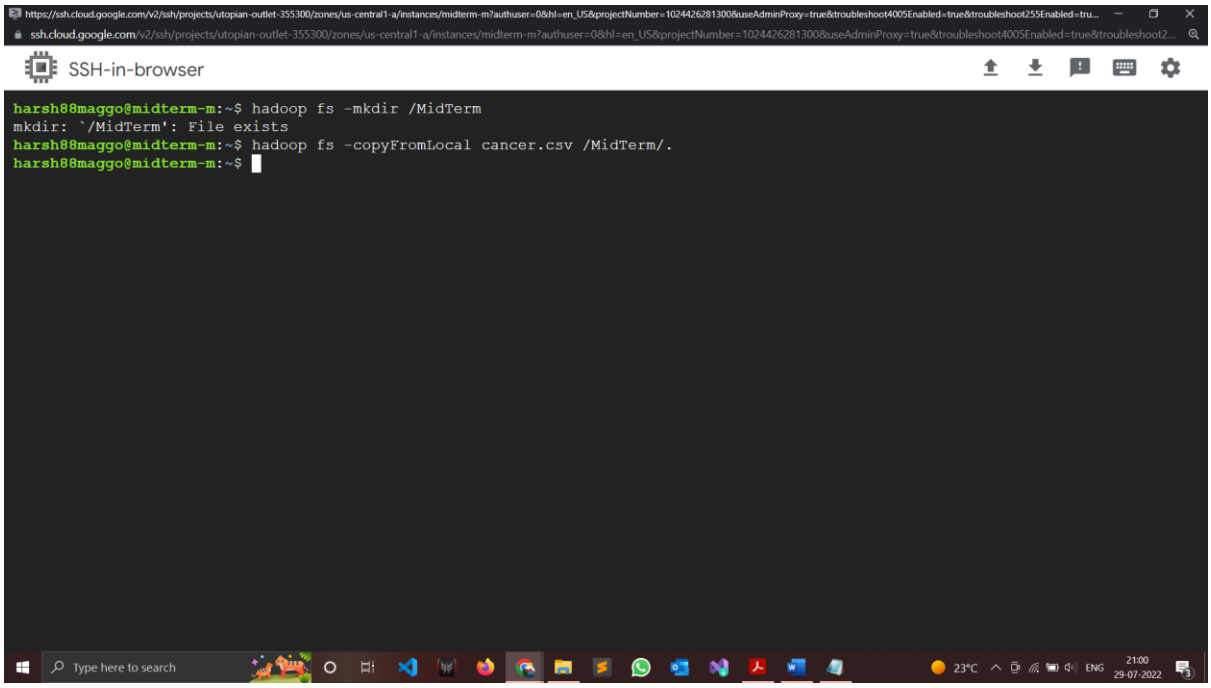
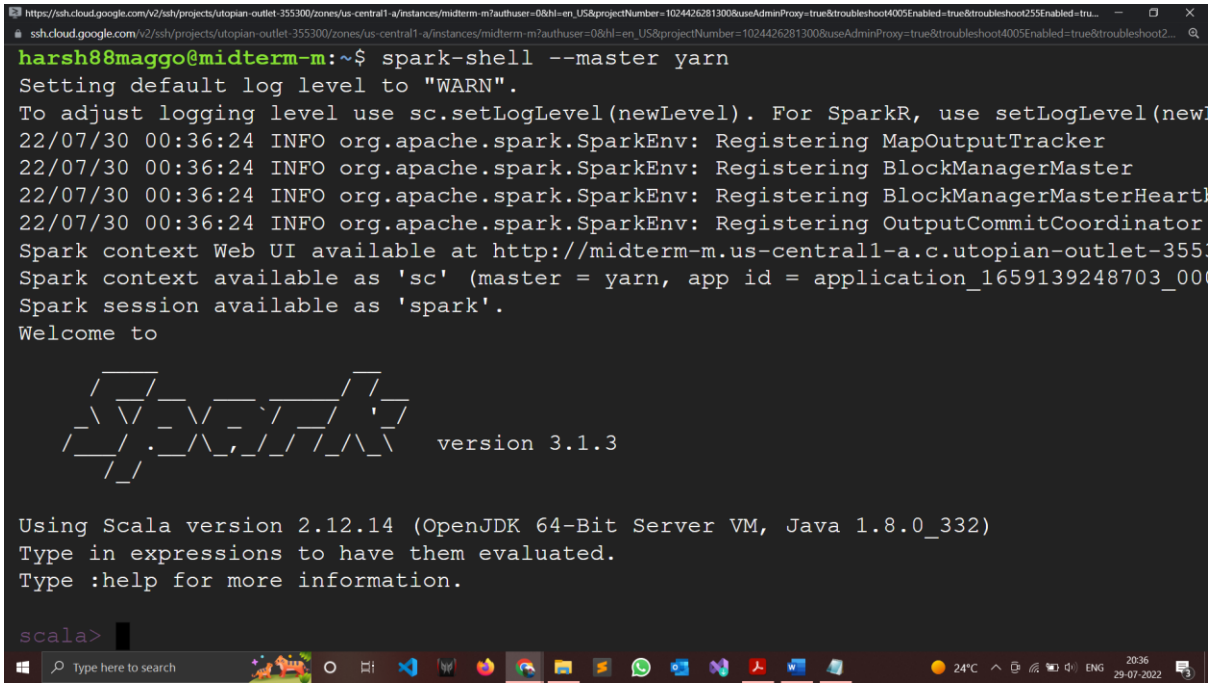
Cancer Data (With Logistic Regression)

