Assignment 2

Dr. Dan Lo

Terry Strickland

Fall Semester 2020 🕵

Objective

Write a Scala program that take **covtype.data** file as **input**
and **output** the "prediction" of the "type of forest" on a piece of land

Input Data

- Download the data here
- Read the **covtype.info**

About Input Data

This 'data set' recorded the TYPES OF FOREST that are covering plots of lands in Colorado, USA.
Each 'sample' of the data set looks like this:

`2596,51,3,258,0,510,221,232,148,6279,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,5`

this is one line in the 'covtype.dat' file, each line is a 'sample'
a 'sample' has 55 columns: 0 to 54
This 'data set' has total of 581012 rows (samples): 0 to 581011

column 0 to 9 has the following 'attribute'

```
 #   Column                              Non-Null Count   Dtype
---  ------                              --------------   -----
 0   Elevation                           581012 non-null  int64
 1   Aspect                              581012 non-null  int64
 2   Slope                               581012 non-null  int64
 3   Horizontal_Distance_To_Hydrology    581012 non-null  int64
 4   Vertical_Distance_To_Hydrology      581012 non-null  int64
 5   Horizontal_Distance_To_Roadways     581012 non-null  int64
 6   Hillshade_9am                       581012 non-null  int64
 7   Hillshade_Noon                      581012 non-null  int64
 8   Hillshade_3pm                       581012 non-null  int64
 9   Horizontal_Distance_To_Fire_Points  581012 non-null  int64
```

column 10 to 11, each of the column represent a binary digit 0 or 1, so all 4 columns together represents a number in 4 bit binary, 0 0 0 1 is 1
This technique has a name that is very descriptive and easy to understand, ONE HOT encoding

```
 10  Wilderness_Area1                        581012 non-null  int64
 11  Wilderness_Area2                        581012 non-null  int64
 12  Wilderness_Area3                        581012 non-null  int64
 13  Wilderness_Area4                        581012 non-null  int64
```

columns 14 to 53 is also the ONE HOT encoding columns for the Soil_Type

```
 14  Soil_Type1                              581012 non-null  int64
 15  Soil_Type2                              581012 non-null  int64
 16  Soil_Type3                              581012 non-null  int64
 17  Soil_Type4                              581012 non-null  int64
    ...
```

the last column, column 54, is the CORRECT ANSWER for each row (sample), the 'Type of Forest' that are covering the land, and its attribute values are in column 0 to 53

```
 54  Cover_Type                              581012 non-null  int64
```

There are 7 possible answer for column 54

```
Forest Cover Type Classes:      1 -- Spruce/Fir
                                2 -- Lodgepole Pine
                                3 -- Ponderosa Pine
                                4 -- Cottonwood/Willow
                                5 -- Aspen
                                6 -- Douglas-fir
                                7 -- Krummholz
```

You can read the 'covtype.info' for a lot more details related to the 'data set'

Output Data

There are 7 possible answer for column 54

```
Forest Cover Type Classes:      1 -- Spruce/Fir
                                2 -- Lodgepole Pine
                                3 -- Ponderosa Pine
                                4 -- Cottonwood/Willow
                                5 -- Aspen
```

```
                                6 -- Douglas-fir
                                7 -- Krummholz
```

"confusionMatrix"
" the correct predictions are the counts along the diagonal"

For each row (sample), we are using the simple DECISION TREE algorithm to predicts the 'TYPE OF FOREST' (1 to 7).
First we split the 'data set' into 3 different sets:

- 'training set'
- 'cross validation set'
- 'test set'

