巨量資料分析與應用 期末報告

基本資料

班級：商資三甲　分組編號：第 8 組

組長：學號 C108193121 姓名 黃昱綺

組員：學號 C108193104 姓名 鄭云瑄

題目

Heart failure clinical records Data Set

心力衰竭臨床記錄數據集

--資料分析目的說明--

說明本分析的目標或目的

本組利用「心力衰竭臨床記錄數據集」，age（年齡）、anaemia（貧血）、high blood pressure（高血壓）、creatinine phosphokinase（肌酐磷酸激酶 CPK）、dabetes（糖尿病）、ejection fraction（射血分數）、patelets（血小板）、sex（性別）、serum creatinine（血清肌酐）、serum sodium（血清鈉）、smoking（吸煙）、time（時間），十二個特徵值預測心力衰竭患者的生存率。

資料集描述

資料集個數：1（以下表格不夠自行拷貝）　總筆數：300

資料集 1 名稱：heart　資料集 1 筆數：300

檔名 1：heart_failure_clinical_records_dataset.csv

來源：

https://archive.ics.uci.edu/ml/machine-learning-databases/00519/heart_failure_clinical_records_dataset.csv

欄位名稱	欄位描述（包括編碼格式）
age **年齡**	患者年齡（歲）。
anaemia **貧血**	紅細胞或血紅蛋白減少（布林值）。
high blood pressure **高血壓**	如果患者患有高血壓（布林值）。
creatinine phosphokinase (CPK) **肌酐磷酸激酶**	血液中 CPK 酶的水平（mcg/L）。
dabetes **糖尿病**	如果患者患有糖尿病（布林值）。
ejection fraction **射血分數**	每次收縮時離開心臟的血液百分比（百分比）。
patelets **血小板**	血液中的血小板（千血小板/mL）。
sex **性別**	女性或男性（二元）。
serum creatinine **血清肌酐**	血液中的血清肌酐水平（mg/dL）。
serum sodium **血清鈉**	血液中的血清鈉水平（mEq/L）。
smoking **吸菸**	患者是否吸煙（布林值）。
time **時間**	觀察期（天）。
[target] death event **[目標] 死亡事件**	如果患者在觀察期內死亡（布林值）。

分析策略說明

說明通過哪些步驟或流程？使用哪些資料集、哪些欄位、用何種方法來分析？

使用方法：迴歸分析。

1. 先進入 Ubuntu 作業系統並啟動 hadoop。
2. 下載心力衰竭臨床記錄數據集並上傳至 hdfs。
3. 進入 spark 環境並匯入程式庫。
4. 讀取文本資料，將以逗號分割資料成陣列型態且首行（欄位名稱）進行刪除，並轉為浮點數。
5. 建立欄位對照表，有以下欄位:age（年齡）、anaemia（貧血）、high blood pressure（高血壓）、creatinine phosphokinase（肌酐磷酸激酶 CPK）、dabetes（糖尿病）、ejection fraction（射血分數）、patelets（血小板）、sex（性別）、serum creatinine（血清肌酐）、serum sodium（血清鈉）、smoking（吸煙）、time（時間）。
6. 列印出數據集的基本統計值。
7. 建立資料類別後進行資料標準化。
8. 切割資料集，將資料用 7:2:1 的比例，分成訓練集、驗證集與測試集。
9. 以步進 0.1 和迭代次數 1000 開始訓練模型。
10. 列印出各特徵權重。
11. 求驗證集預測值。
12. 寫 getMSE 副程式取得真實評分與預測評分，合併後算出 MSE。
13. 寫 modelAndMSE 副程式訓練各參數組合（步進+迭代次數）的模型求 MSE 值並選取最低 MSE 值的模型。
14. 列出各特徵變數權重。

程式碼

- 請使用 Consolas 字型

進入 Ubuntu 環境	
`start-all.sh`	啟動 hadoop，若已經啟動，則不需要再執行此指令。
`cd ~`	回到家目錄。
`wget https://archive.ics.uci.edu/ml/machine-learning-databases/00519/heart_failure_clinical_records_dataset.csv`	下載心力衰竭臨床記錄數據集。
`hdfs dfs -mkdir /heart`	在 hdfs 上建立目錄。
`hdfs dfs -put ~/heart_failure_clinical_records_dataset.csv /heart`	將檔案上傳 hdfs。
`hdfs dfs -ls /heart`	查看 hdfs 資料。
進入 spark-shell 環境，開始處理	
`spark-shell`	進入 spark-shell

`import org.apache.spark.mllib.linalg.Vectors`
`import org.apache.spark.mllib.regression.LabeledPoint`
`import org.apache.spark.mllib.regression.LinearRegressionWithSGD`
`import org.apache.spark.mllib.feature.StandardScaler`
	匯入四個必要的程式庫。

`val rawData=sc.textFile("/home/mis/heart_failure_clinical_records_dataset.csv")`
 	
`val heartStringRDD=rawData.map(line => line.split(",")).mapPartitionsWithIndex { (index,lines) => if (index==0) lines.drop(1) else lines }`
 	
`val`
	讀入文本資料，去除首行（欄位名稱），並轉成浮點數。

heartRDD=heartStringRDD.map(x=>x.map(x=>x .toDouble)) heartRDD.count heartRDD.first	
val colNameMap=Map(0->"age", 1->"anaemia", 2->"creatinine_phosphokinase", 3->"diabetes", 4->"ejection_fraction", 5->"high_blood_pressure", 6->"platelets", 7->"serum_creatinine", 8->"serum_sodium", 9->"sex", 10->"smoking", 11->"time", 12->"DEATH_EVENT")	建立欄位對照表。
基本統計資料	

def printStats(rdd:
org.apache.spark.rdd.RDD[Array[Double]]) {
  for(i<-0 until rdd.first.length) {
    val name=colNameMap(i)
    val s=rdd.map(x=>x(i)).stats
    val sp="".padTo(25-name.length,"
").mkString("")
    println(f"$name$sp$s")
  }
}
 	
printStats(heartRDD)
	列印資料集的基本統計值。

LabeledPoint	
val heartLP=heartRDD.map(line=>new LabeledPoint(line.last, Vectors.dense(line.init))) heartLP.first	建立 LabeledPoint 資料類別。

heartLP.first.label heartLP.first.features	
資料標準化（standardize）	
val heartScaler = new StandardScaler(withMean = true, withStd = true).fit(heartLP.map(x=>x.features))	建立 heartScaler 物件，配適 heartLP 的尺度。
val scaledHeartLP=heartLP.map(x => new LabeledPoint(x.label, heartScaler.transform(x.features))) printStats(scaledHeartLP.map(x=>x.features.toArray))	進行資料標準化。
切割資料集	
val heartSplit = scaledHeartLP.randomSplit(Array(0.7, 0.2, 0.1),1688) val heartTrainSet = heartSplit(0) val heartValidSet = heartSplit(1) val heartTestSet = heartSplit(2) heartTrainSet.count heartValidSet.count heartTestSet.count heartTrainSet.cache	切割資料集。
模型訓練	
val stepSize=0.1 val numOfIter = 1000 val heartModelInst = new LinearRegressionWithSGD().setIntercept(true) heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize) val heartModel=heartModelInst.run(heartTrainSet)	以步進 0.1 和迭代次數 1000 訓練模型。

heartModel.weights	

def printWeights(w:
org.apache.spark.mllib.linalg.Vector) {
val
pw=colNameMap.toArray.sortBy(x=>x._1).map
(x=>x._2).zip(w.toArray).sortBy(x=>x._2).
reverse
pw.foreach{ case (name, wgt) =>
      val sp="".padTo(25-name.length,"
").mkString("")
      println(f"$name$sp$wgt")
    }
}
printWeights(heartModel.weights)
	列印各特徵權重的副程式。

預測與模式評估	
val heartPred=heartModel.predict(heartValidSe t.map(x=>x.features)) heartPred.first heartValidSet.first	求驗證集預測值。

def getMSE(model:
org.apache.spark.mllib.regression.LinearR
egressionModel, dataset:
org.apache.spark.rdd.RDD[org.apache.spark
.mllib.regression.LabeledPoint])={
   val heartReal=dataset.map(x=>x.label)
// 取得真實評分
   val
heartPred=model.predict(dataset.map(x=>x.
features)) // 預測評分
   val
realWithPred=heartReal.zip(heartPred) //
真實與預測分數合併
   val MSE=realWithPred.map{case (real,
pred)=>math.pow(real-pred,2)}.mean()
   MSE // 最後結果 MSE
}
 	
val MSE=getMSE(heartModel, heartValidSet)
	計算 MSE 的副程式，輸入一個已經訓練好的模型和資料集（通常是驗證資料集）。

`println("MSE: "+ MSE)`	
最佳化參數的最佳組合	

`def modelAndMSE(stepSize:Double, numOfIter:Int, dataset: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]) {`
`// 訓練模型`
`val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)`
`heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)`
`val heartModel=heartModelInst.run(heartTrainSet)`
 	
`// 求預測值`
`  val heartPred=heartModel.predict(dataset.map(x=>x.features))`
`  val heartReal=dataset.map(x=>x.label)`
 	
`// 計算 MSE`
`val MSE=getMSE(heartModel, dataset)`
`println("MSE: "+ MSE+"<=="+stepSize+"/"+numOfIter)`
 	
`}`
	訓練某一參數組合（步進+迭代次數）的模型，並計算 MSE。

`for(i<-Array(0.1,0.2,0.3,0.4,0.5,0.6); j<-Array(100,150,200,250,300)) {`
`  modelAndMSE(i,j, heartValidSet)`
`}`
	求取各超參數組合的 MSE。

