INSTITUTO DE ASTROFÍSICA DE ANDALUCÍA
EXCELENCIA SEVERO OCHOA
CSIC
DaSCI
Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence

Big Data: Apache Spark & MLlib

Diego J. García Gil

djgarcia@decsai.ugr.es

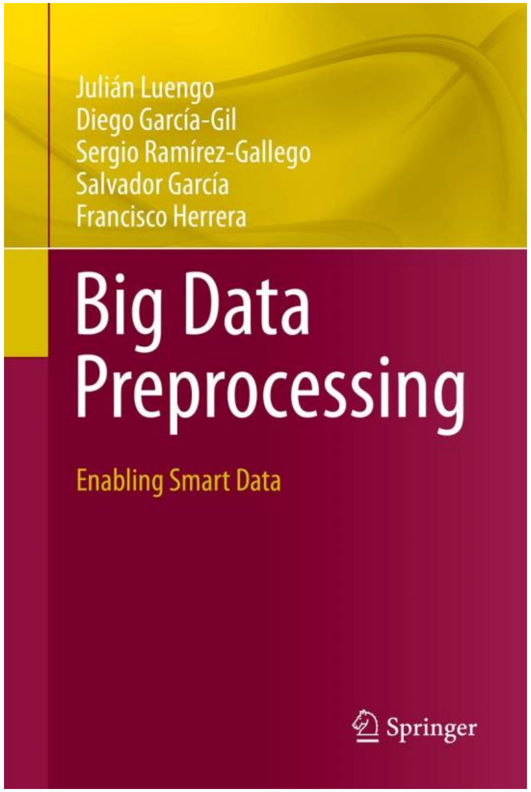
DECSAI

Outline

- Big Data
- Apache Spark
- Imperfect Data
- Data Reduction

Big Data Preprocessing: Enabling Smart Data

- Offers a comprehensible overview of Big Data Preprocessing
- Focuses on the most relevant proposed solutions
- Illustrates actual implementations of algorithms



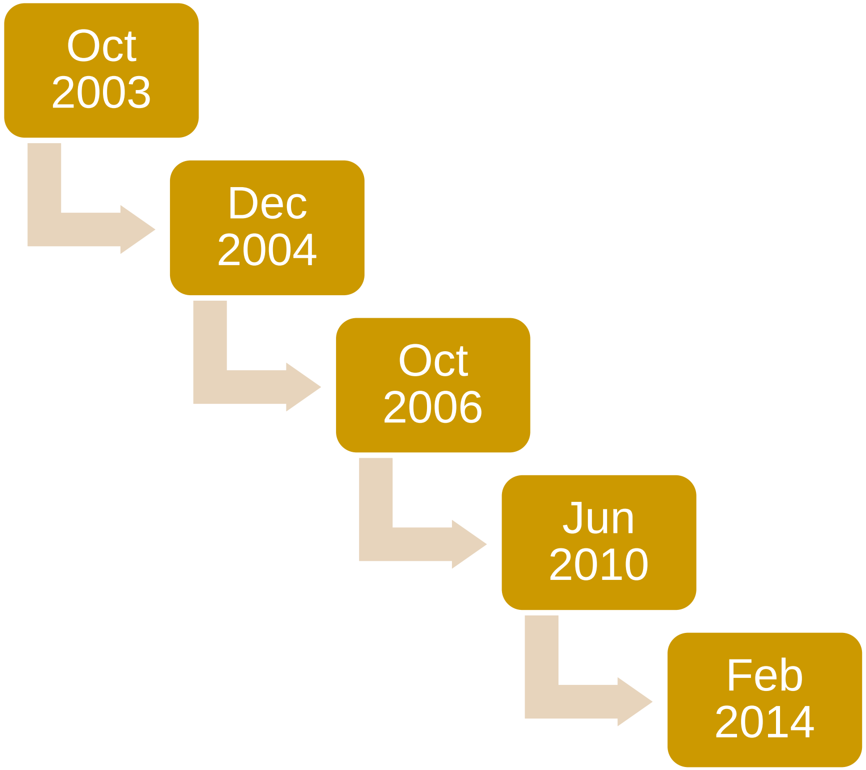
Julián Luengo
Diego García-Gil
Sergio Ramírez-Gallego
Salvador García
Francisco Herrera