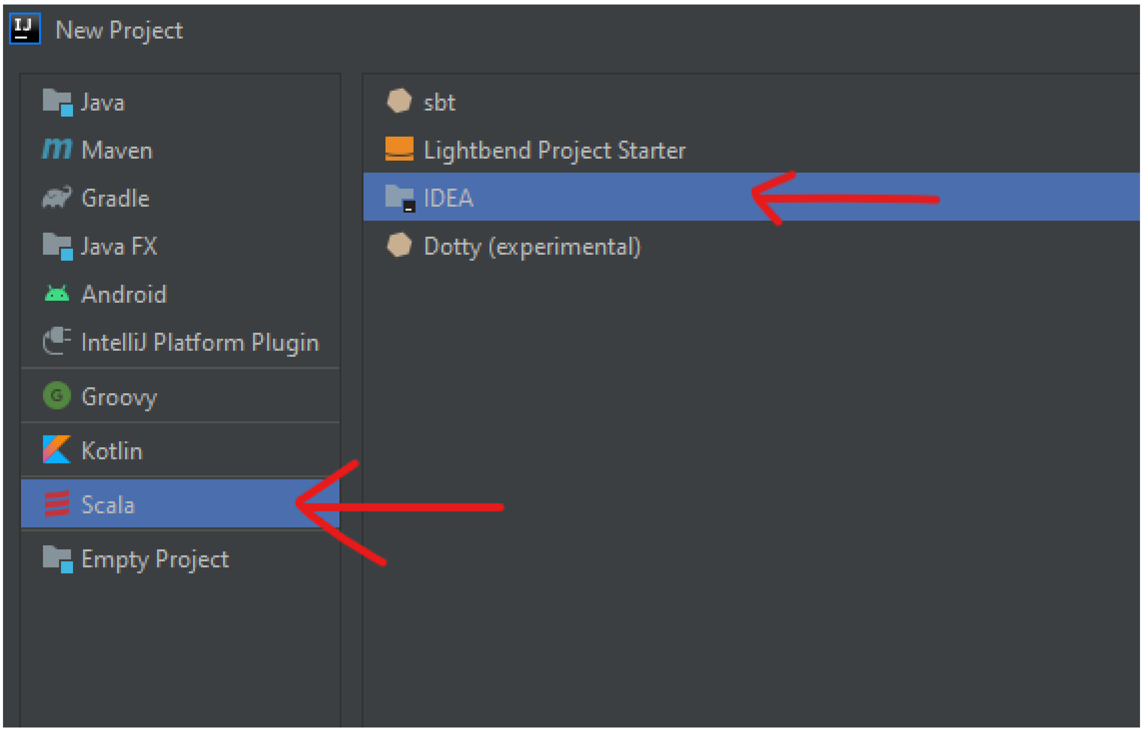
We input 'training set' and 'cross validation set' into the algorithm to 'train' it, after that we input the 'test set' into the algorithm and see how many correct answers the algorithm gives us.

JDK, SDK requirements, Setup

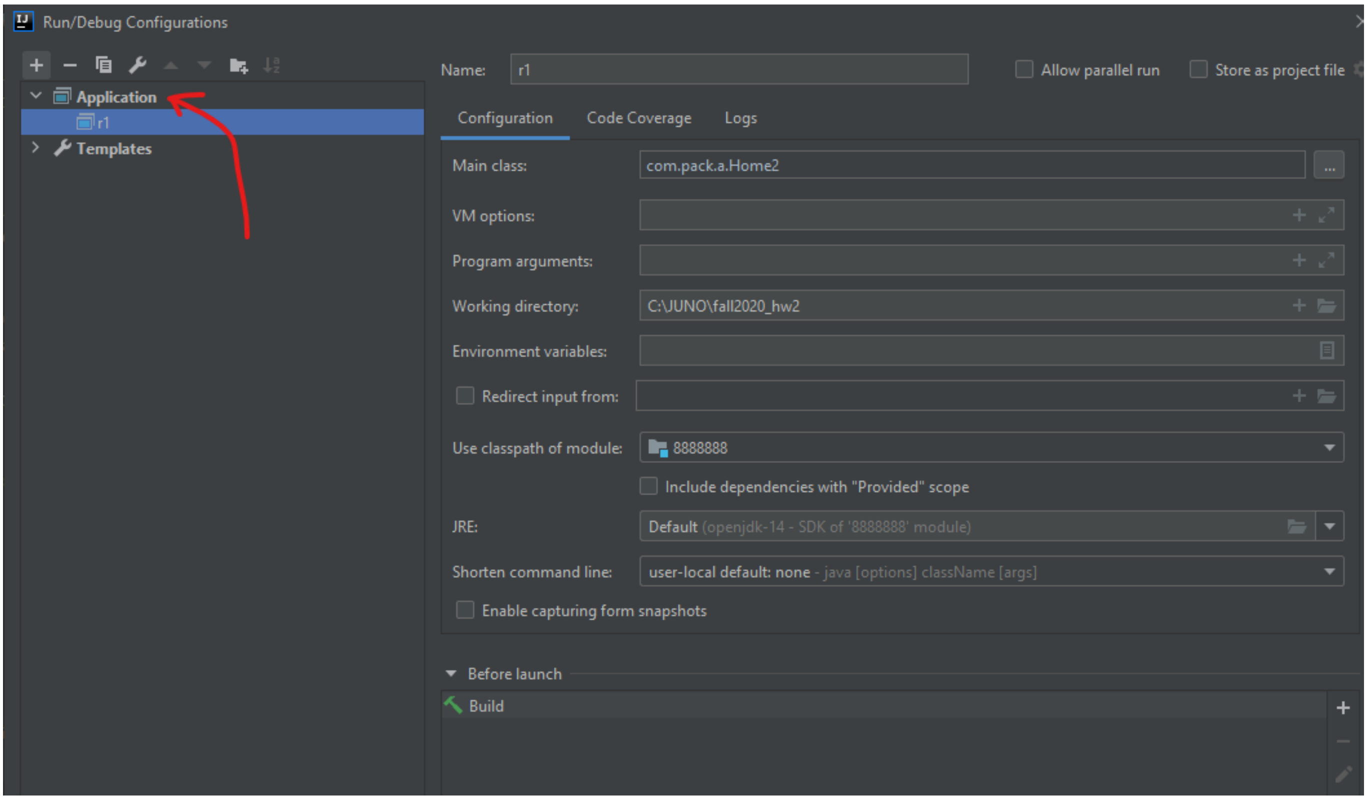
```
C:/Users/ <username> /.jdks/openjdk-14.0.2-1
```

