Lesson 5 Practice Hands-On

Directions

This Hands-On will **not** be graded, but you are encouraged to complete it. The best way to become a great data scientist is to practice. Once you have submitted your project, you will be able to access the solution on the next page. Note that the solution will be slightly different from yours, but should look similar.

Caution!

Do not submit your project until you have completed all requirements, as you will not be able to resubmit.

Description

Using **this data on boardgame ratings**, perform a decision tree to predict the average rating of boardgames (`average_rating`). You will need to upload this data file to your HDFS.

Please copy your Scala code into a text file, and include at the bottom the answer to the following questions:

- What was the best model after hyperparameter tuning?
- What is the overall accuracy?

Alternative Assignment if You Can't Run Hadoop and/or Ambari

If your computer refuses to run Hadoop and/or Ambari, **here** is an alternative exam to test your understanding of the material. Please attach it instead.

Lesson 5 Practice Hands-On Solution

Best Model After Hyperparameter Tuning

The best model after hyperparamter tuning was the one that used entropy, a max of 300 bins, a depth of 20, and .05 of minimum information gain.

Overall Accuracy

The overall accuracy was 62% with the best hyperparamter model.

Code

Below you will find all the code to provide the answers above.

Done in the Command Prompt

```
export SPARK_MAJOR_VERSION=2

spark-shell --master local[*]
```

Done in Spark Shell

```
val data = spark.read.
option("inferSchema", true).
option("header", true).
csv("hdfs:///user/maria_dev/boardgames3.csv")

data.printSchema()

val Array(trainData, testData) = data.randomSplit(Array(0.9,
0.1))
trainData.cache()
testData.cache()

import org.apache.spark.ml.feature.VectorAssembler
val inputCols = trainData.columns.filter(_ != "average_rating")
val assembler = new VectorAssembler().
setInputCols(inputCols).
setOutputCol("featureVector")
val assembledTrainData = assembler.transform(trainData)
assembledTrainData.select("featureVector").show(truncate =
false)

import org.apache.spark.ml.classification.DecisionTreeClassifier
import scala.util.Random
```

```
val classifier = new DecisionTreeClassifier().
setSeed(Random.nextLong()).
setLabelCol("average_rating").
setFeaturesCol("featureVector").
setPredictionCol("prediction")
val model = classifier.fit(assembledTrainData)
println(model.toDebugString)

model.featureImportances.toArray.zip(inputCols).
sorted.reverse.foreach(println)

//The best feature for prediction is users_rated.

val predictions = model.transform(assembledTrainData)
predictions.select("average_rating", "prediction",
"probability").
show(truncate = false)

//Eyeball analysis shows that right now it is not predicting
very well.

import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
val evaluator = new MulticlassClassificationEvaluator().
setLabelCol("average_rating").
setPredictionCol("prediction")
evaluator.setMetricName("accuracy").evaluate(predictions)

//Right now the model has 57% accuracy.

val confusionMatrix = predictions.
groupBy("average_rating").
pivot("prediction", (0 to 10)).
count().
na.fill(0.0).
orderBy("average_rating")
confusionMatrix.show()

//Looks like most ratings, both higher and lower, are getting
misclassified as 5s, 6s, and 7s. There were no accurate
predictions of 1-4 or 8-10.
```

```
import org.apache.spark.sql.DataFrame
def classProbabilities(data: DataFrame): Array[Double] = {
val total = data.count()
data.groupBy("average_rating").count().
orderBy("average_rating").
select("count").as[Double].
map(_ / total).
collect()
}

val trainPriorProbabilities = classProbabilities(trainData)
val testPriorProbabilities = classProbabilities(testData)
trainPriorProbabilities.zip(testPriorProbabilities).map {
case (trainProb, cvProb) => trainProb * cvProb
}.sum

//Random guessing accuracy is 18%, so the current model is
better than just guessing.  Yay!

import org.apache.spark.ml.Pipeline
val inputCols = trainData.columns.filter(_ != "average_rating")
val assembler = new VectorAssembler().
setInputCols(inputCols).
setOutputCol("featureVector")
val classifier = new DecisionTreeClassifier().
setSeed(Random.nextLong()).
setLabelCol("average_rating").
setFeaturesCol("featureVector").
setPredictionCol("prediction")
val pipeline = new Pipeline().setStages(Array(assembler,
classifier))

import org.apache.spark.ml.tuning.ParamGridBuilder
val paramGrid = new ParamGridBuilder().
addGrid(classifier.impurity, Seq("gini", "entropy")).
addGrid(classifier.maxDepth, Seq(1, 20)).
addGrid(classifier.maxBins, Seq(40, 300)).
addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).
build()
val multiclassEval = new MulticlassClassificationEvaluator().
```

```
setLabelCol("average_rating").
setPredictionCol("prediction").
setMetricName("accuracy")

import org.apache.spark.ml.tuning.TrainValidationSplit
val validator = new TrainValidationSplit().
setSeed(Random.nextLong()).
setEstimator(pipeline).
setEvaluator(multiclassEval).
setEstimatorParamMaps(paramGrid).
setTrainRatio(0.9)
val validatorModel = validator.fit(trainData)

import org.apache.spark.ml.PipelineModel
val bestModel = validatorModel.bestModel
bestModel.asInstanceOf[PipelineModel].stages.last.extractParamMa
p

//The best model uses entropy, a max of 300 bins, a depth of 20,
and .05 of minimum information gain.

val validatorModel = validator.fit(trainData)
val paramsAndMetrics = validatorModel.validationMetrics.
zip(validatorModel.getEstimatorParamMaps).sortBy(-_._1)
paramsAndMetrics.foreach { case (metric, params) =>
println(metric)
println(params)
println()
}

validatorModel.validationMetrics.max
multiclassEval.evaluate(bestModel.transform(testData))
```

Lesson 5 Practice Hands-On Solution - Alternative Assignment

This exam serves as the assessment for those students who cannot utilize the Hadoop system and/or Ambari GUI. Correct answers are show in bold.

1. Which of the Spark components most interests you and why?
   Spark ML seems so powerful! It also seems the most like "regular" programming that isn't done with big data, which makes it a little easier.
2. True or False? "Zeppelin is very similar in structure and function to Jupyter Notebook." **a. True** b. False
3. How do the three types of Spark data storage differ from each other?
   RDDs are the original way to store data, but they are slow. DataSets are more efficient. DataFrames are meant specially for relational data and hold row and column data.
4. How do you denote comments in Scala? a. # b. Change it to markdown **c. //** d. /#
5. What are the four hyperparameters for decision trees? Give both names and descriptions.
   Maximum Depth: The number of decisions you can make in a tree.
   Maximum Bins: The number of decision rules you can use in a tree.
   Impurity Measure: How much mix-up you allow between categories.
   MInimum Information Gain: Keep only things that add to the accuracy of your data.