Big Data Preprocessing

Enabling Smart Data

Springer

Big Data Frameworks Timeline



Oct 2003

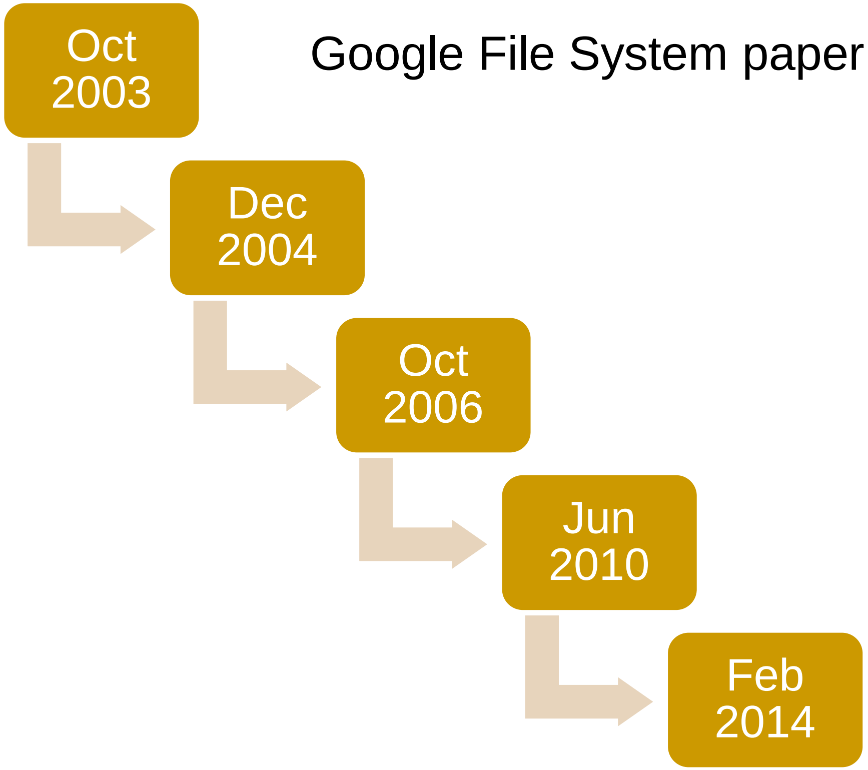
Dec 2004

Oct 2006

Jun 2010

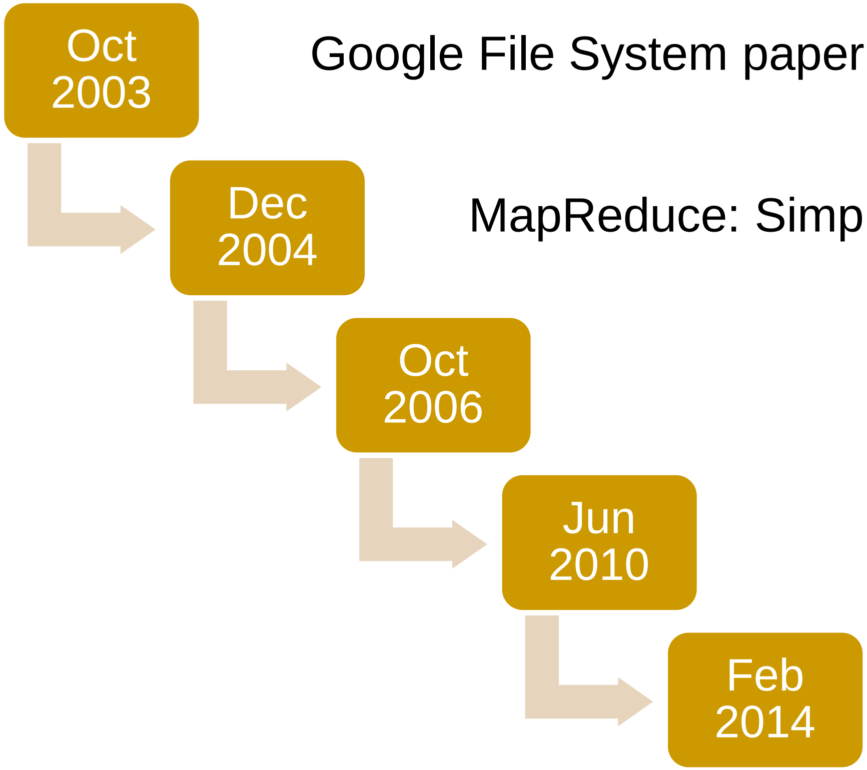
Feb 2014

Big Data Frameworks Timeline



Google File System paper released

Big Data Frameworks Timeline



Oct 2003 — Google File System paper released

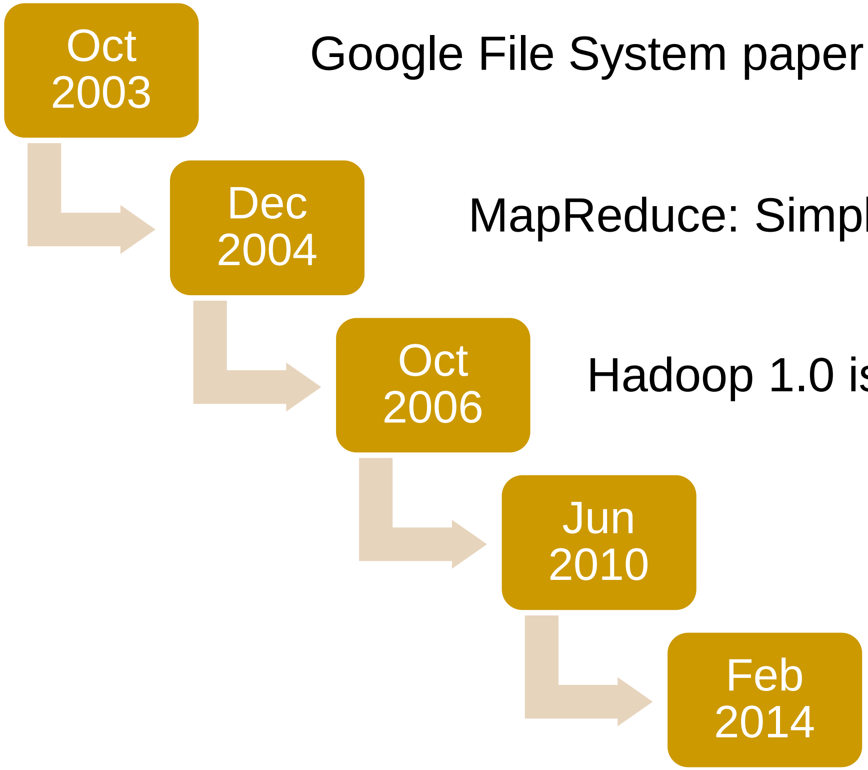
Dec 2004 — MapReduce: Simplified Data Processing on Large Clusters

Oct 2006

Jun 2010

Feb 2014

Big Data Frameworks Timeline



Oct 2003

Google File System paper released

Dec 2004

MapReduce: Simplified Data Processing on Large Clusters

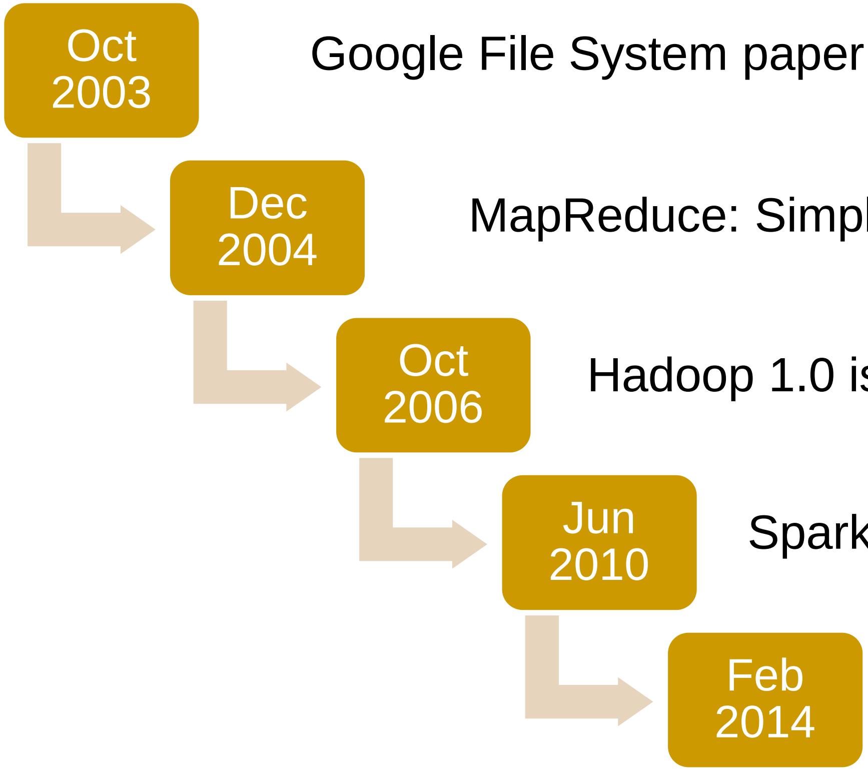
Oct 2006

Hadoop 1.0 is released by the ASF

Jun 2010

Feb 2014

Big Data Frameworks Timeline



Oct 2003 — Google File System paper released

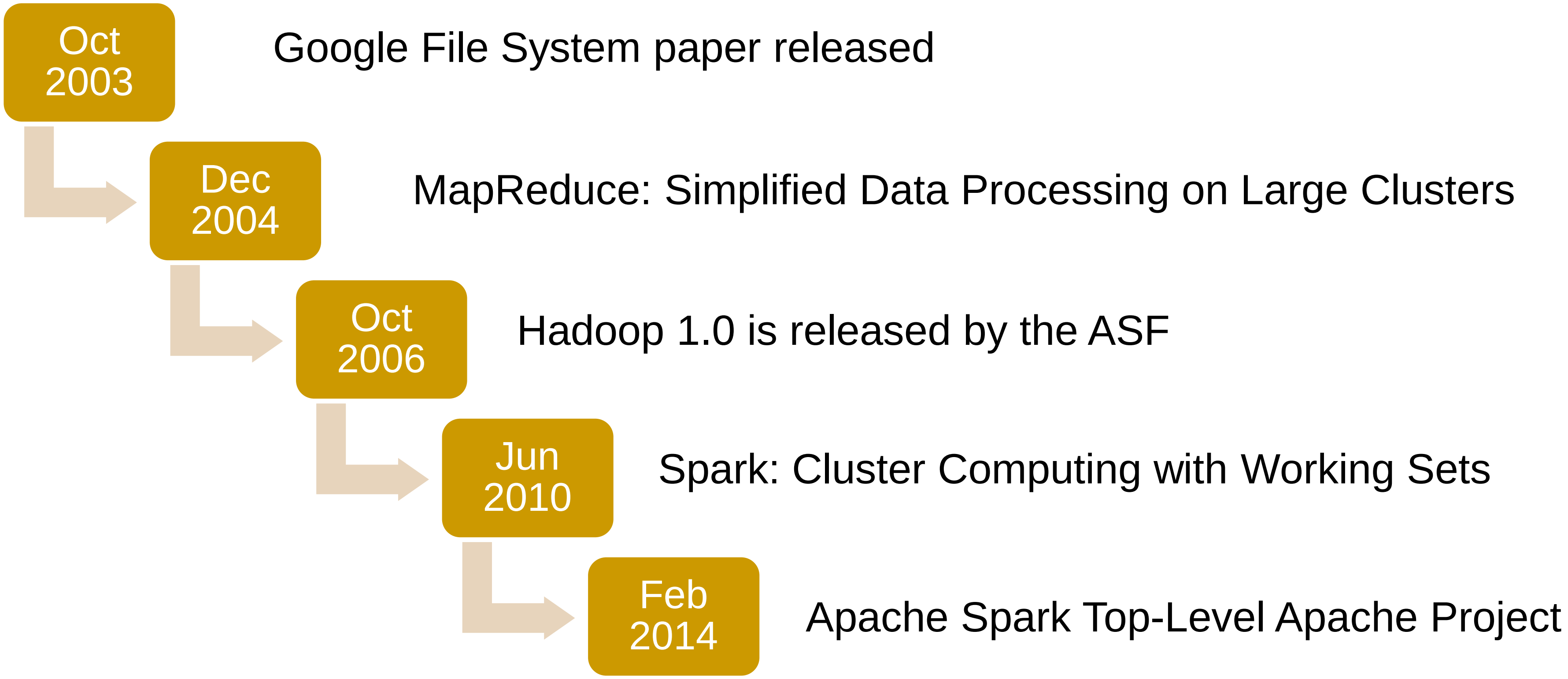
Dec 2004 — MapReduce: Simplified Data Processing on Large Clusters