Creating a Hadoop directory and loading the data from the local files system to Hadoop



Running the Spark Shell



Import statements

```
Type in expressions to have them evaluated.
Type :help for more information.

scala> :paste
// Entering paste mode (ctrl-D to finish)

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression

// Exiting paste mode, now interpreting.

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression

scala>
```

Reading the CSV file

```
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression

// Exiting paste mode, now interpreting.

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression

scala> :paste
// Entering paste mode (ctrl-D to finish)

val data_harshdeep= spark.read
 .format("csv")
 .option("header", "true")
 .load("hdfs://10.128.0.7/MidTerm/cancer.csv")

// Exiting paste mode, now interpreting.

data_harshdeep: org.apache.spark.sql.DataFrame = [id: string, Clump Thickness: string ... 9 more fields]

scala>
```

Dropping the ‘ID’ column since it won’t help us in prediction

```
// Exiting paste mode, now interpreting.

import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression

scala> :paste
// Entering paste mode (ctrl-D to finish)

val data_harshdeep= spark.read
 .format("csv")
 .option("header", "true")
 .load("hdfs://10.128.0.7/MidTerm/cancer.csv")

// Exiting paste mode, now interpreting.

data_harshdeep: org.apache.spark.sql.DataFrame = [id: string, Clump Thickness: string ... 9 more fields]

scala> val df_harshdeep = data_harshdeep.drop("id")
df_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: string, UofCSize: string ... 8 more fields]

scala>
```

Typecasting the data into type Double for our model training

```
 .load("hdfs://10.128.0.7/MidTerm/cancer.csv")

// Exiting paste mode, now interpreting.

data_harshdeep: org.apache.spark.sql.DataFrame = [id: string, Clump Thickness: string ... 9 more fields]

scala> val df_harshdeep = data_harshdeep.drop("id")
df_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: string, UofCSize: string ... 8 more fields]

scala> :paste
// Entering paste mode (ctrl-D to finish)

val dataset_harshdeep = df_harshdeep.select(
 col("Clump Thickness").cast(DoubleType),
 col("UofCSize").cast(DoubleType),
 col("UofCShape").cast(DoubleType),
 col("Marginal Adhesion").cast(DoubleType),
 col("SECSize").cast(DoubleType),
 col("Bare Nuclei").cast(DoubleType),
 col("Bland Chromatin").cast(DoubleType),
col("Normal Nucleoli").cast(DoubleType),
 col("Mitoses").cast(DoubleType),
 col("Class").cast(DoubleType))

// Exiting paste mode, now interpreting.

dataset_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: double, UofCSize: double ... 8 more fields]

scala>
```

Split the dataset into train and test

```
scala> val df_harshdeep = data_harshdeep.drop("id")
df_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: string, UofCSize: string ... 8 more fields]

scala> :paste
// Entering paste mode (ctrl-D to finish)

val dataset_harshdeep = df_harshdeep.select(
 col("Clump Thickness").cast(DoubleType),
 col("UofCSize").cast(DoubleType),
 col("UofCShape").cast(DoubleType),
 col("Marginal Adhesion").cast(DoubleType),
 col("SECSize").cast(DoubleType),
 col("Bare Nuclei").cast(DoubleType),
 col("Bland Chromatin").cast(DoubleType),
col("Normal Nucleoli").cast(DoubleType),
 col("Mitoses").cast(DoubleType),
 col("Class").cast(DoubleType))

// Exiting paste mode, now interpreting.

dataset_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: double, UofCSize: double ... 8 more fields]

scala> val Array(trainingdata_harshdeep, testdata_harshdeep) = dataset_harshdeep.randomSplit(Array(0.8, 0.2), 521)
trainingdata_harshdeep: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Clump Thickness: double, UofCSize: double ...
 8 more fields]
testdata_harshdeep: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Clump Thickness: double, UofCSize: double ... 8 m
ore fields]

scala>
```