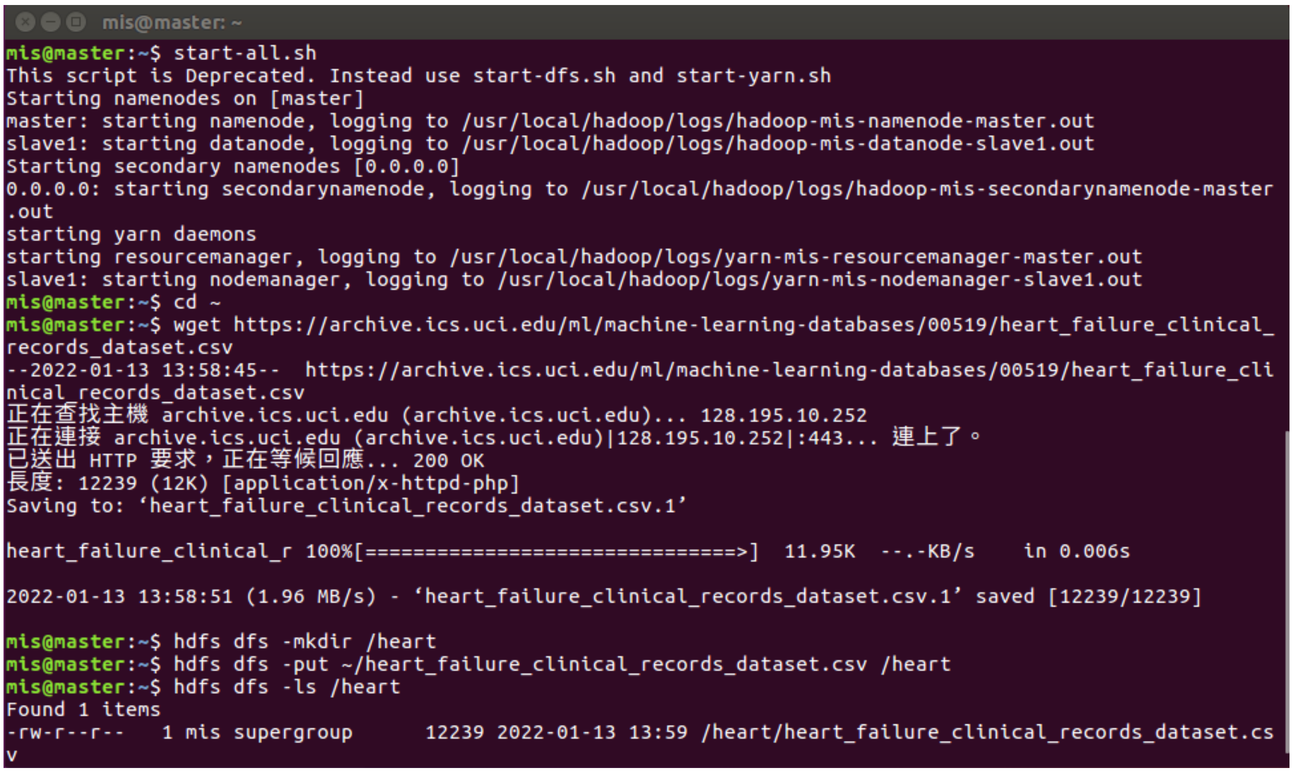
`val stepSize=0.5`
`val numOfIter=100`
`val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)`
`heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)`
	使用最佳超操數組合訓練模型。

`val heartModel=heartModelInst.run(heartTrainSet)`	

`val MSE=getMSE(heartModel , heartTestSet)`
`println("MSE: "+ MSE)`
	求最佳模型的 MSE。

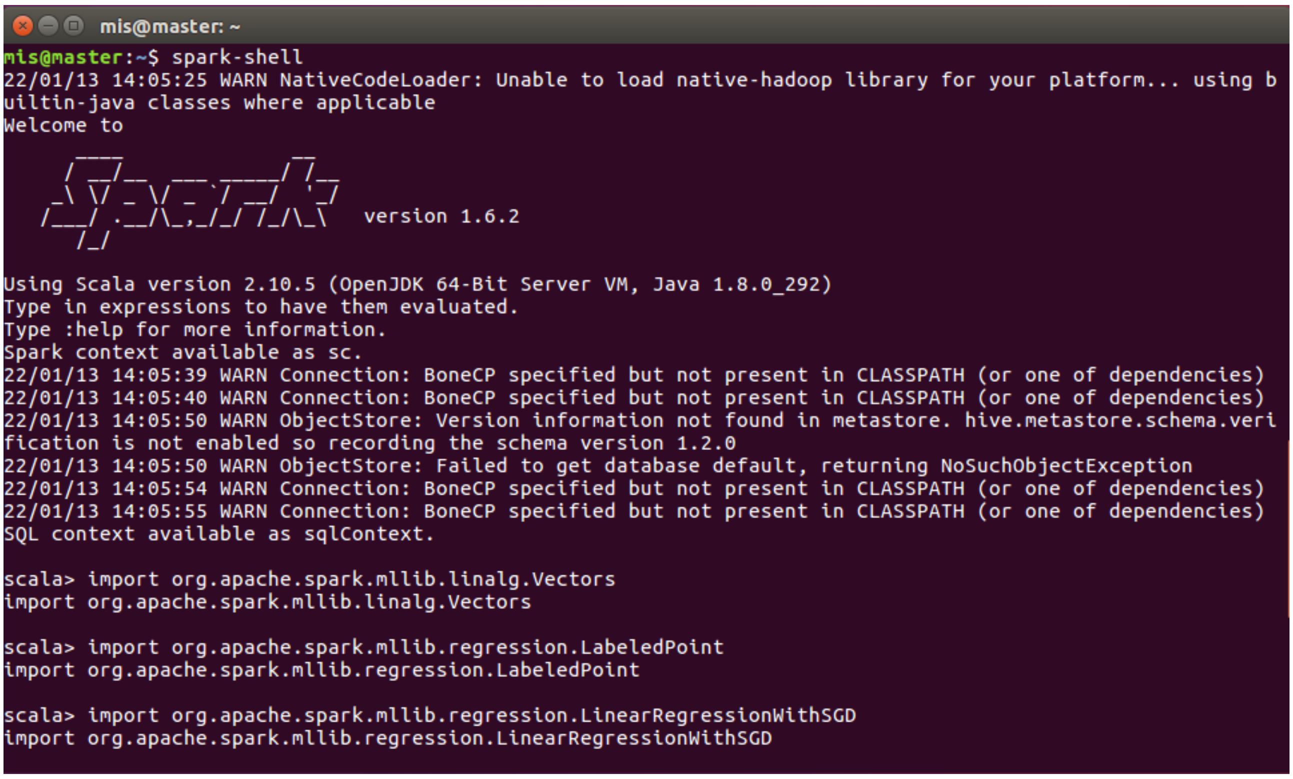
`printWeights(heartModel.weights)`	列印各特徵變數權重。

HardCopy執行畫面



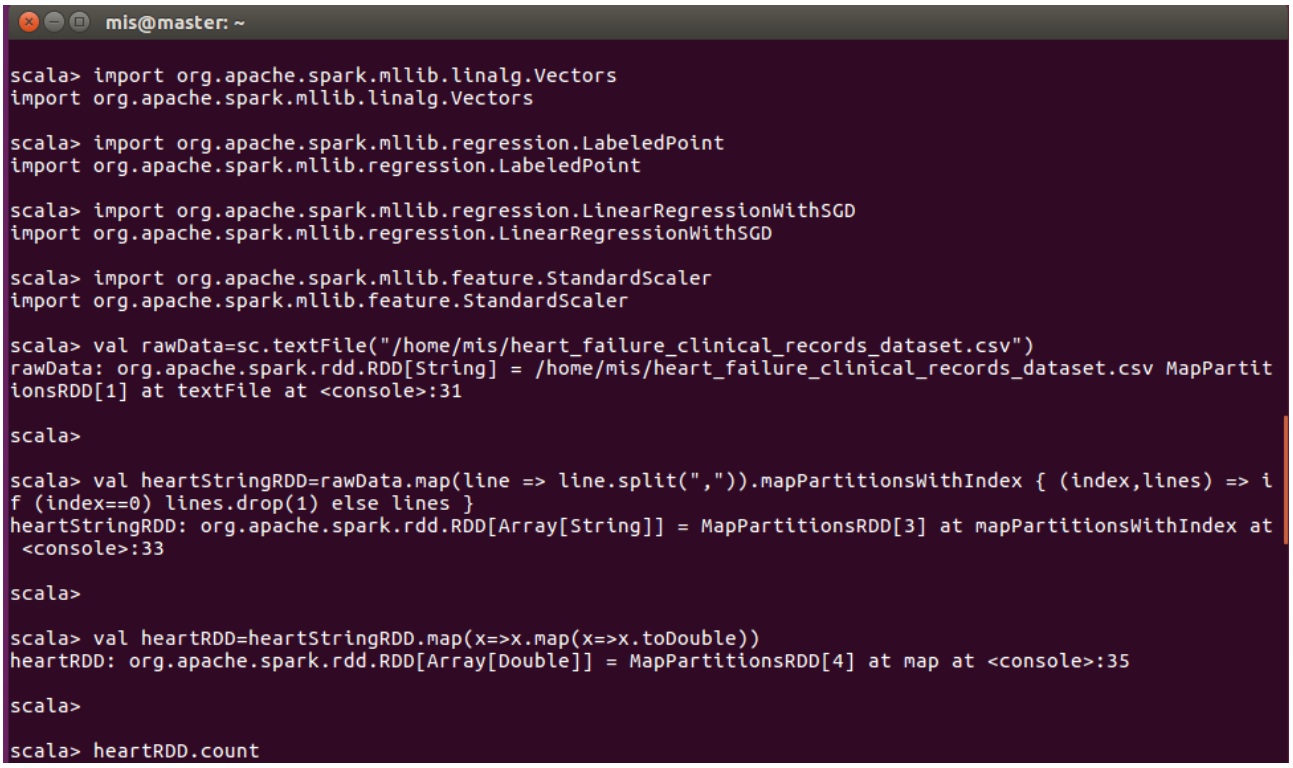
mis@master: ~

```
mis@master:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [master]
master: starting namenode, logging to /usr/local/hadoop/logs/hadoop-mis-namenode-master.out
slave1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-mis-datanode-slave1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-mis-secondarynamenode-master
.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-mis-resourcemanager-master.out
slave1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-mis-nodemanager-slave1.out
mis@master:~$ cd ~
mis@master:~$ wget https://archive.ics.uci.edu/ml/machine-learning-databases/00519/heart_failure_clinical_
records_dataset.csv
--2022-01-13 13:58:45--  https://archive.ics.uci.edu/ml/machine-learning-databases/00519/heart_failure_cli
nical_records_dataset.csv
正在查找主機 archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
正在連接 archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... 連上了。
已送出 HTTP 要求，正在等候回應... 200 OK
長度: 12239 (12K) [application/x-httpd-php]
Saving to: ‘heart_failure_clinical_records_dataset.csv.1’

heart_failure_clinical_r 100%[================================>]  11.95K  --.-KB/s    in 0.006s

2022-01-13 13:58:51 (1.96 MB/s) - ‘heart_failure_clinical_records_dataset.csv.1’ saved [12239/12239]

mis@master:~$ hdfs dfs -mkdir /heart
mis@master:~$ hdfs dfs -put ~/heart_failure_clinical_records_dataset.csv /heart
mis@master:~$ hdfs dfs -ls /heart
Found 1 items
-rw-r--r--   1 mis supergroup      12239 2022-01-13 13:59 /heart/heart_failure_clinical_records_dataset.cs
v
```