Oct 2006 — Hadoop 1.0 is released by the ASF

Jun 2010 — Spark: Cluster Computing with Working Sets

Feb 2014

Big Data Frameworks Timeline



Oct 2003 — Google File System paper released

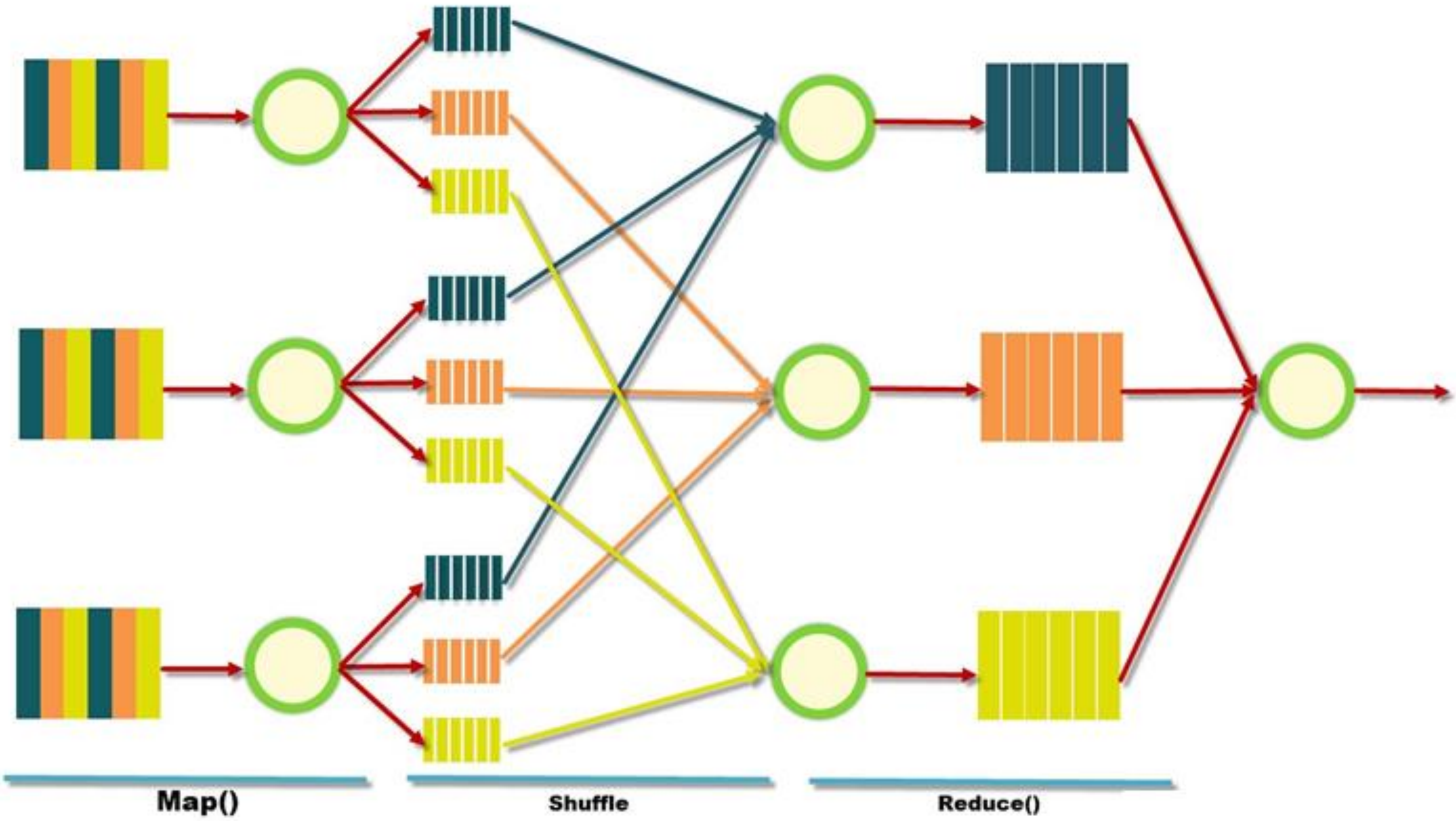
Dec 2004 — MapReduce: Simplified Data Processing on Large Clusters

Oct 2006 — Hadoop 1.0 is released by the ASF

Jun 2010 — Spark: Cluster Computing with Working Sets

Feb 2014 — Apache Spark Top-Level Apache Project

MapReduce



Apache Hadoop Weaknesses

- Use of HDD disc
- Java programming
- There is no interactive shell
- You can not iterate over the data
- However, it is widely used for its great advantages

https://spark.apache.org/

What is Spark?

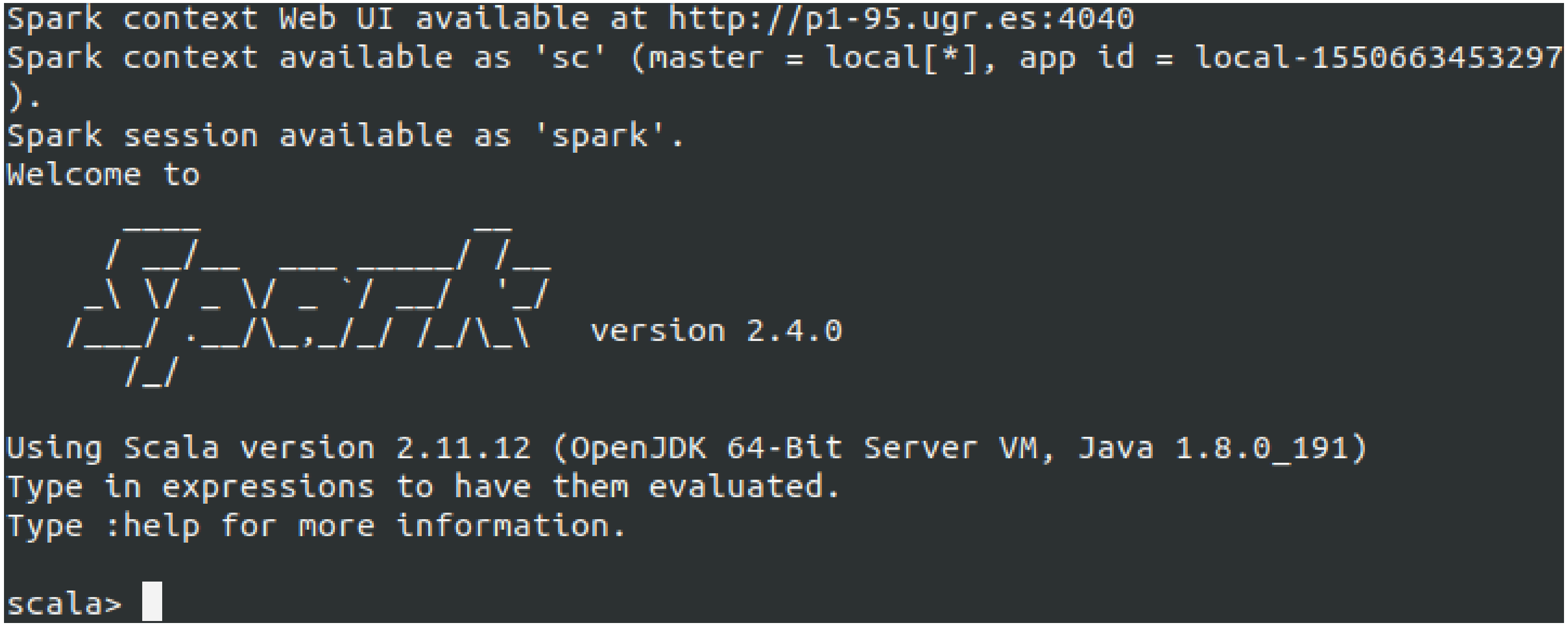
- **Fast** and **expressive** cluster computing system compatible with Apache Hadoop
  - Works with any Hadoop-supported storage system (HDFS, S3, …)
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python and R
  - …

And the most important feature…

And the most important feature…

Interactive Shell!!!

