UNIVERSIDAD DE GRANADA

Escuela de Posgrado Universidad de Granada

MLlib Ensembles

Big Data II

Diego García (djgarcia@decsai.ugr.es) 10/04/2018

Outline

- **Introduction**
- Decision Tree
- Random Forest
- PCARD
- Metrics

Introduction

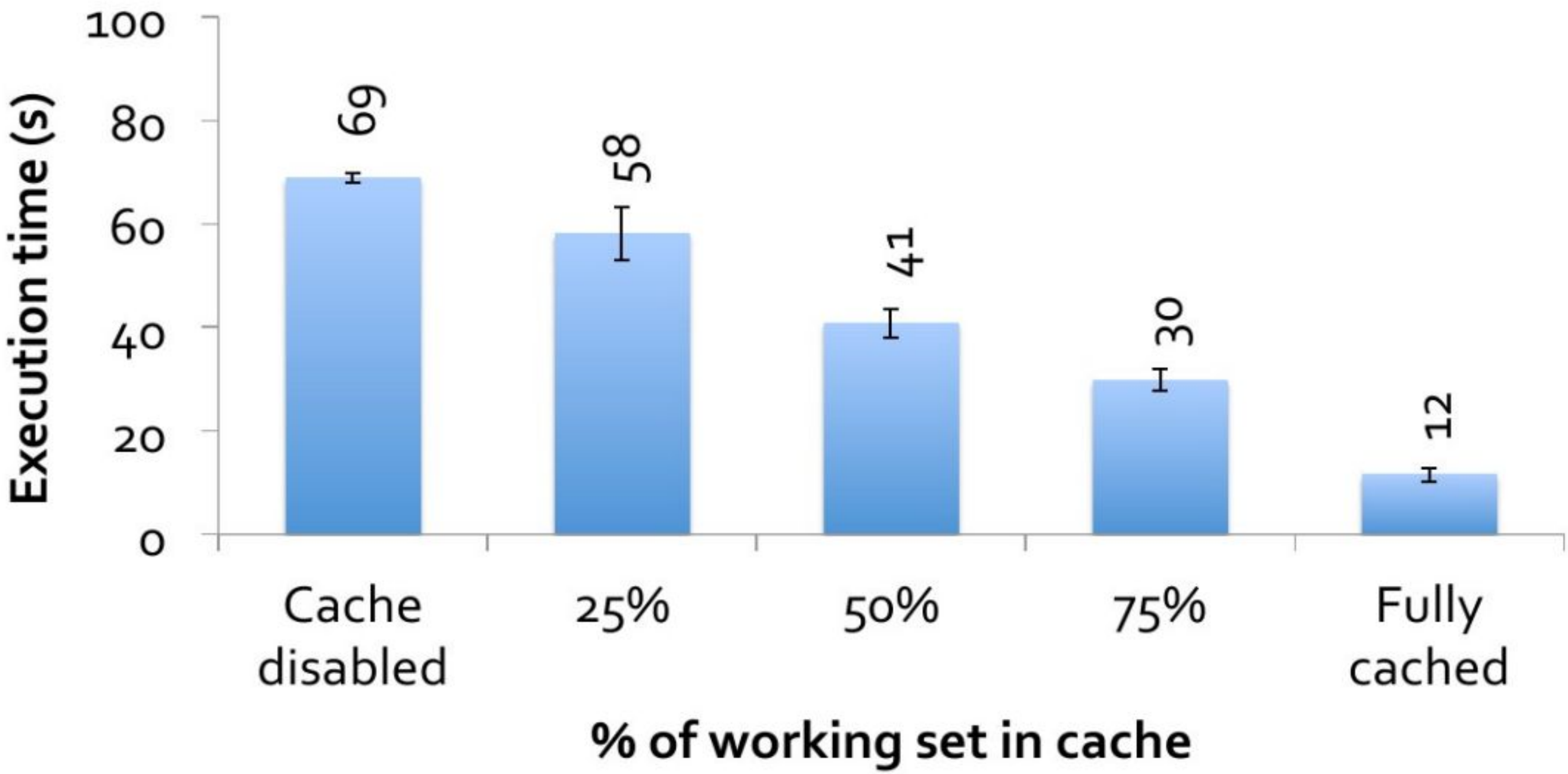
- LabeledPoint:
  - Main MLlib data type
  - Local vector with a label
  - Only double values

(label, features)