- openjdk-14.0.2-1
- IntelliJ community edition 2020.2.1
- scala-sdk-2.12.10

intelli J Set up

New Project >> Scala (IDEA** based Scala project)


intelli J, run configuration



Source code

Source code on Github click HERE

```
package com.pack.a

import org.apache.log4j.{Level, Logger}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.{DecisionTree, RandomForest}
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

object Home2 {

  def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]):
MulticlassMetrics = {
    val predictionsAndLabels = data.map(example =>
      (model.predict(example.features), example.label)
    )

    new MulticlassMetrics(predictionsAndLabels)

  } // def getMetrics()


  // main() function
```

```
  def main( args: Array[String] ) {

    // set the log level
    Logger.getLogger("org").setLevel(Level.ERROR)

    // make new 'sc' object thing
    val sc = new SparkContext( new
SparkConf().setAppName("RDF").setMaster("local") )

    // read the file
    val rawData = sc.textFile("./covtype.data")
//    rawData.foreach(println)

    val data = rawData.map { line =>
      val values = line.split(',').map(_.toDouble)
      val featureVector = Vectors.dense(values.init)
      val label = values.last - 1
      LabeledPoint(label, featureVector)
    }

    // Split into 80% train, 10% cross validation, 10% test
    val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8, 0.1,
0.1))

    // "cache data to RAM"
    trainData.cache()
    cvData.cache()
//    testData.cache()

    println("\ntotal data count: " + data.count())

    println("\ntotal trainData count: " + trainData.count() )

    println("\ntotal testData count: " + testData.count() )

    println("\ntotal CV count: " + cvData.count() )


    // Build a simple default DecisionTreeModel and compute precision and recall
    //    simpleDecisionTree(trainData, cvData)
    val model = DecisionTree.trainClassifier(trainData, 7, Map[Int,Int](), "gini",
4, 100)
    val metrics = getMetrics(model, cvData)
    var sum1 = 0.0

    println("\nPrinting the PRECISION VALUE for each 'Class'")

    for ( asdf <-  0 to 6) {  // we have total of 7 'classes'
      sum1 += metrics.precision(asdf)

      println("\nclass " + asdf)

//      print("Printing the metrics.accuracy: ")
//      println(metrics.accuracy)
```

```
      println("precision value: " + metrics.precision(asdf) )
    }

    println("\nPrinting the CONFUSION MATRIX")
    println(metrics.confusionMatrix)

    println("\n")
    def lalala( inputData: RDD[LabeledPoint] ): Array[Double] = {

      val numOfSampleOutofTotal = inputData.map(_.label).countByValue()
      // showing how many 'samples' each 'class (1 to 7) there are
      numOfSampleOutofTotal.foreach(println)

//      val count1 = numOfSampleOutofTotal.toArray.sortBy(_._1).map(_._2)
      val count1 = numOfSampleOutofTotal.toArray.sortBy(asdf => asdf._1).map(
asdf2 => asdf2._2)
      count1.foreach(println)

      count1.map(_.toDouble / count1.sum)
    }

    val probForsetCV = lalala(cvData)

    val value1 = for (asdf <- 0 to 6) yield {
      probForsetCV(asdf) * metrics.precision(asdf)
    }

    println("\nCALCULATED OVERALL PRECISION: " + value1.sum)

    println("\nPrinting the metrics.weightedPrecision")
    println(metrics.weightedPrecision)

//    println("\nPrinting the SUM of OVERALL PRECISION")
//    println(sum1)


    // remove data from RAM?
    trainData.unpersist()
    cvData.unpersist()
//    testData.unpersist()

    println("\nMain function() finished running, yay!")

  } // def main()


} // Object Home2
```