And the most important feature…



```
Spark context Web UI available at http://p1-95.ugr.es:4040
Spark context available as 'sc' (master = local[*], app id = local-1550663453297
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.0
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

(exit with :q)

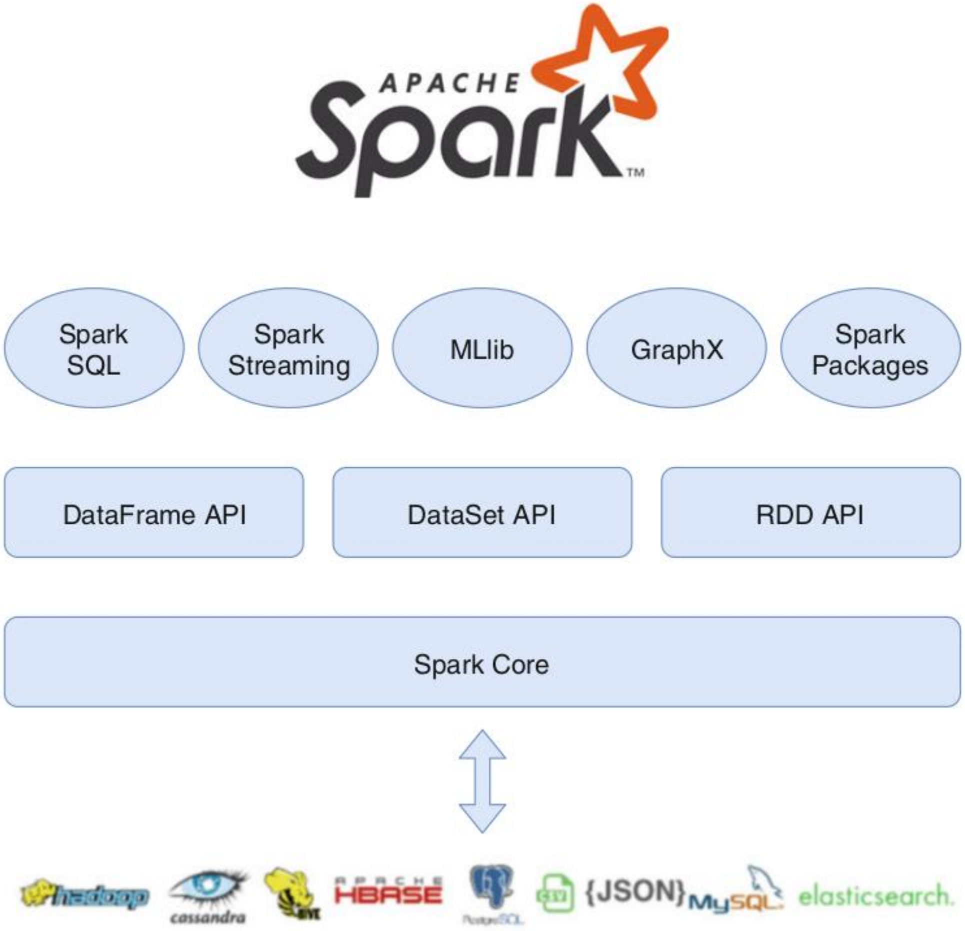
Interactive Shell

- The fastest way to learn Spark
- Powerful tool to analyze data interactively
- Runs as an application on an existing Spark Cluster
- Or can run locally

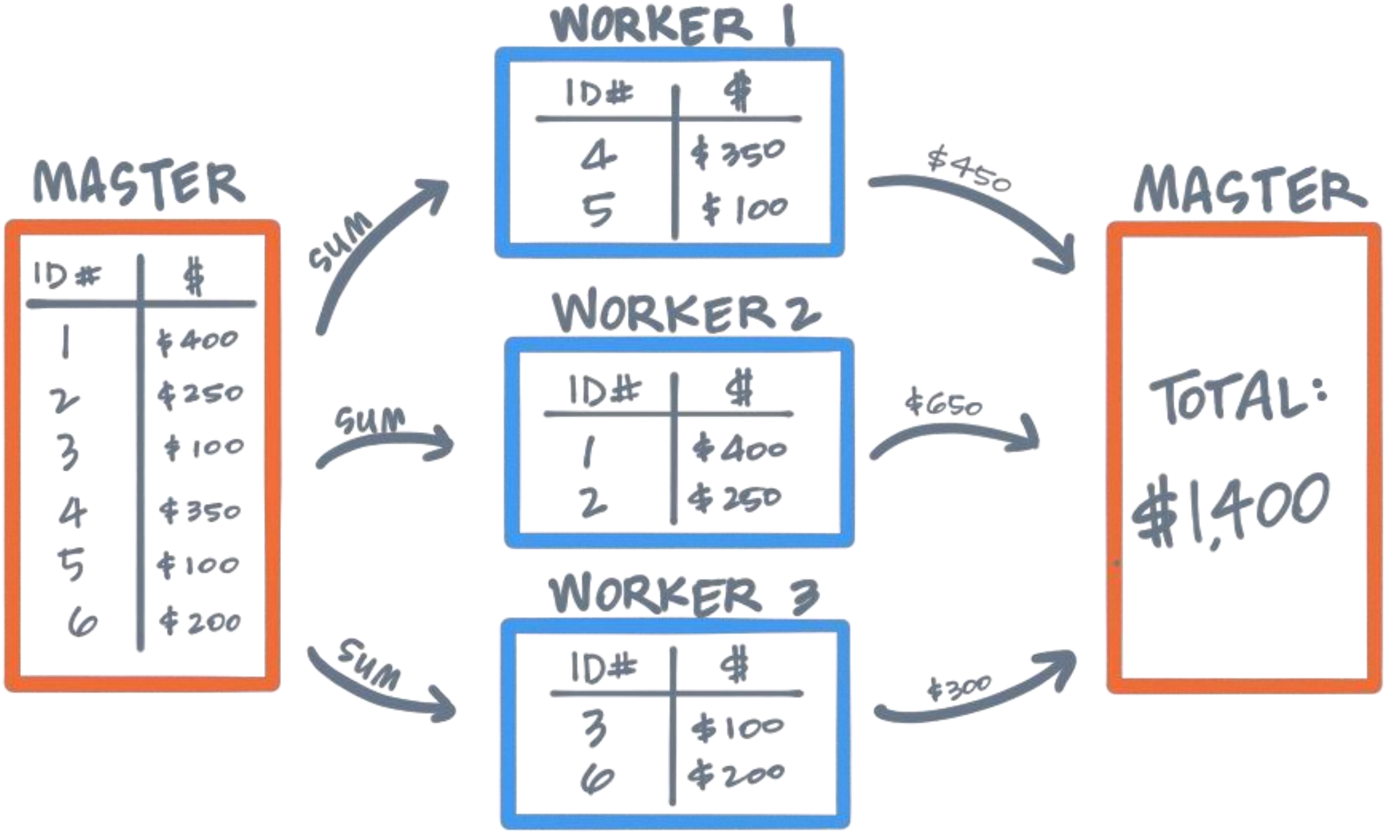
Spark’s Key Idea

- Work with distributed collections as you would with local ones
- Concept: Resilient Distributed Datasets (RDDs)
  - Immutable collection of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM)

Spark Framework



RDDs