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint

val pos = LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0))
```

Cache



Execution time (s)
100
80
60
40
20
0
69
58
41
30
12
Cache disabled
25%
50%
75%
Fully cached
% of working set in cache

Cache

RDD.cache() or RDD.persist()

The first time it is computed in an action, it will be kept in memory on the nodes

Spark automatically persists some intermediate data in certain operations

RDD.persist(level)

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

Upload files to cluster

To upload the file archivo.txt from our computer to /home/user folder, we do the following:

scp archivo.txt user@hadoop.ugr.es.com:/home/user

Download files from cluster

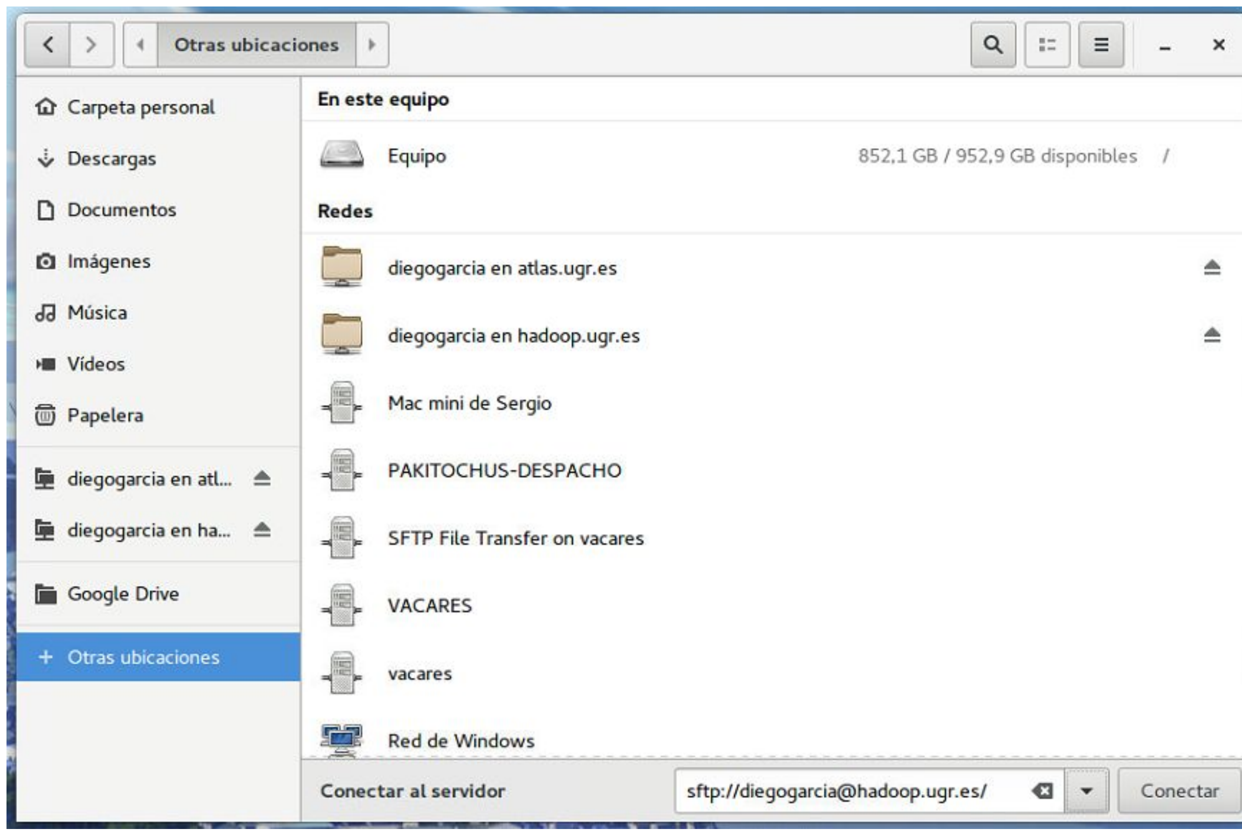
To download the file archivo.txt from the cluster to our computer in Docs folder, we do the following:

```
scp user@hadoop.ugr.es:/home/user/archivo.txt Docs
```

Alternatives

FileZilla

Nautilus



Run jobs in cluster

Always use **spark-submit** in cluster

/opt/spark-2.2.0/bin/spark-submit

Limit resources (workers/memory)

/opt/spark-2.2.0/bin/spark-submit --total-executor-cores 10 --executor-memory 10g

Load data