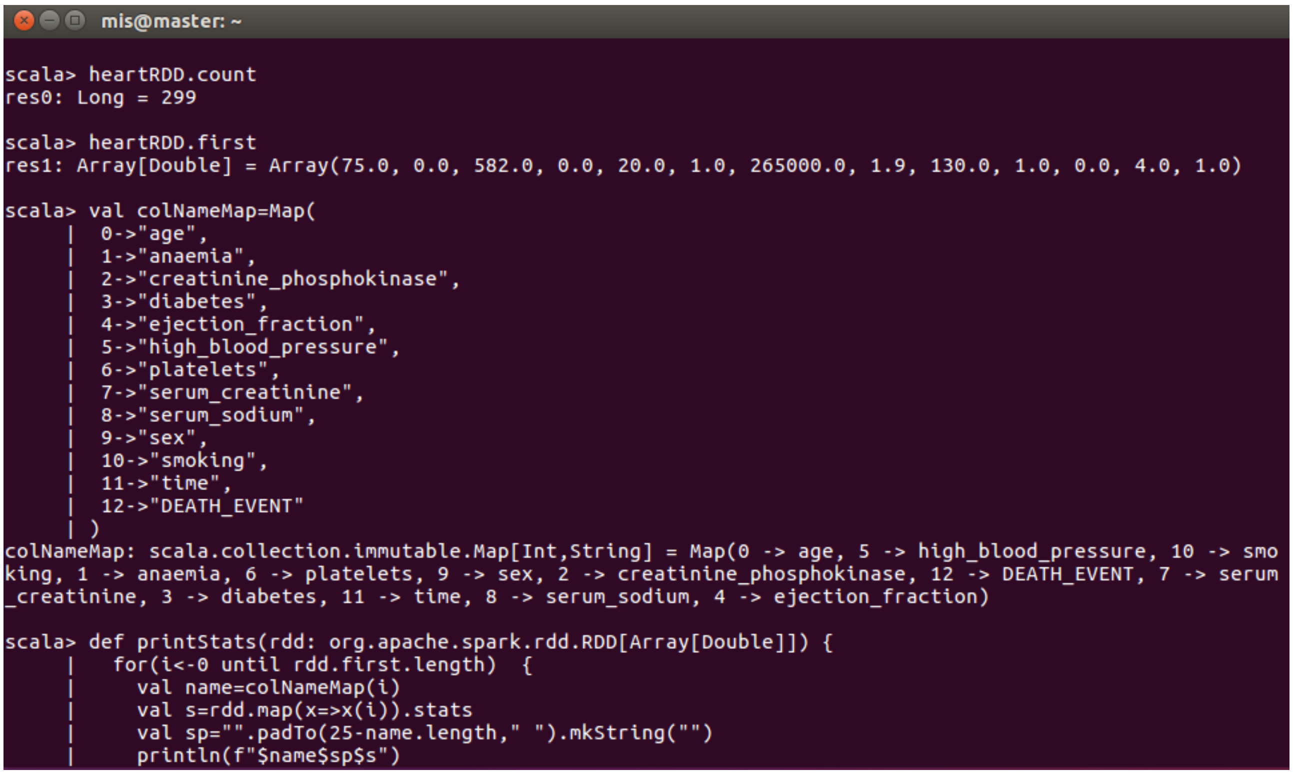


mis@master: ~

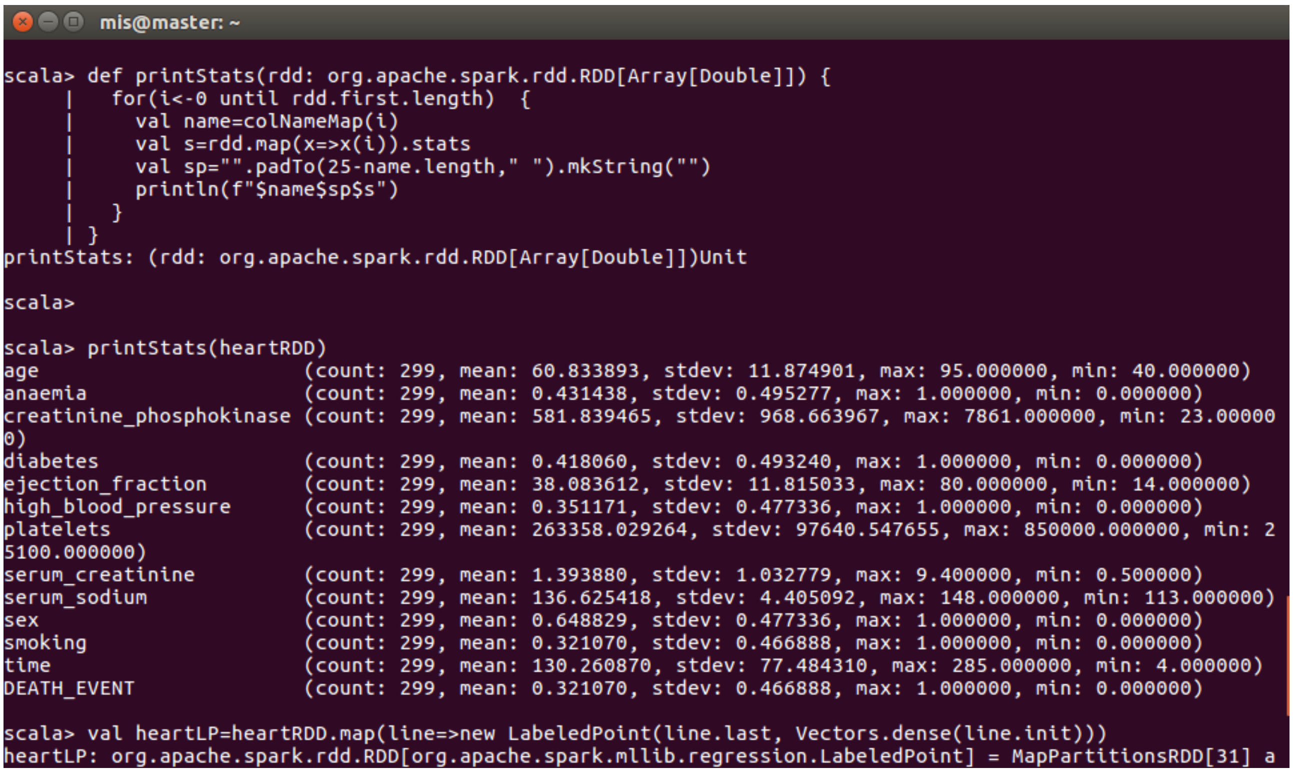
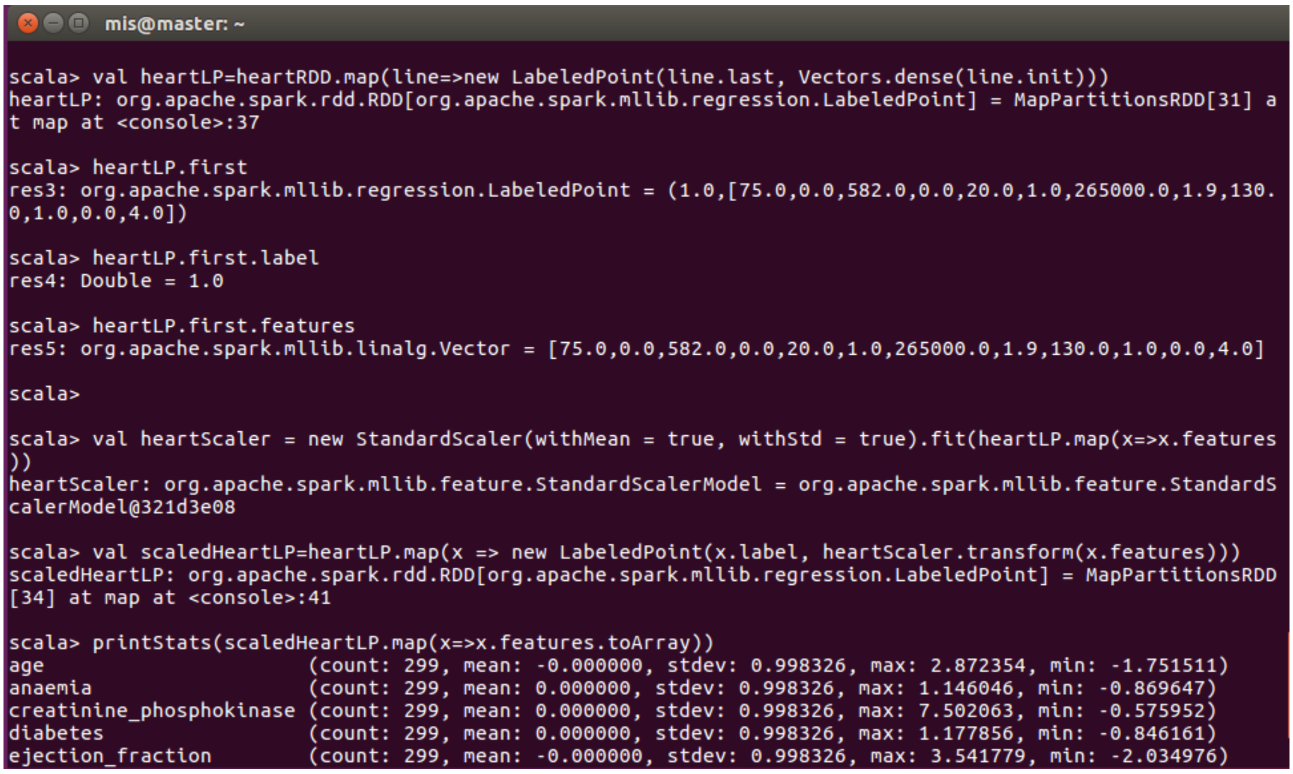
```
mis@master:~$ spark-shell
22/01/13 14:05:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using b
uiltin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.2
      /_/

Using Scala version 2.10.5 (OpenJDK 64-Bit Server VM, Java 1.8.0_292)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
22/01/13 14:05:39 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
22/01/13 14:05:40 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
22/01/13 14:05:50 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.veri
fication is not enabled so recording the schema version 1.2.0
22/01/13 14:05:50 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
22/01/13 14:05:54 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
22/01/13 14:05:55 WARN Connection: BoneCP specified but not present in CLASSPATH (or one of dependencies)
SQL context available as sqlContext.

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
```