Outputs and Screenshots 👍

What type of forest is on this piece of land?
Output also includes the "CONFUSION MATRIX"

The 'precision value' ranges from 0 to 1

- 0 means no precision at all
- 1 means 100% precision

You want the 'precision value' to be as close to 1 as possible
For this 'data set' we want to know how accurate the 'decision tree' algorithm will predict the answer

Remember, the 'answers' for the this 'data set' is the TYPE OF FOREST

```
Forest Cover Type Classes:      1 -- Spruce/Fir
                                2 -- Lodgepole Pine
                                3 -- Ponderosa Pine
                                4 -- Cottonwood/Willow
                                5 -- Aspen
                                6 -- Douglas-fir
                                7 -- Krummholz
```

Each 'TYPE OF FOREST' is also can be referred to as the 'CLASS'
Yes, let keep things confusing and convoluted by adding words.

So, looking at the output of the program, we should get SEVEN numbers range from 0 to 1
BUT, our 'TYPE OF FOREST' ranges from 1 to 7

- so 'class 0' will refers to 'FOREST TYPE 1'
- 'class 1' will refers to 'FOREST TYPE 2'
- etc

Interpretation of the answer

So we can see below that when you feed the 'sample' to the 'decision tree' algorithm; you can see the output below.

This is the updated version, because an updated announcement was posted ON THE SAME DAY AS THE DUE DATE for what you wanted for the assignment; thus I DID NOT HAVE ENOUGH TIME to figure out how to get the answer, and I was NOT SLACKING; I turned in the original assignment 2 DAYS BEFORE THE DUE DATE, but the INSTRUCTION WASN'T CLEAR and there aren't much documentations online for Scala!

Please forgive me for being a little upset, because I put in a lot of effort into my work but then I won't be getting a credit for it because the instructions provided to me WASN'T CLEAR.

```
total data count: 581012

total trainData count: 464839

total testData count: 57934
```

```
total CV count: 58239

Printing the PRECISION VALUE for each 'Class'

class 0
precision value: 0.6831815364074197

class 1
precision value: 0.7266184707568061

class 2
precision value: 0.6270841805612037

class 3
precision value: 0.33986928104575165

class 4
precision value: 1.0

class 5
precision value: 0.0

class 6
precision value: 0.6917293233082706

Printing the CONFUSION MATRIX
14327.0  6546.0   10.0    0.0    0.0  0.0  376.0
5518.0   22313.0  441.0   30.0   0.0  0.0  34.0
0.0      439.0    3084.0  88.0   0.0  0.0  0.0
0.0      1.0      164.0   104.0  0.0  0.0  0.0
0.0      920.0    31.0    0.0    6.0  0.0  0.0
0.0      458.0    1188.0  84.0   0.0  0.0  0.0
1126.0   31.0     0.0     0.0    0.0  0.0  920.0


(0.0,21259)
(5.0,1730)
(1.0,28336)
(6.0,2077)
(2.0,3611)
(3.0,269)
(4.0,957)
21259
28336
3611
269
957
1730
2077

CALCULATED OVERALL PRECISION: 0.6844685672310528

Printing the metrics.weightedPrecision
```

```
0.6844685672310529

Main function() finished running, yay!

Process finished with exit code 0
```