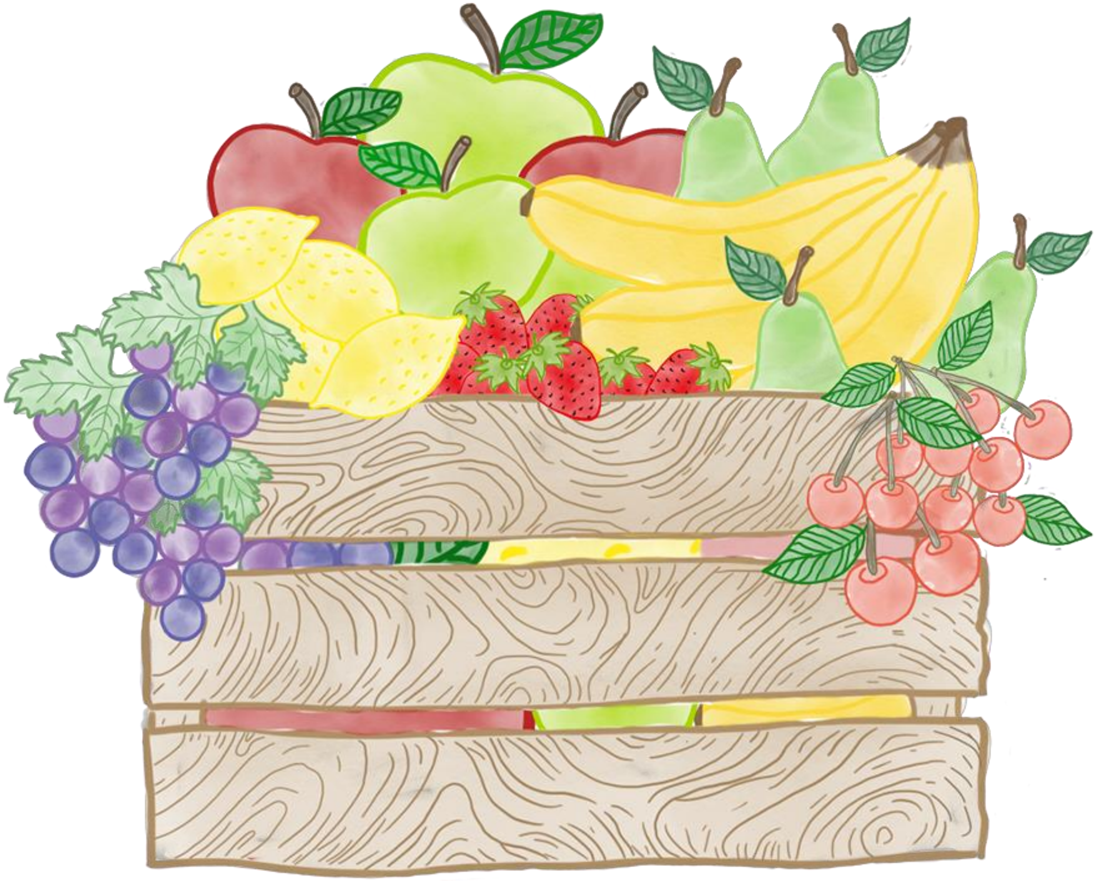
ID #	$
1	$400
2	$250
3	$100
4	$350
5	$100
6	$200

ID#	$
4	$350
5	$100

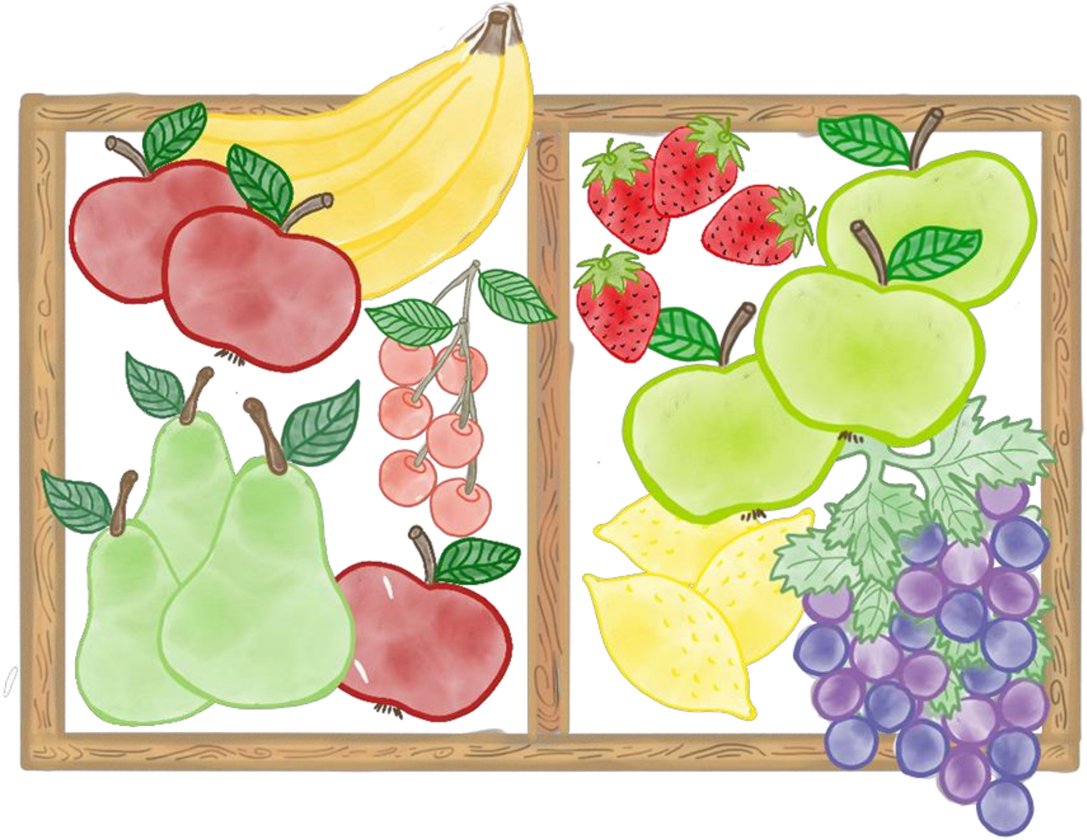
ID#	$
1	$400
2	$250

ID#	$
3	$100
6	$200

Partitions



Partitions x 2



Operations

- Transformations
  - Lazy operations to build RDDs from other RDDs
  - Not evaluated when defined
  - No loops!
  - map, mapPartition, filter, repartition, sample, union, …

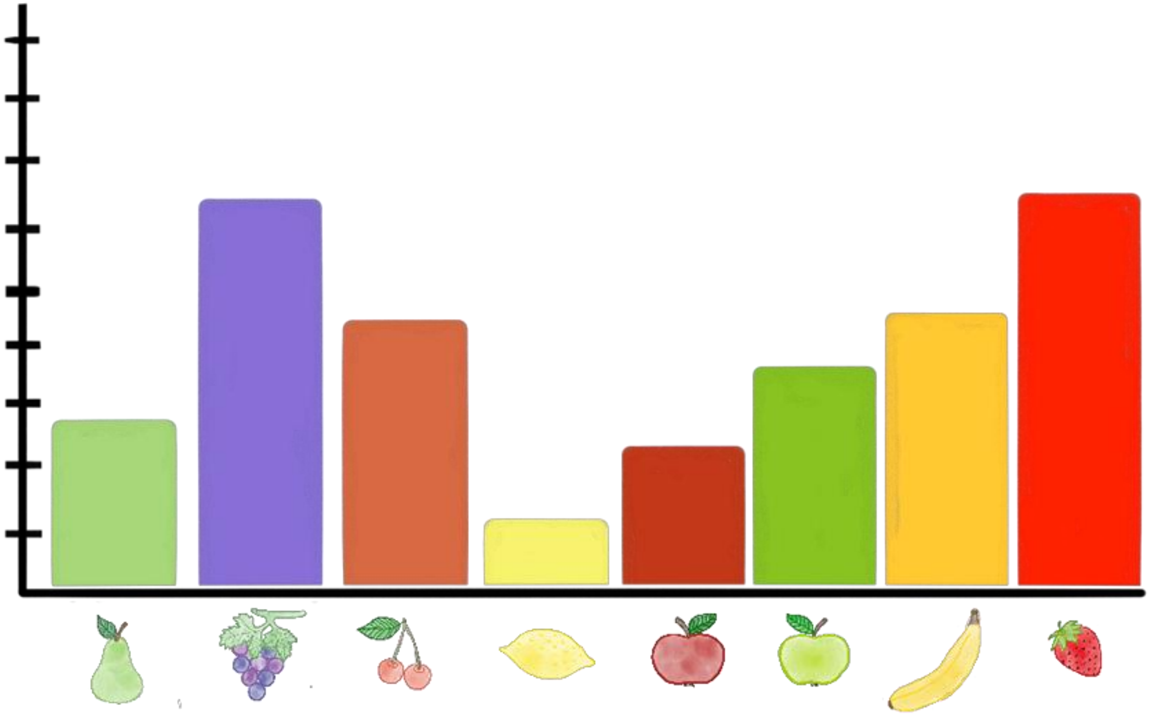
http://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

Operations

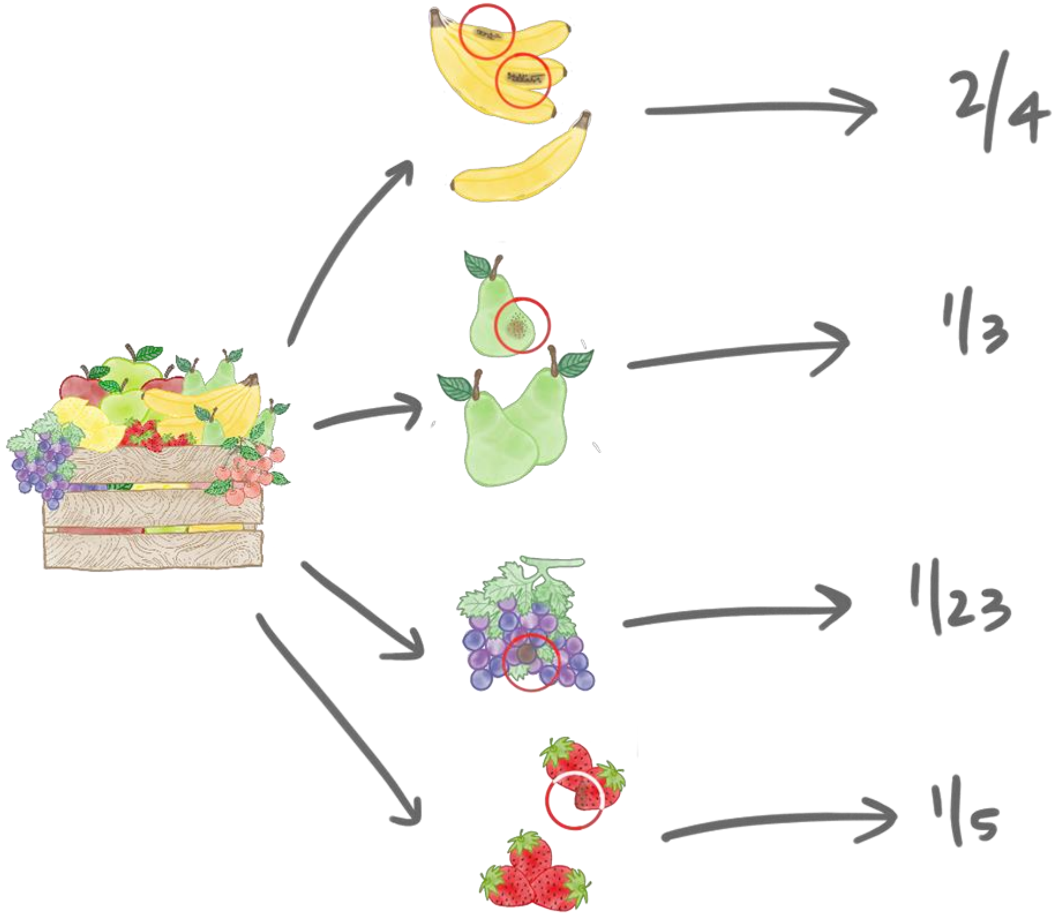
- Actions
  - Return a result or write it to storage
  - Triggers all previous transformations
  - count, first, take, collect, saveAsTextFile, reduce…

http://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

Map