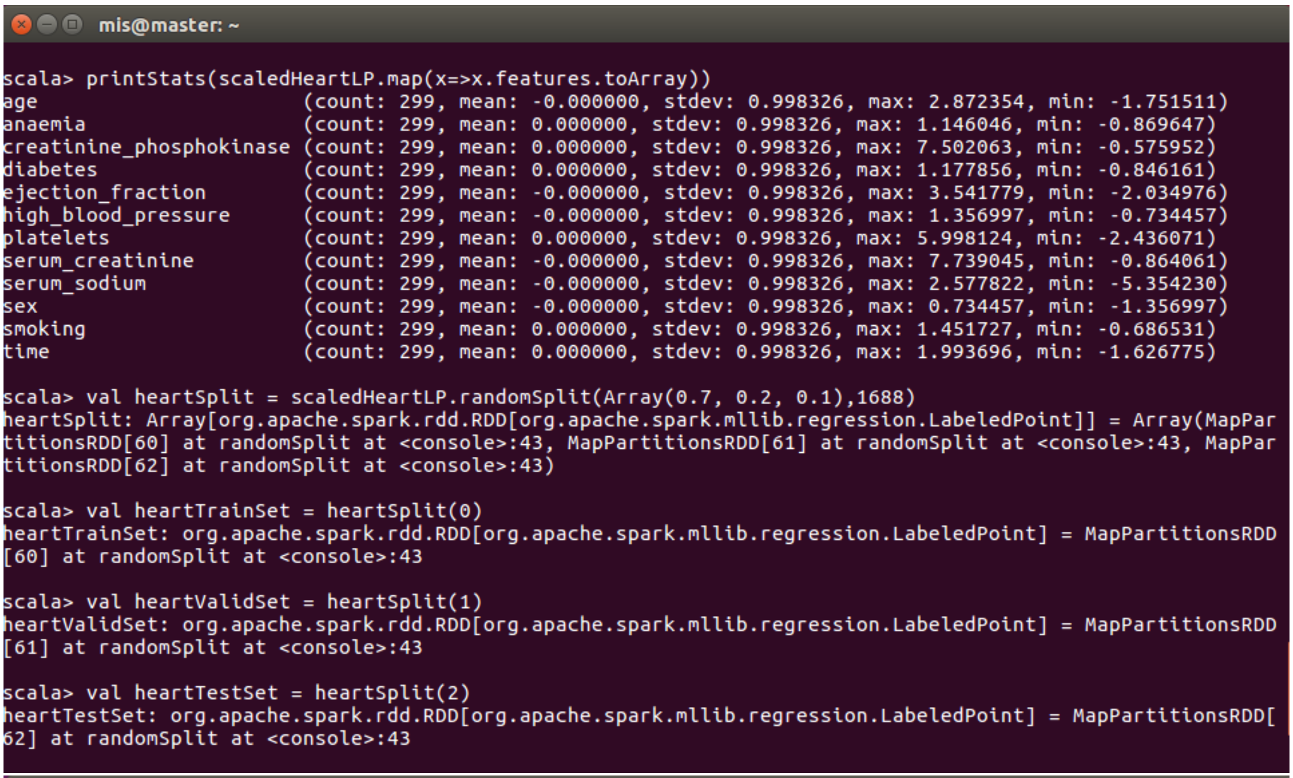


```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

scala> import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.feature.StandardScaler

scala> val rawData=sc.textFile("/home/mis/heart_failure_clinical_records_dataset.csv")
rawData: org.apache.spark.rdd.RDD[String] = /home/mis/heart_failure_clinical_records_dataset.csv MapPartit
ionsRDD[1] at textFile at <console>:31

scala>

scala> val heartStringRDD=rawData.map(line => line.split(",")).mapPartitionsWithIndex { (index,lines) => i
f (index==0) lines.drop(1) else lines }
heartStringRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at mapPartitionsWithIndex at
 <console>:33

scala>

scala> val heartRDD=heartStringRDD.map(x=>x.map(x=>x.toDouble))
heartRDD: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[4] at map at <console>:35

scala>

scala> heartRDD.count
```

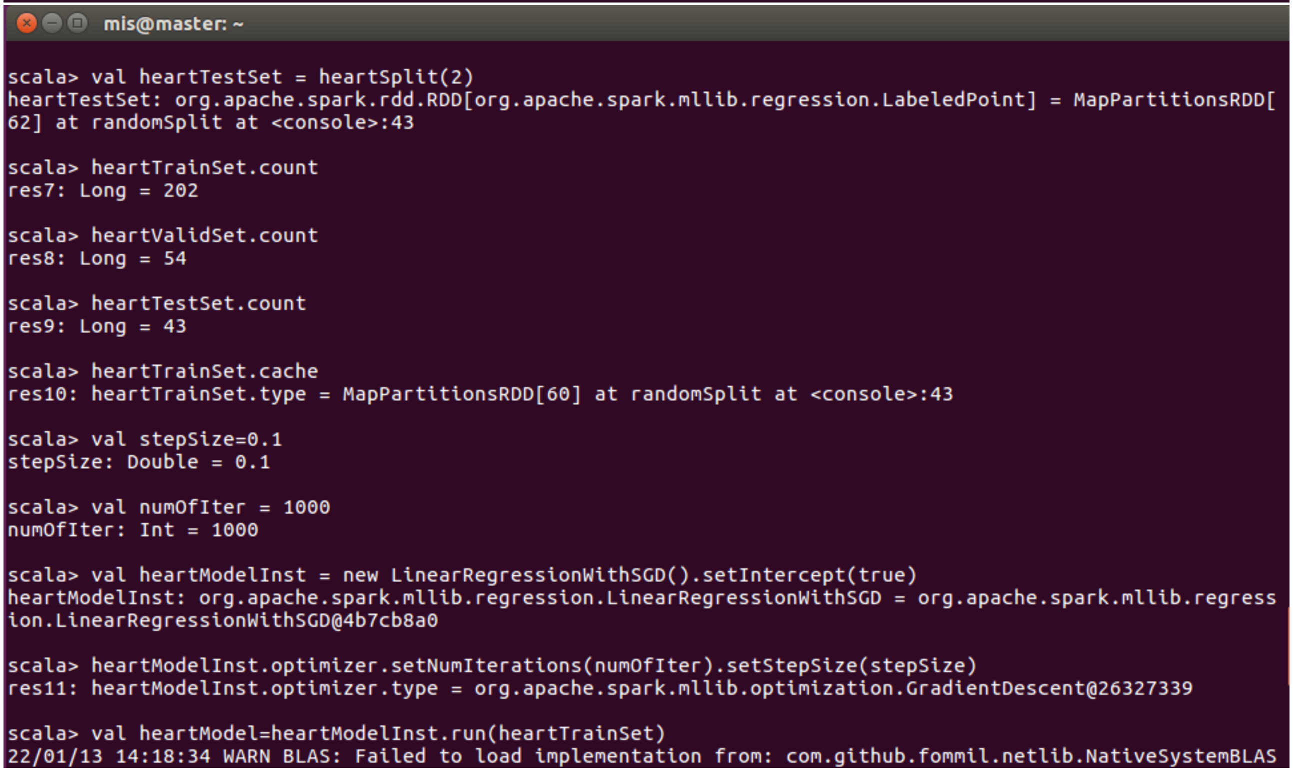


```
scala> heartRDD.count
res0: Long = 299

scala> heartRDD.first
res1: Array[Double] = Array(75.0, 0.0, 582.0, 0.0, 20.0, 1.0, 265000.0, 1.9, 130.0, 1.0, 0.0, 4.0, 1.0)

scala> val colNameMap=Map(
     |   0->"age",
     |   1->"anaemia",
     |   2->"creatinine_phosphokinase",
     |   3->"diabetes",
     |   4->"ejection_fraction",
     |   5->"high_blood_pressure",
     |   6->"platelets",
     |   7->"serum_creatinine",
     |   8->"serum_sodium",
     |   9->"sex",
     |   10->"smoking",
     |   11->"time",
     |   12->"DEATH_EVENT"
     | )
colNameMap: scala.collection.immutable.Map[Int,String] = Map(0 -> age, 5 -> high_blood_pressure, 10 -> smo
king, 1 -> anaemia, 6 -> platelets, 9 -> sex, 2 -> creatinine_phosphokinase, 12 -> DEATH_EVENT, 7 -> serum
_creatinine, 3 -> diabetes, 11 -> time, 8 -> serum_sodium, 4 -> ejection_fraction)

scala> def printStats(rdd: org.apache.spark.rdd.RDD[Array[Double]]) {
     |    for(i<-0 until rdd.first.length)  {
     |      val name=colNameMap(i)
     |      val s=rdd.map(x=>x(i)).stats
     |      val sp="".padTo(25-name.length," ").mkString("")
     |      println(f"$name$sp$s")
```