Assembling the features using VectorAssembler

```
 col("Bland Chromatin").cast(DoubleType),
col("Normal Nucleoli").cast(DoubleType),
 col("Mitoses").cast(DoubleType),
 col("Class").cast(DoubleType))

// Exiting paste mode, now interpreting.

dataset_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: double, UofCSize: double ... 8 more fields]

scala> val Array(trainingdata_harshdeep, testdata_harshdeep) = dataset_harshdeep.randomSplit(Array(0.8, 0.2), 521)
trainingdata_harshdeep: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Clump Thickness: double, UofCSize: double ...
 8 more fields]
testdata_harshdeep: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Clump Thickness: double, UofCSize: double ... 8 m
ore fields]

scala> :paste
// Entering paste mode (ctrl-D to finish)

val assembler_harshdeep = new VectorAssembler()
 .setInputCols(Array("Clump Thickness", "UofCSize", "UofCShape", "Marginal Adhesion", "SECSize", "Bare Nuclei", "Normal Nucleol
i","Mitoses"))
 .setOutputCol("assembled-features")

// Exiting paste mode, now interpreting.

assembler_harshdeep: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_3fc50a0ac881, handleInvali
d=error, numInputCols=8

scala>
```

Creating the Logistic Regression Object and passing the features

```
 8 more fields]
testdata_harshdeep: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Clump Thickness: double, UofCSize: double ... 8 m
ore fields]

scala> :paste
// Entering paste mode (ctrl-D to finish)

val assembler_harshdeep = new VectorAssembler()
 .setInputCols(Array("Clump Thickness", "UofCSize", "UofCShape", "Marginal Adhesion", "SECSize", "Bare Nuclei", "Normal Nucleol
i","Mitoses"))
 .setOutputCol("assembled-features")

// Exiting paste mode, now interpreting.

assembler_harshdeep: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_3fc50a0ac881, handleInvali
d=error, numInputCols=8

scala> :paste
// Entering paste mode (ctrl-D to finish)

val lr_harshdeep = new LogisticRegression()
 .setFeaturesCol("assembled-features")
 .setLabelCol("Class")

// Exiting paste mode, now interpreting.

lr_harshdeep: org.apache.spark.ml.classification.LogisticRegression = logreg_0cc145354c9e

scala>
```

Creating the pipeling

```
 .setOutputCol("assembled-features")

// Exiting paste mode, now interpreting.

assembler_harshdeep: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_3fc50a0ac881, handleInvali
d=error, numInputCols=8

scala> :paste
// Entering paste mode (ctrl-D to finish)

val lr_harshdeep = new LogisticRegression()
 .setFeaturesCol("assembled-features")
 .setLabelCol("Class")

// Exiting paste mode, now interpreting.

lr_harshdeep: org.apache.spark.ml.classification.LogisticRegression = logreg_0cc145354c9e

scala> :paste
// Entering paste mode (ctrl-D to finish)

val pipeline_harshdeep = new Pipeline()
  .setStages(Array(assembler_harshdeep, lr_harshdeep))

// Exiting paste mode, now interpreting.

pipeline_harshdeep: org.apache.spark.ml.Pipeline = pipeline_e9e22cac5bc9

scala>
```

Evaluator for our model

```
// Exiting paste mode, now interpreting.

lr_harshdeep: org.apache.spark.ml.classification.LogisticRegression = logreg_0cc145354c9e

scala> :paste
// Entering paste mode (ctrl-D to finish)

val pipeline_harshdeep = new Pipeline()
  .setStages(Array(assembler_harshdeep, lr_harshdeep))

// Exiting paste mode, now interpreting.

pipeline_harshdeep: org.apache.spark.ml.Pipeline = pipeline_e9e22cac5bc9

scala> :paste
// Entering paste mode (ctrl-D to finish)

val evaluator_harshdeep = new MulticlassClassificationEvaluator()
  .setLabelCol("Class")
  .setPredictionCol("prediction")
  .setMetricName("accuracy")

// Exiting paste mode, now interpreting.

evaluator_harshdeep: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=
mcEval_1f84ef79fa3a, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>
```