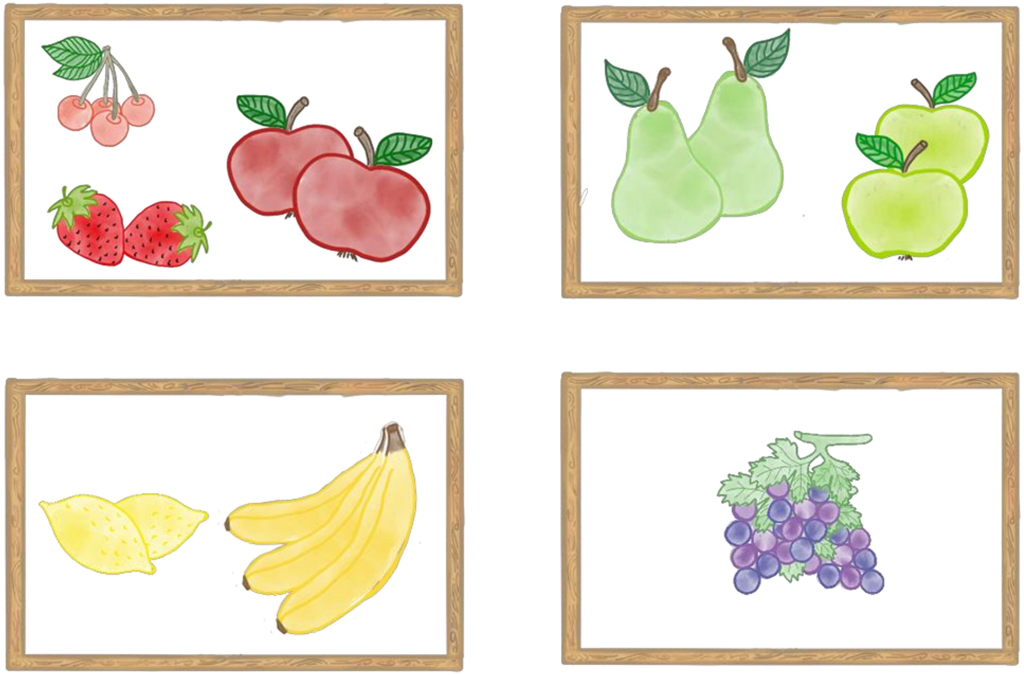


Map

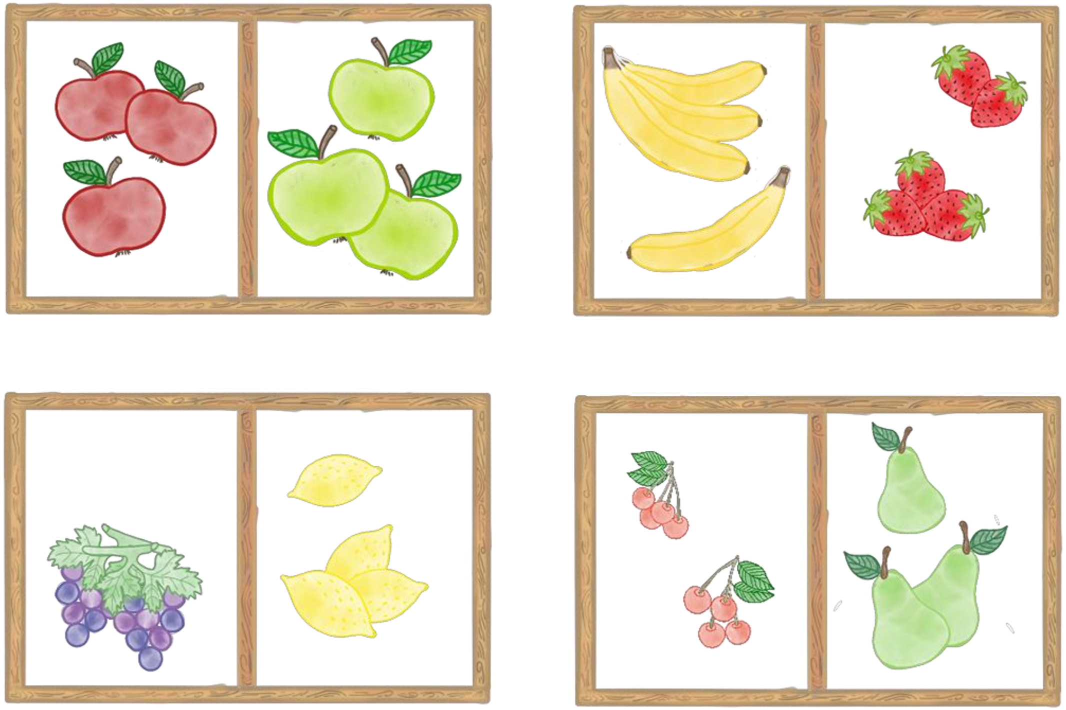


Map Partitions

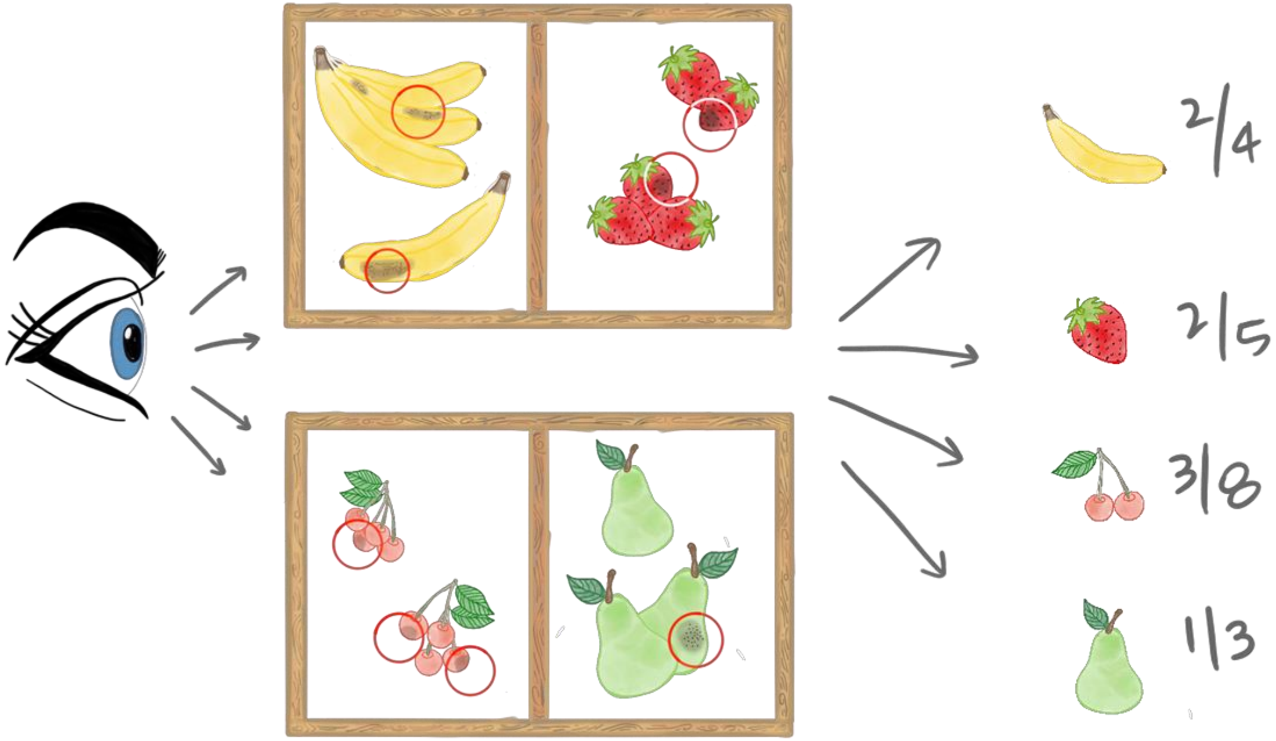
Group By color



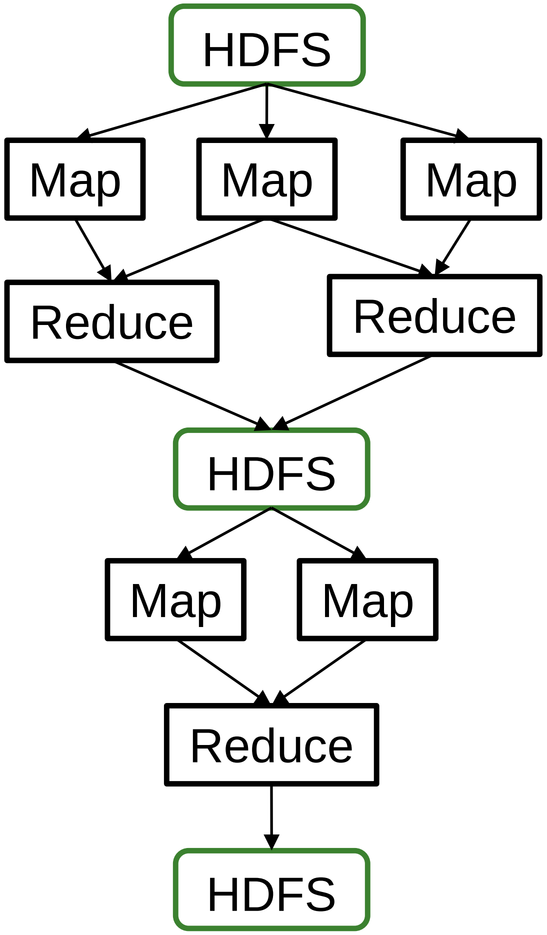
Group By type

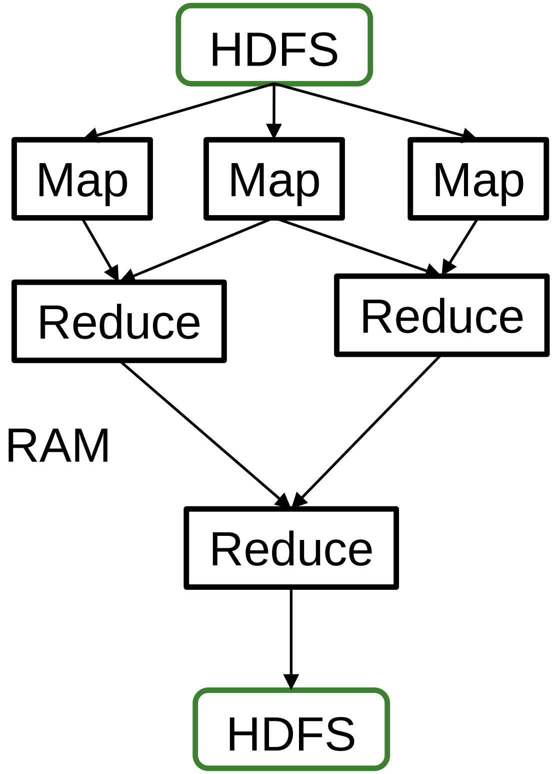


Map Partitions

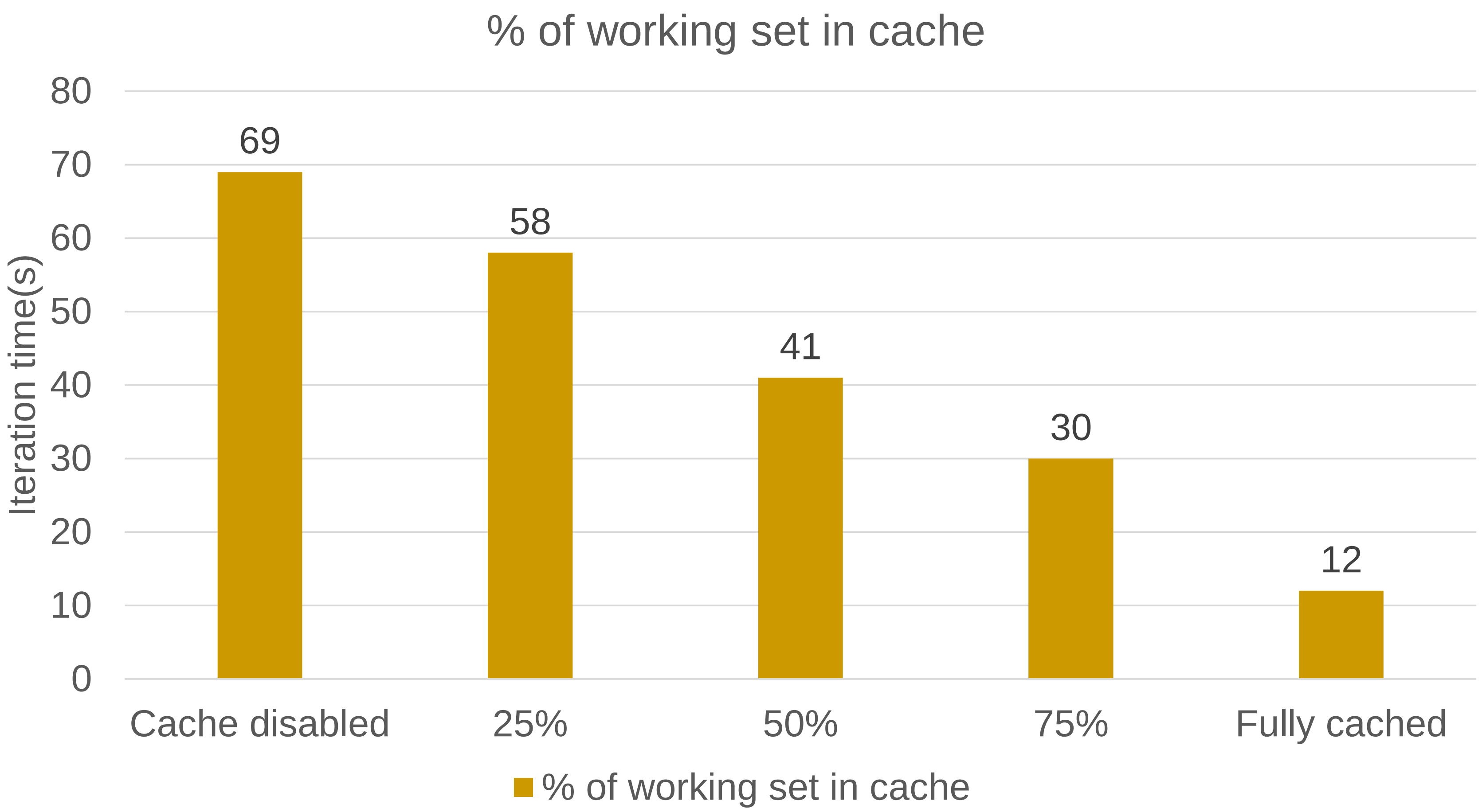


Hadoop vs Spark



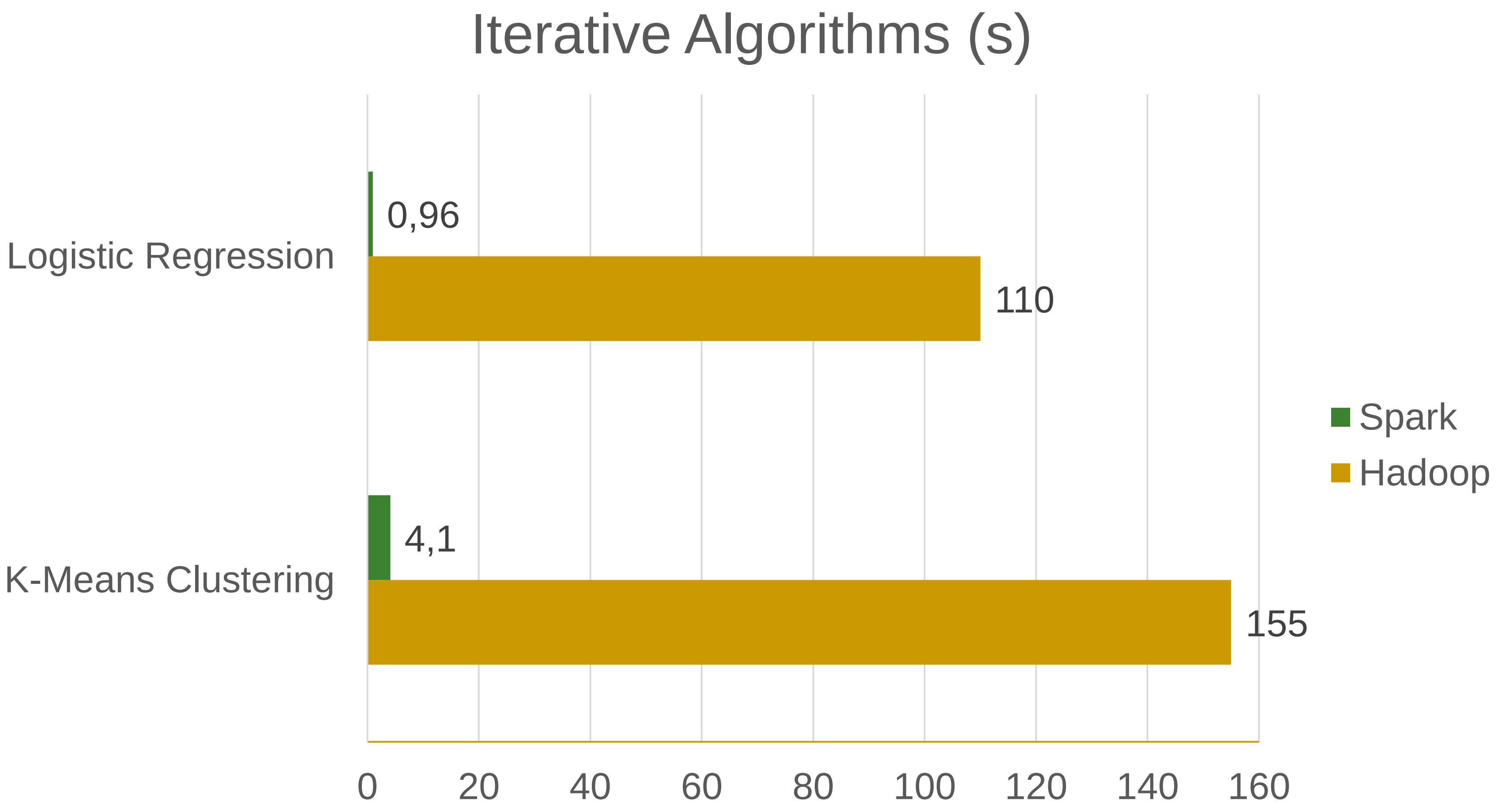


Efficiency



% of working set in cache
Iteration time(s)
80
70
60
50
40
30
20
10
0
69
58
41
30
12
Cache disabled
25%
50%
75%
Fully cached
■ % of working set in cache

Efficiency


Iterative Algorithms (s)
Logistic Regression
0,96
110
K-Means Clustering
4,1
155
0
20
40
60
80
100
120
140
160
Spark
Hadoop

Large-Scale Sorting

	Hadoop MR Record
Data Size	102.5 TB
Elapsed Time	72 mins
# Nodes	2100
# Cores	50400 physical
Cluster disk throughput	3150 GB/s (est.)