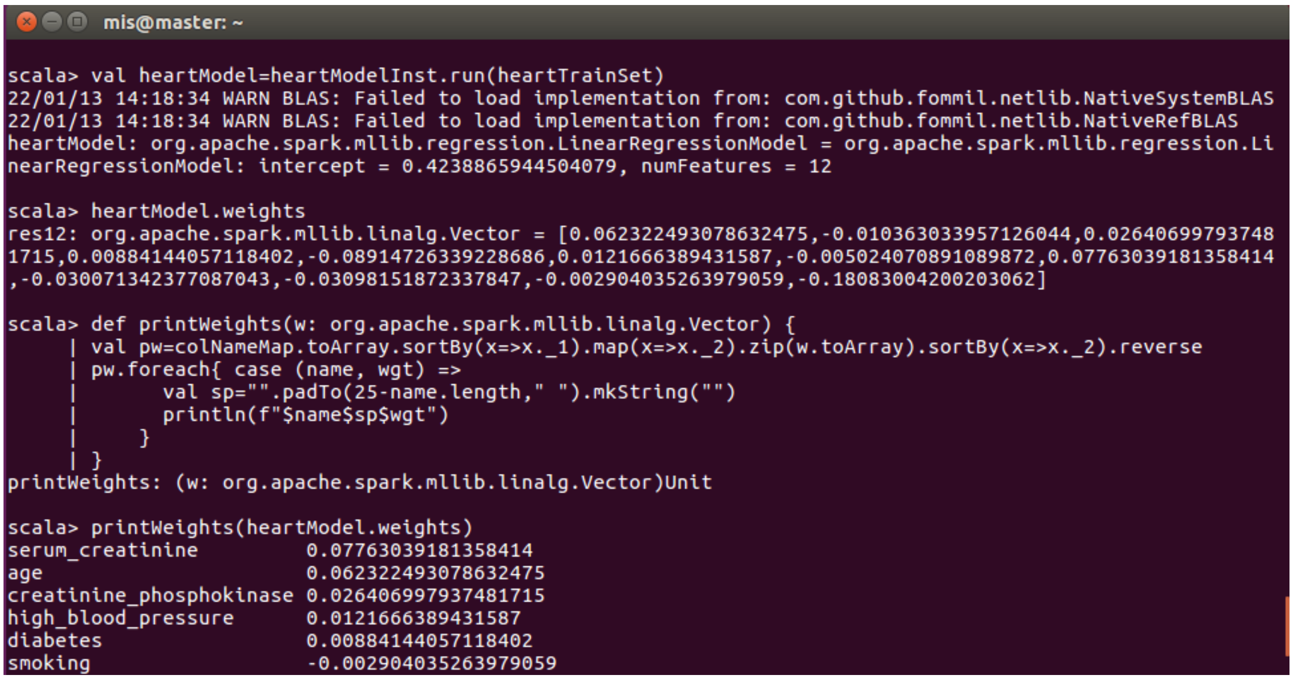

```
mis@master: ~

scala> def printStats(rdd: org.apache.spark.rdd.RDD[Array[Double]]) {
     |   for(i<-0 until rdd.first.length)  {
     |     val name=colNameMap(i)
     |     val s=rdd.map(x=>x(i)).stats
     |     val sp="".padTo(25-name.length," ").mkString("")
     |     println(f"$name$sp$s")
     |   }
     | }
printStats: (rdd: org.apache.spark.rdd.RDD[Array[Double]])Unit

scala>

scala> printStats(heartRDD)
age                      (count: 299, mean: 60.833893, stdev: 11.874901, max: 95.000000, min: 40.000000)
anaemia                  (count: 299, mean: 0.431438, stdev: 0.495277, max: 1.000000, min: 0.000000)
creatinine_phosphokinase (count: 299, mean: 581.839465, stdev: 968.663967, max: 7861.000000, min: 23.00000
0)
diabetes                 (count: 299, mean: 0.418060, stdev: 0.493240, max: 1.000000, min: 0.000000)
ejection_fraction        (count: 299, mean: 38.083612, stdev: 11.815033, max: 80.000000, min: 14.000000)
high_blood_pressure      (count: 299, mean: 0.351171, stdev: 0.477336, max: 1.000000, min: 0.000000)
platelets                (count: 299, mean: 263358.029264, stdev: 97640.547655, max: 850000.000000, min: 2
5100.000000)
serum_creatinine         (count: 299, mean: 1.393880, stdev: 1.032779, max: 9.400000, min: 0.500000)
serum_sodium             (count: 299, mean: 136.625418, stdev: 4.405092, max: 148.000000, min: 113.000000)
sex                      (count: 299, mean: 0.648829, stdev: 0.477336, max: 1.000000, min: 0.000000)
smoking                  (count: 299, mean: 0.321070, stdev: 0.466888, max: 1.000000, min: 0.000000)
time                     (count: 299, mean: 130.260870, stdev: 77.484310, max: 285.000000, min: 4.000000)
DEATH_EVENT              (count: 299, mean: 0.321070, stdev: 0.466888, max: 1.000000, min: 0.000000)

scala> val heartLP=heartRDD.map(line=>new LabeledPoint(line.last, Vectors.dense(line.init)))
heartLP: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[31] a
```

```
mis@master: ~

scala> val heartLP=heartRDD.map(line=>new LabeledPoint(line.last, Vectors.dense(line.init)))
heartLP: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[31] a
t map at <console>:37

scala> heartLP.first
res3: org.apache.spark.mllib.regression.LabeledPoint = (1.0,[75.0,0.0,582.0,0.0,20.0,1.0,265000.0,1.9,130.
0,1.0,0.0,4.0])

scala> heartLP.first.label
res4: Double = 1.0

scala> heartLP.first.features
res5: org.apache.spark.mllib.linalg.Vector = [75.0,0.0,582.0,0.0,20.0,1.0,265000.0,1.9,130.0,1.0,0.0,4.0]

scala>

scala> val heartScaler = new StandardScaler(withMean = true, withStd = true).fit(heartLP.map(x=>x.features
))
heartScaler: org.apache.spark.mllib.feature.StandardScalerModel = org.apache.spark.mllib.feature.StandardS
calerModel@321d3e08

scala> val scaledHeartLP=heartLP.map(x => new LabeledPoint(x.label, heartScaler.transform(x.features)))
scaledHeartLP: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD
[34] at map at <console>:41

scala> printStats(scaledHeartLP.map(x=>x.features.toArray))
age                      (count: 299, mean: -0.000000, stdev: 0.998326, max: 2.872354, min: -1.751511)
anaemia                  (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.146046, min: -0.869647)
creatinine_phosphokinase (count: 299, mean: 0.000000, stdev: 0.998326, max: 7.502063, min: -0.575952)
diabetes                 (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.177856, min: -0.846161)
ejection_fraction        (count: 299, mean: -0.000000, stdev: 0.998326, max: 3.541779, min: -2.034976)
```


mis@master: ~

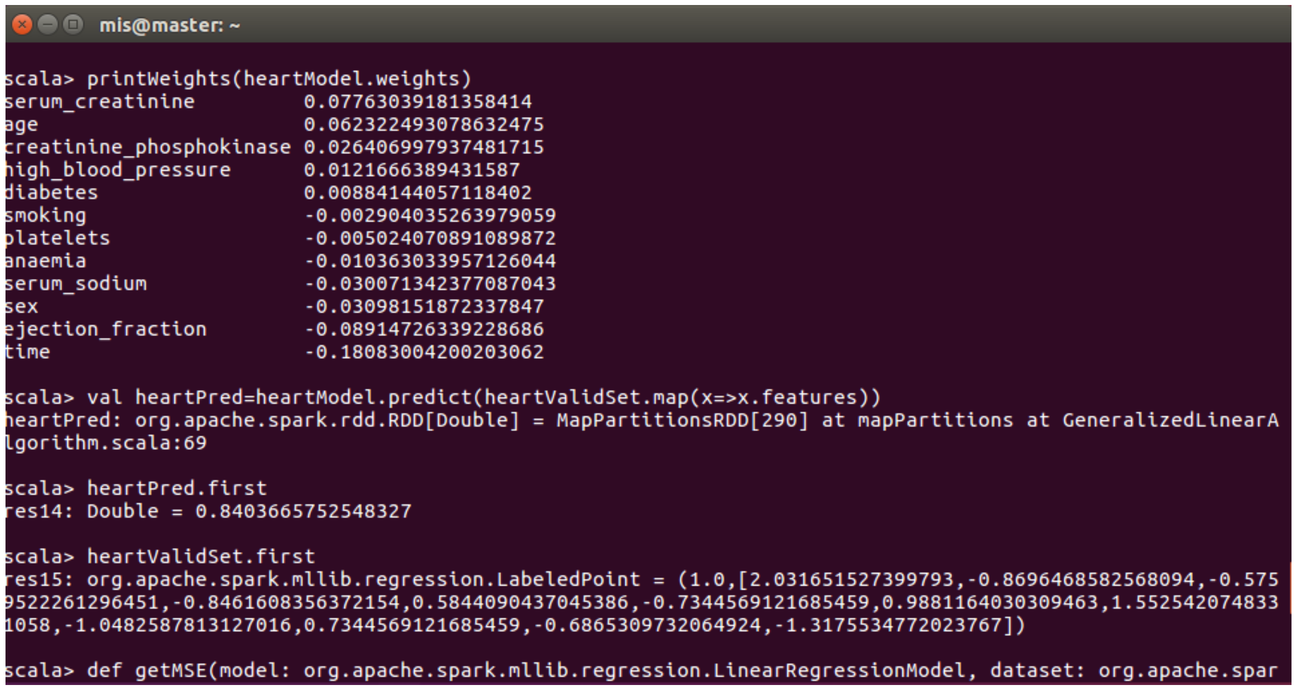
```
scala> printStats(scaledHeartLP.map(x=>x.features.toArray))
age                      (count: 299, mean: -0.000000, stdev: 0.998326, max: 2.872354, min: -1.751511)
anaemia                  (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.146046, min: -0.869647)
creatinine_phosphokinase (count: 299, mean: 0.000000, stdev: 0.998326, max: 7.502063, min: -0.575952)
diabetes                 (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.177856, min: -0.846161)
ejection_fraction        (count: 299, mean: -0.000000, stdev: 0.998326, max: 3.541779, min: -2.034976)
high_blood_pressure      (count: 299, mean: -0.000000, stdev: 0.998326, max: 1.356997, min: -0.734457)
platelets                (count: 299, mean: 0.000000, stdev: 0.998326, max: 5.998124, min: -2.436071)
serum_creatinine         (count: 299, mean: -0.000000, stdev: 0.998326, max: 7.739045, min: -0.864061)
serum_sodium             (count: 299, mean: -0.000000, stdev: 0.998326, max: 2.577822, min: -5.354230)
sex                      (count: 299, mean: -0.000000, stdev: 0.998326, max: 0.734457, min: -1.356997)
smoking                  (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.451727, min: -0.686531)
time                     (count: 299, mean: 0.000000, stdev: 0.998326, max: 1.993696, min: -1.626775)

scala> val heartSplit = scaledHeartLP.randomSplit(Array(0.7, 0.2, 0.1),1688)
heartSplit: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]] = Array(MapPar
titionsRDD[60] at randomSplit at <console>:43, MapPartitionsRDD[61] at randomSplit at <console>:43, MapPar
titionsRDD[62] at randomSplit at <console>:43)

scala> val heartTrainSet = heartSplit(0)
heartTrainSet: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD
[60] at randomSplit at <console>:43

scala> val heartValidSet = heartSplit(1)
heartValidSet: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD
[61] at randomSplit at <console>:43

scala> val heartTestSet = heartSplit(2)
heartTestSet: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[
62] at randomSplit at <console>:43
```