Setting the hyperparameters

```
// Entering paste mode (ctrl-D to finish)

val paramGrid_harshdeep = new ParamGridBuilder()
  .addGrid(lr.regParam, Array(0.1, 0.01))
  .build()

// Exiting paste mode, now interpreting.

<pastie>:34: error: not found: value lr
  .addGrid(lr.regParam, Array(0.1, 0.01))
           ^

scala> :paste
// Entering paste mode (ctrl-D to finish)

val paramGrid_harshdeep = new ParamGridBuilder()
  .addGrid(lr_harshdeep.regParam, Array(0.1, 0.01))
  .build()

// Exiting paste mode, now interpreting.

paramGrid_harshdeep: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        logreg_0cc145354c9e-regParam: 0.1
}, {
        logreg_0cc145354c9e-regParam: 0.01
})

scala>
```

Creating the Cross Validator

```
// Entering paste mode (ctrl-D to finish)

val paramGrid_harshdeep = new ParamGridBuilder()
  .addGrid(lr_harshdeep.regParam, Array(0.1, 0.01))
  .build()

// Exiting paste mode, now interpreting.

paramGrid_harshdeep: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        logreg_0cc145354c9e-regParam: 0.1
}, {
        logreg_0cc145354c9e-regParam: 0.01
})

scala> :paste
// Entering paste mode (ctrl-D to finish)

val cross_validator_harshdeep = new CrossValidator()
  .setEstimator(pipeline_harshdeep)
  .setEvaluator(evaluator_harshdeep)
  .setEstimatorParamMaps(paramGrid_harshdeep)
  .setNumFolds(3)

// Exiting paste mode, now interpreting.

cross_validator_harshdeep: org.apache.spark.ml.tuning.CrossValidator = cv_bc459bd801a1

scala>
```

Training our model

```
paramGrid_harshdeep: Array[org.apache.spark.ml.param.ParamMap] =
Array({
        logreg_0cc145354c9e-regParam: 0.1
}, {
        logreg_0cc145354c9e-regParam: 0.01
})

scala> :paste
// Entering paste mode (ctrl-D to finish)

val cross_validator_harshdeep = new CrossValidator()
  .setEstimator(pipeline_harshdeep)
  .setEvaluator(evaluator_harshdeep)
  .setEstimatorParamMaps(paramGrid_harshdeep)
  .setNumFolds(3)

// Exiting paste mode, now interpreting.

cross_validator_harshdeep: org.apache.spark.ml.tuning.CrossValidator = cv_bc459bd801a1

scala> val cvModel_harshdeep = cross_validator_harshdeep.fit(trainingdata_harshdeep)
22/07/30 01:11:36 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystem
BLAS
22/07/30 01:11:36 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLA
S
cvModel_harshdeep: org.apache.spark.ml.tuning.CrossValidatorModel = CrossValidatorModel: uid=cv_bc459bd801a1, bestModel=pipelin
e_e9e22cac5bc9, numFolds=3

scala>
```

Prediction using testdata

```
  .setNumFolds(3)

//Train the model

val cvModel_harshdeep = cross_validator_harshdeep.fit(trainingdata_harshdeep)

// Exiting paste mode, now interpreting.

22/07/30 01:14:04 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystem
BLAS
22/07/30 01:14:04 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLA
S
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression
data_harshdeep: org.apache.spark.sql.DataFrame = [id: string, Clump Thickness: string ... 9 more fields]
df_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: string, UofCSize: string ... 8 more fields]
datase...

scala> val predictions_harshdeep = cvModel_harshdeep.transform(testdata_harshdeep)
predictions_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: double, UofCSize: double ... 12 more fields]

scala>
```

Evaluating the performance of our model

```
import org.apache.spark.sql.expressions.Window
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.tuning.{CrossValidator, CrossValidatorModel, ParamGridBuilder}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.sql.types.{IntegerType, DoubleType}
import org.apache.spark.ml.classification.LogisticRegression
data_harshdeep: org.apache.spark.sql.DataFrame = [id: string, Clump Thickness: string ... 9 more fields]
df_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: string, UofCSize: string ... 8 more fields]
datase...

scala> val predictions_harshdeep = cvModel_harshdeep.transform(testdata_harshdeep)
predictions_harshdeep: org.apache.spark.sql.DataFrame = [Clump Thickness: double, UofCSize: double ... 12 more fields]

scala> :paste
// Entering paste mode (ctrl-D to finish)


val accuracy_harshdeep = evaluator_harshdeep.evaluate(predictions_harshdeep)

println("accuracy on test data = " + accuracy_harshdeep)

// Exiting paste mode, now interpreting.

accuracy on test data = 0.9568345323741008
accuracy_harshdeep: Double = 0.9568345323741008

scala>
```

ACCURACY: 95.68%