```
/opt/spark-2.2.0-bin-hadoop2.7/bin/spark-shell --packages djgg:PCARD:1.3
```

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.{Vector, Vectors}

//Load train and test
val pathTrain = "file:////home/spark/datasets/susy-10k-tra.data"
val rawDataTrain = sc.textFile(pathTrain)

val pathTest = "file:////home/spark/datasets/susy-10k-tst.data"
val rawDataTest = sc.textFile(pathTest)
```

Train and Test RDDs

```
val train = rawDataTrain.map{line =>
  val array = line.split(",")
  var arrayDouble = array.map(f => f.toDouble)
  val featureVector = Vectors.dense(arrayDouble.init)
  val label = arrayDouble.last
  LabeledPoint(label, featureVector)
}.persist

val test = rawDataTest.map { line =>
  val array = line.split(",")
  var arrayDouble = array.map(f => f.toDouble)
  val featureVector = Vectors.dense(arrayDouble.init)
  val label = arrayDouble.last
  LabeledPoint(label, featureVector)
}.persist
```

Check train and test

```
train.count
train.first

test.count
test.first


//Class balance

val classInfo = train.map(lp => (lp.label, 1L)).reduceByKey(_ + _).collectAsMap()
```

1.0 -> 4907, 0.0 -> 5093

Outline

- Introduction
- **Decision Tree**
- Random Forest
- PCARD
- Metrics

Decision Tree

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel

// Train a DecisionTree model.
//  Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val model = DecisionTree.trainClassifier(train, numClasses, categoricalFeaturesInfo, impurity,
maxDepth, maxBins)
```

Decision Tree

```
// Evaluate model on test instances and compute test error
val labelAndPreds = test.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val testAcc = 1 - labelAndPreds.filter(r => r._1 != r._2).count().toDouble / test.count()
println(s"Test Accuracy = $testAcc")
```

Test Accuracy = 0.7876

Outline

- Introduction
- Decision Tree
- **Random Forest**
- PCARD
- Metrics

Random Forest

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel

// Train a RandomForest model.
// Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 100
val featureSubsetStrategy = "auto" // Let the algorithm choose.
val impurity = "gini"
val maxDepth = 4
val maxBins = 32

val model = RandomForest.trainClassifier(train, numClasses, categoricalFeaturesInfo, numTrees,
featureSubsetStrategy, impurity, maxDepth, maxBins)
```

Random Forest

```
// Evaluate model on test instances and compute test error
val labelAndPreds = test.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val testAcc = 1 - labelAndPreds.filter(r => r._1 != r._2).count.toDouble / test.count()
println(s"Test Accuracy = $testAcc")
```

Test Accuracy: 0.7920

Outline

- Introduction
- Decision Tree
- Random Forest
- **PCARD**
- Metrics

PCARD

```
import org.apache.spark.mllib.tree.PCARD

val cuts = 5
val trees = 10

val pcardTrain = PCARD.train(train, trees, cuts)

val pcard = pcardTrain.predict(test)
```

PCARD

```
def avgAcc(labels: Array[Double], predictions: Array[Double]): (Double, Double) = {
  var cont = 0
  for (i <- labels.indices) {
    if (labels(i) == predictions(i)) {
      cont += 1
    }
  }
  (cont / labels.length.toFloat, 1 - cont / labels.length.toFloat)
}

print("PCARD Accuracy: " + avgAcc(test.map(_.label).collect(), pcard)._1)
```

Test Accuracy: 0.8020

Maven

Add to pom.xml:

```
<dependencies>
  <!-- list of dependencies -->
  <dependency>
    <groupId>djgg</groupId>
    <artifactId>PCARD</artifactId>
    <version>1.3</version>
  </dependency>
</dependencies>
```

Outline

- Introduction
- Decision Tree
- Random Forest
- PCARD
- **Metrics**

Metrics

Binary classification

Metric	Definition
Precision (Positive Predictive Value)	$PPV = \frac{TP}{TP+FP}$
Recall (True Positive Rate)	$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$
F-measure	$F(\beta) = (1+\beta^2) \cdot \left(\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR}\right)$
Receiver Operating Characteristic (ROC)	$FPR(T) = \int_T^\infty P_0(T)\,dT$ $TPR(T) = \int_T^\infty P_1(T)\,dT$
Area Under ROC Curve	$AUROC = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$
Area Under Precision-Recall Curve	$AUPRC = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{P}\right)$