mis@master: ~

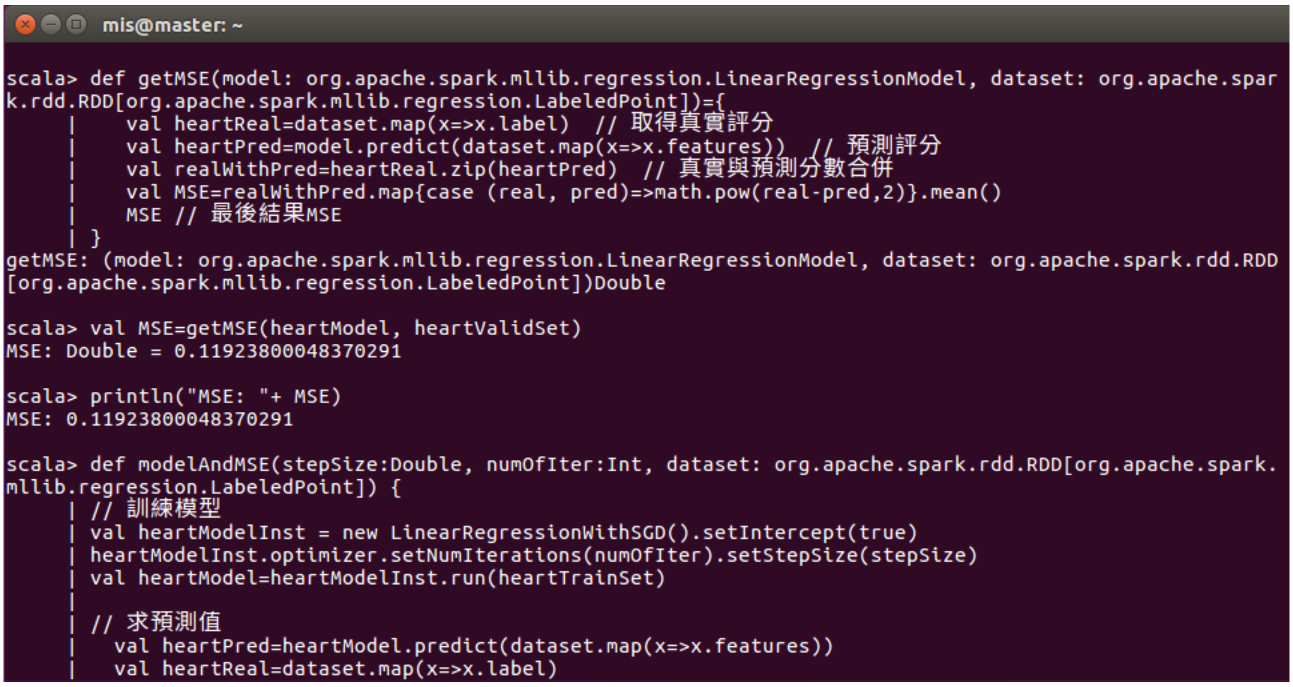
```
scala> val heartTestSet = heartSplit(2)
heartTestSet: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[
62] at randomSplit at <console>:43

scala> heartTrainSet.count
res7: Long = 202

scala> heartValidSet.count
res8: Long = 54

scala> heartTestSet.count
res9: Long = 43

scala> heartTrainSet.cache
res10: heartTrainSet.type = MapPartitionsRDD[60] at randomSplit at <console>:43

scala> val stepSize=0.1
stepSize: Double = 0.1

scala> val numOfIter = 1000
numOfIter: Int = 1000

scala> val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)
heartModelInst: org.apache.spark.mllib.regression.LinearRegressionWithSGD = org.apache.spark.mllib.regress
ion.LinearRegressionWithSGD@4b7cb8a0

scala> heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)
res11: heartModelInst.optimizer.type = org.apache.spark.mllib.optimization.GradientDescent@26327339

scala> val heartModel=heartModelInst.run(heartTrainSet)
22/01/13 14:18:34 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
```



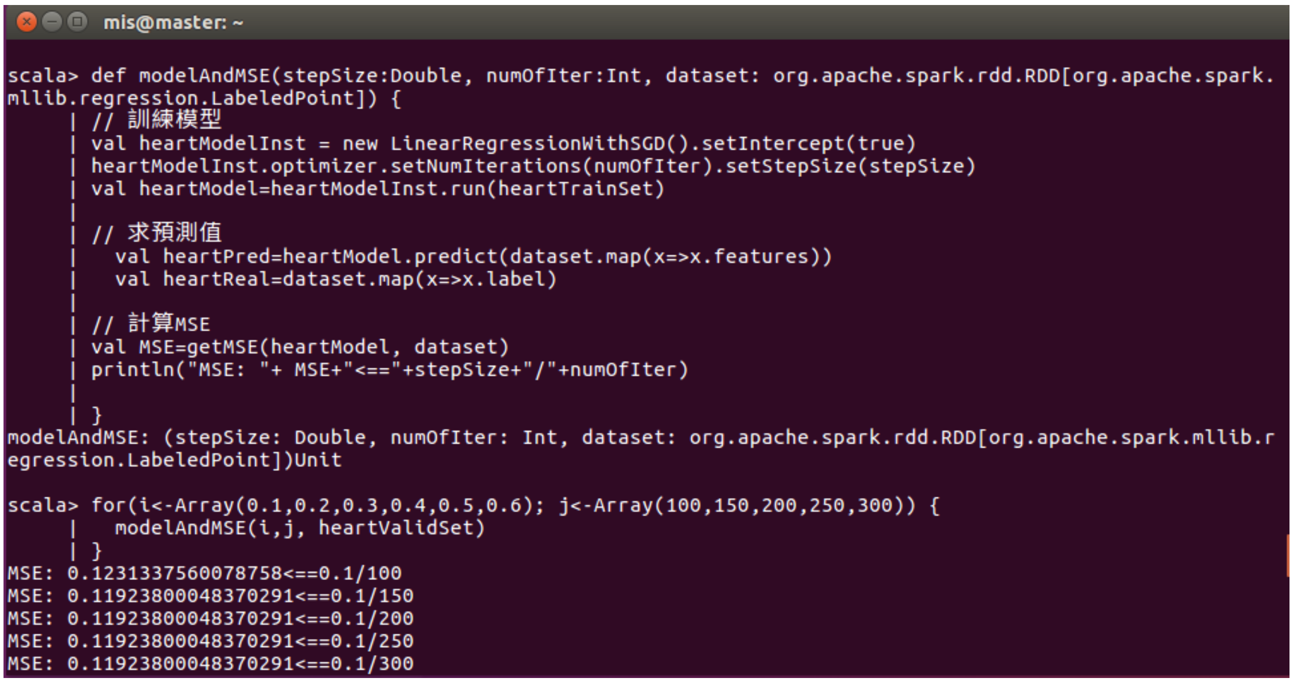
```
scala> val heartModel=heartModelInst.run(heartTrainSet)
22/01/13 14:18:34 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
22/01/13 14:18:34 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
heartModel: org.apache.spark.mllib.regression.LinearRegressionModel = org.apache.spark.mllib.regression.Li
nearRegressionModel: intercept = 0.4238865944504079, numFeatures = 12

scala> heartModel.weights
res12: org.apache.spark.mllib.linalg.Vector = [0.062322493078632475,-0.010363033957126044,0.02640699793748
1715,0.00884144057118402,-0.08914726339228686,0.0121666389431587,-0.005024070891089872,0.07763039181358414
,-0.030071342377087043,-0.03098151872337847,-0.002904035263979059,-0.18083004200203062]

scala> def printWeights(w: org.apache.spark.mllib.linalg.Vector) {
     | val pw=colNameMap.toArray.sortBy(x=>x._1).map(x=>x._2).zip(w.toArray).sortBy(x=>x._2).reverse
     | pw.foreach{ case (name, wgt) =>
     |       val sp="".padTo(25-name.length," ").mkString("")
     |       println(f"$name$sp$wgt")
     |     }
     | }
printWeights: (w: org.apache.spark.mllib.linalg.Vector)Unit

scala> printWeights(heartModel.weights)
serum_creatinine         0.07763039181358414
age                      0.062322493078632475
creatinine_phosphokinase 0.026406997937481715
high_blood_pressure      0.0121666389431587
diabetes                 0.00884144057118402
smoking                  -0.002904035263979059
```