Metrics

Multiclass classification

Metric	Definition
Confusion Matrix	$C_{ij} = \sum_{k=0}^{N-1} \hat{\delta}(\mathbf{y}_k - \ell_i) \cdot \hat{\delta}(\hat{\mathbf{y}}_k - \ell_j)$ $\begin{pmatrix} \sum_{k=0}^{N-1} \hat{\delta}(\mathbf{y}_k - \ell_1) \cdot \hat{\delta}(\hat{\mathbf{y}}_k - \ell_1) & \ldots & \sum_{k=0}^{N-1} \hat{\delta}(\mathbf{y}_k - \ell_1) \cdot \hat{\delta}(\hat{\mathbf{y}}_k - \ell_N) \\ \vdots & \ddots & \vdots \\ \sum_{k=0}^{N-1} \hat{\delta}(\mathbf{y}_k - \ell_N) \cdot \hat{\delta}(\hat{\mathbf{y}}_k - \ell_1) & \ldots & \sum_{k=0}^{N-1} \hat{\delta}(\mathbf{y}_k - \ell_N) \cdot \hat{\delta}(\hat{\mathbf{y}}_k - \ell_N) \end{pmatrix}$
Accuracy	$ACC = \frac{TP}{TP+FP} = \frac{1}{N} \sum_{i=0}^{N-1} \hat{\delta}\left(\hat{\mathbf{y}}_i - \mathbf{y}_i\right)$
Precision by label	$PPV(\ell) = \frac{TP}{TP+FP} = \frac{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell) \cdot \hat{\delta}(\mathbf{y}_i - \ell)}{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell)}$
Recall by label	$TPR(\ell) = \frac{TP}{P} = \frac{\sum_{i=0}^{N-1} \hat{\delta}(\hat{\mathbf{y}}_i - \ell) \cdot \hat{\delta}(\mathbf{y}_i - \ell)}{\sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)}$
F-measure by label	$F(\beta, \ell) = \left(1 + \beta^2\right) \cdot \left(\frac{PPV(\ell) \cdot TPR(\ell)}{\beta^2 \cdot PPV(\ell) + TPR(\ell)}\right)$
Weighted precision	$PPV_w = \frac{1}{N} \sum_{\ell \in L} PPV(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$
Weighted recall	$TPR_w = \frac{1}{N} \sum_{\ell \in L} TPR(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$
Weighted F-measure	$F_w(\beta) = \frac{1}{N} \sum_{\ell \in L} F(\beta, \ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$

Metrics

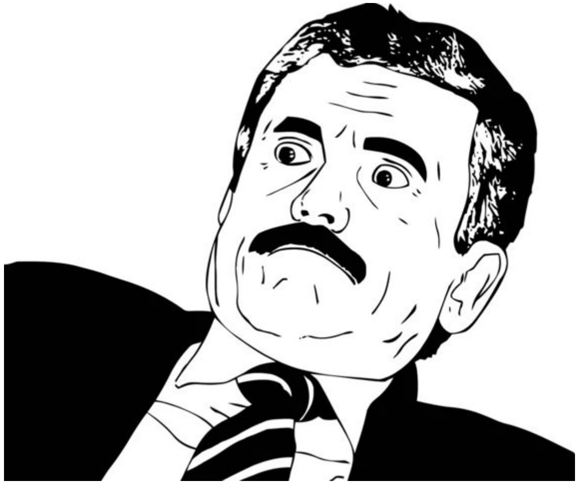
RDD[(Double, Double)] => (label, prediction)

test.map(_.label) => RDD[Double]

pcard => Array[Double] ¿? to RDD ¿?

Metrics

```
val labelAndPreds =
sc.parallelize(pcard).zipWithIndex.map{case
(v,k) => (k,v)}.join(test.zipWithIndex.map{case
(v,k) => (k,v.label)}).map(_._2)
```



Metrics

```
val labelAndPreds = sc.parallelize(pcard)
.zipWithIndex
.map{case (v,k) => (k,v)}
.join(test
.zipWithIndex
.map{case (v,k) => (k,v.label)})
.map(_._2)
```

Metrics

import org.apache.spark.mllib.evaluation.MulticlassMetrics

val metrics = new MulticlassMetrics(labelAndPreds)

val precision = metrics.precision

val cm = metrics.confusionMatrix

```
scala> val cm = metrics.confusionMatrix
cm: org.apache.spark.mllib.linalg.Matrix =
5612.0  1022.0
958.0   2408.0
```

UNIVERSIDAD DE GRANADA

Escuela de Posgrado
Universidad de Granada

APACHE Spark™

MLlib Ensembles

Big Data II

Diego García (djgarcia@decsai.ugr.es) 10/04/2018