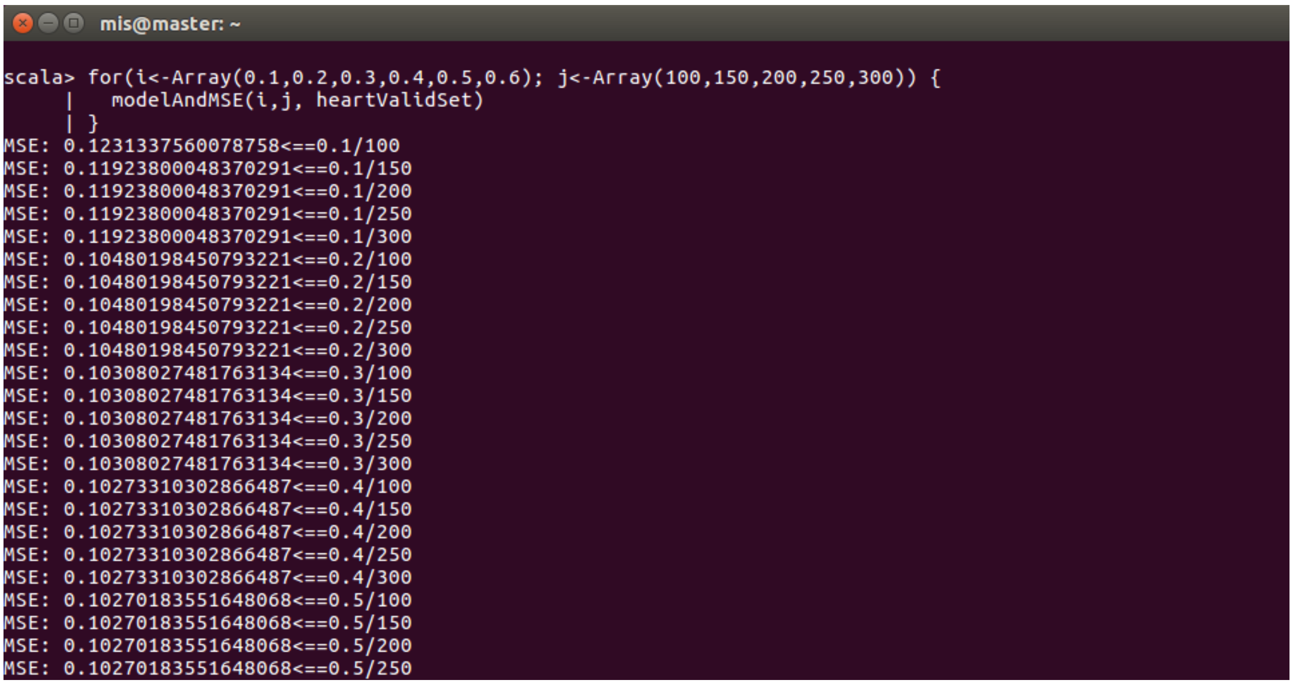
```
scala> printWeights(heartModel.weights)
serum_creatinine         0.07763039181358414
age                      0.062322493078632475
creatinine_phosphokinase 0.026406997937481715
high_blood_pressure      0.0121666389431587
diabetes                 0.00884144057118402
smoking                  -0.002904035263979059
platelets                -0.005024070891089872
anaemia                  -0.010363033957126044
serum_sodium             -0.030071342377087043
sex                      -0.03098151872337847
ejection_fraction        -0.08914726339228686
time                     -0.18083004200203062

scala> val heartPred=heartModel.predict(heartValidSet.map(x=>x.features))
heartPred: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[290] at mapPartitions at GeneralizedLinearA
lgorithm.scala:69

scala> heartPred.first
res14: Double = 0.8403665752548327

scala> heartValidSet.first
res15: org.apache.spark.mllib.regression.LabeledPoint = (1.0,[2.031651527399793,-0.8696468582568094,-0.575
9522261296451,-0.8461608356372154,0.5844090437045386,-0.7344569121685459,0.9881164030309463,1.552542074833
1058,-1.0482587813127016,0.7344569121685459,-0.6865309732064924,-1.3175534772023767])

scala> def getMSE(model: org.apache.spark.mllib.regression.LinearRegressionModel, dataset: org.apache.spar
```



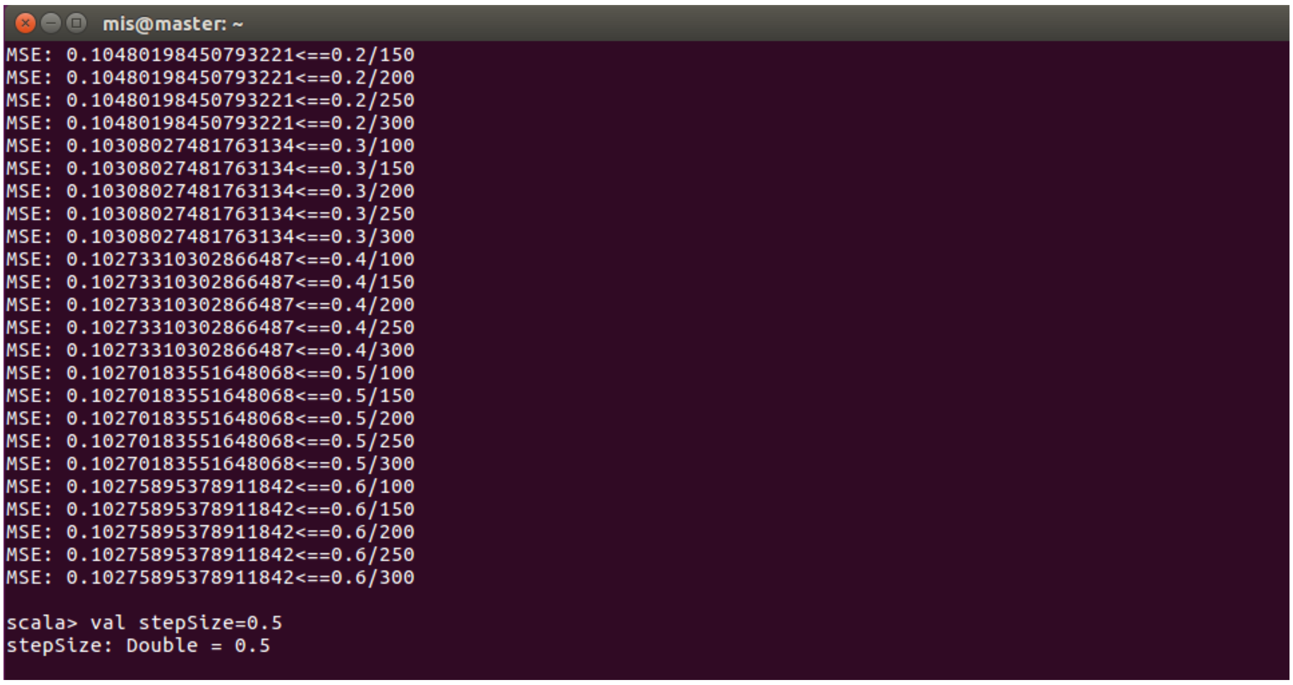
```
scala> def getMSE(model: org.apache.spark.mllib.regression.LinearRegressionModel, dataset: org.apache.spar
k.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint])={
     |     val heartReal=dataset.map(x=>x.label)  // 取得真實評分
     |     val heartPred=model.predict(dataset.map(x=>x.features))  // 預測評分
     |     val realWithPred=heartReal.zip(heartPred)  // 真實與預測分數合併
     |     val MSE=realWithPred.map{case (real, pred)=>math.pow(real-pred,2)}.mean()
     |     MSE // 最後結果MSE
     | }
getMSE: (model: org.apache.spark.mllib.regression.LinearRegressionModel, dataset: org.apache.spark.rdd.RDD
[org.apache.spark.mllib.regression.LabeledPoint])Double

scala> val MSE=getMSE(heartModel, heartValidSet)
MSE: Double = 0.11923800048370291

scala> println("MSE: "+ MSE)
MSE: 0.11923800048370291

scala> def modelAndMSE(stepSize:Double, numOfIter:Int, dataset: org.apache.spark.rdd.RDD[org.apache.spark.
mllib.regression.LabeledPoint]) {
     | // 訓練模型
     | val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)
     | heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)
     | val heartModel=heartModelInst.run(heartTrainSet)
     |
     | // 求預測值
     |   val heartPred=heartModel.predict(dataset.map(x=>x.features))
     |   val heartReal=dataset.map(x=>x.label)
```



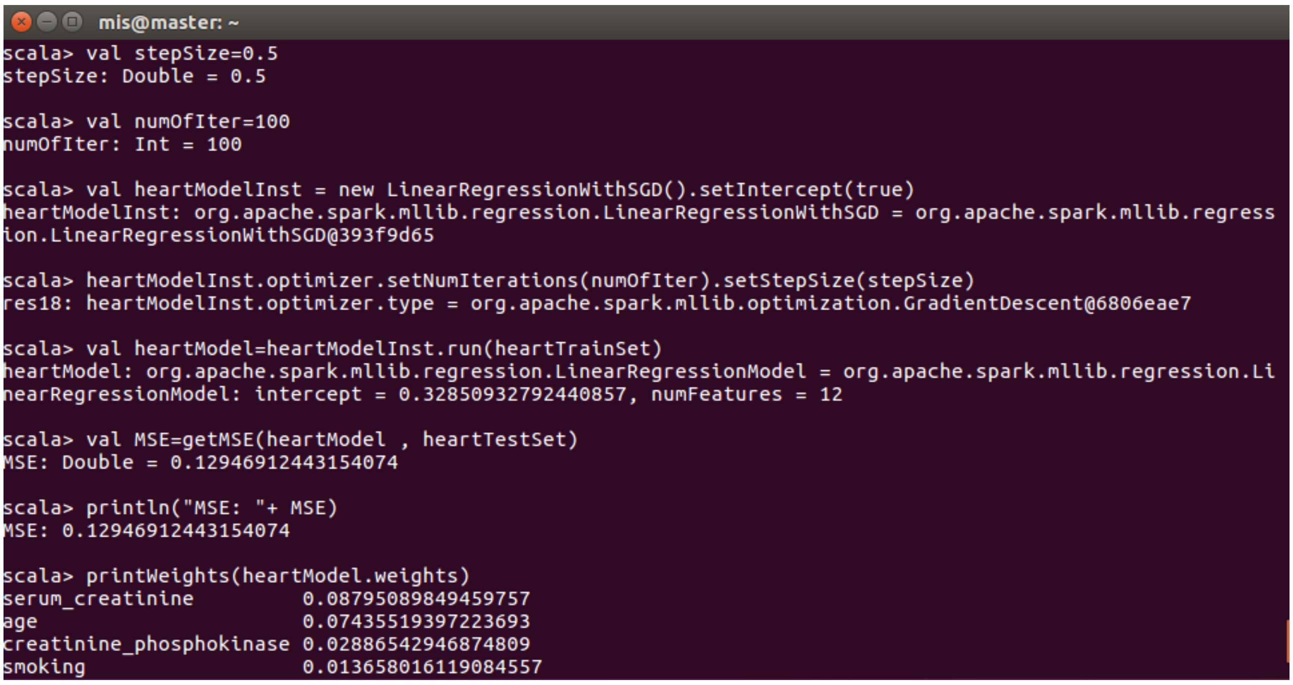
```
scala> def modelAndMSE(stepSize:Double, numOfIter:Int, dataset: org.apache.spark.rdd.RDD[org.apache.spark.
mllib.regression.LabeledPoint]) {
     | // 訓練模型
     | val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)
     | heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)
     | val heartModel=heartModelInst.run(heartTrainSet)
     |
     | // 求預測值
     |   val heartPred=heartModel.predict(dataset.map(x=>x.features))
     |   val heartReal=dataset.map(x=>x.label)
     |
     | // 計算MSE
     | val MSE=getMSE(heartModel, dataset)
     | println("MSE: "+ MSE+"<=="+stepSize+"/"+numOfIter)
     |
     | }
modelAndMSE: (stepSize: Double, numOfIter: Int, dataset: org.apache.spark.rdd.RDD[org.apache.spark.mllib.r
egression.LabeledPoint])Unit

scala> for(i<-Array(0.1,0.2,0.3,0.4,0.5,0.6); j<-Array(100,150,200,250,300)) {
     |   modelAndMSE(i,j, heartValidSet)
     | }
MSE: 0.1231337560078758<==0.1/100
MSE: 0.11923800048370291<==0.1/150
MSE: 0.11923800048370291<==0.1/200
MSE: 0.11923800048370291<==0.1/250
MSE: 0.11923800048370291<==0.1/300
```



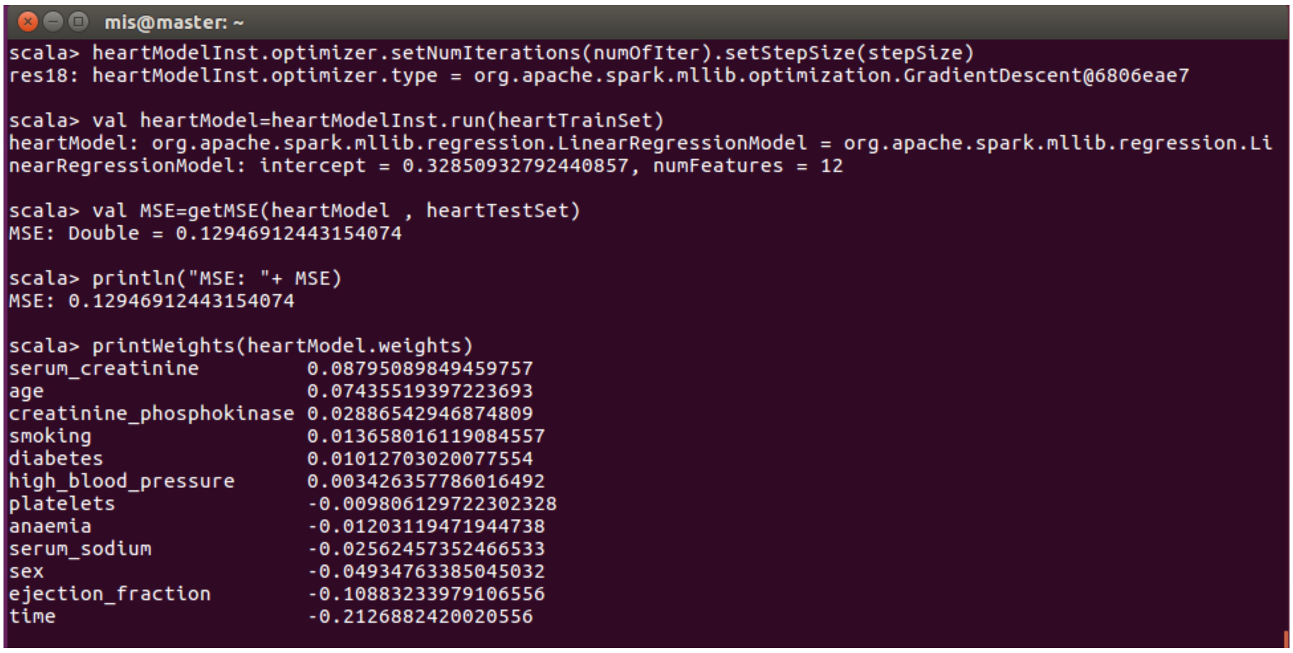
```
scala> for(i<-Array(0.1,0.2,0.3,0.4,0.5,0.6); j<-Array(100,150,200,250,300)) {
     |   modelAndMSE(i,j, heartValidSet)
     | }
MSE: 0.123133756007875 8<==0.1/100
MSE: 0.11923800048370291<==0.1/150
MSE: 0.11923800048370291<==0.1/200
MSE: 0.11923800048370291<==0.1/250
MSE: 0.11923800048370291<==0.1/300
MSE: 0.10480198450793221<==0.2/100
MSE: 0.10480198450793221<==0.2/150
MSE: 0.10480198450793221<==0.2/200
MSE: 0.10480198450793221<==0.2/250
MSE: 0.10480198450793221<==0.2/300
MSE: 0.10308027481763134<==0.3/100
MSE: 0.10308027481763134<==0.3/150
MSE: 0.10308027481763134<==0.3/200
MSE: 0.10308027481763134<==0.3/250
MSE: 0.10308027481763134<==0.3/300
MSE: 0.10273310302866487<==0.4/100
MSE: 0.10273310302866487<==0.4/150
MSE: 0.10273310302866487<==0.4/200
MSE: 0.10273310302866487<==0.4/250
MSE: 0.10273310302866487<==0.4/300
MSE: 0.10270183551648068<==0.5/100
MSE: 0.10270183551648068<==0.5/150
MSE: 0.10270183551648068<==0.5/200
MSE: 0.10270183551648068<==0.5/250
```



```
MSE: 0.10480198450793221<==0.2/150
MSE: 0.10480198450793221<==0.2/200
MSE: 0.10480198450793221<==0.2/250
MSE: 0.10480198450793221<==0.2/300
MSE: 0.10308027481763134<==0.3/100
MSE: 0.10308027481763134<==0.3/150
MSE: 0.10308027481763134<==0.3/200
MSE: 0.10308027481763134<==0.3/250
MSE: 0.10308027481763134<==0.3/300
MSE: 0.10273310302866487<==0.4/100
MSE: 0.10273310302866487<==0.4/150
MSE: 0.10273310302866487<==0.4/200
MSE: 0.10273310302866487<==0.4/250
MSE: 0.10273310302866487<==0.4/300
MSE: 0.10270183551648068<==0.5/100
MSE: 0.10270183551648068<==0.5/150
MSE: 0.10270183551648068<==0.5/200
MSE: 0.10270183551648068<==0.5/250
MSE: 0.10270183551648068<==0.5/300
MSE: 0.10275895378911842<==0.6/100
MSE: 0.10275895378911842<==0.6/150
MSE: 0.10275895378911842<==0.6/200
MSE: 0.10275895378911842<==0.6/250
MSE: 0.10275895378911842<==0.6/300

scala> val stepSize=0.5
stepSize: Double = 0.5
```



```
scala> val stepSize=0.5
stepSize: Double = 0.5

scala> val numOfIter=100
numOfIter: Int = 100

scala> val heartModelInst = new LinearRegressionWithSGD().setIntercept(true)
heartModelInst: org.apache.spark.mllib.regression.LinearRegressionWithSGD = org.apache.spark.mllib.regress
ion.LinearRegressionWithSGD@393f9d65

scala> heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)
res18: heartModelInst.optimizer.type = org.apache.spark.mllib.optimization.GradientDescent@6806eae7

scala> val heartModel=heartModelInst.run(heartTrainSet)
heartModel: org.apache.spark.mllib.regression.LinearRegressionModel = org.apache.spark.mllib.regression.Li
nearRegressionModel: intercept = 0.32850932792440857, numFeatures = 12

scala> val MSE=getMSE(heartModel , heartTestSet)
MSE: Double = 0.12946912443154074

scala> println("MSE: "+ MSE)
MSE: 0.12946912443154074

scala> printWeights(heartModel.weights)
serum_creatinine         0.08795089849459757
age                      0.07435519397223693
creatinine_phosphokinase 0.02886542946874809
smoking                  0.013658016119084557
```



```
scala> heartModelInst.optimizer.setNumIterations(numOfIter).setStepSize(stepSize)
res18: heartModelInst.optimizer.type = org.apache.spark.mllib.optimization.GradientDescent@6806eae7

scala> val heartModel=heartModelInst.run(heartTrainSet)
heartModel: org.apache.spark.mllib.regression.LinearRegressionModel = org.apache.spark.mllib.regression.Li
nearRegressionModel: intercept = 0.32850932792440857, numFeatures = 12

scala> val MSE=getMSE(heartModel , heartTestSet)
MSE: Double = 0.12946912443154074

scala> println("MSE: "+ MSE)
MSE: 0.12946912443154074

scala> printWeights(heartModel.weights)
serum_creatinine         0.08795089849459757
age                      0.07435519397223693
creatinine_phosphokinase 0.02886542946874809
smoking                  0.013658016119084557
diabetes                 0.01012703020077554
high_blood_pressure      0.003426357786016492
platelets                -0.009806129722302328
anaemia                  -0.01203119471944738
serum_sodium             -0.02562457352466533
sex                      -0.04934763385045032
ejection_fraction        -0.10883233979106556
time                     -0.2126882420020556
```