Sort Benchmark Daytona Rules	Yes
Network	dedicated data center, 10Gbps
Sort rate	**1.42 TB/min**
Sort rate/node	**0.67 GB/min**

Large-Scale Sorting

	Hadoop MR Record	**Spark Record**
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Sort Benchmark Daytona Rules	Yes	Yes
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Large-Scale Sorting

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Scala

Introduction to Scala

- **Sca**lable **La**nguage
- High-level language for the JVM
  - Object oriented + functional programming
- Statically typed
  - Type inference saves us from having to write explicit types most of the time
- Interoperates with Java
  - Can use any Java class
  - Can be called from Java code

Open spark-shell

- VM password: somachine20

- Scripts
  - https://github.com/spsrc/somachine2020/tree/master/tutorials/tutorial_02_BD_spark

- Terminal
  - /opt/spark/bin/spark-shell

Quick Tour of Scala

Variable declaration

```
var x: Int = 7
var x = 7
val y = "hi"
```

Java equivalent

```
int x = 7;

final String y = "hi"
```

Functions

```
def square(x: Int): Int = x*x
def square(x: Int): Int={
   x*x  //last line returned
}
def announce(text:String)={
   println(text)
}
```

Java equivalent

```
int square(int x){
   return x*x;
}

void announce(String text){
  System.out.println(text);
}
```

Quick Tour of Scala

Processing collections with functional programming

```
val list = List(1, 2, 3)
list.foreach(x => println(x))    //prints 1, 2, 3
list.foreach(println)            //same

list.map(x => x + 2)      //returns a new List(3,4,5)
list.map(_ + 2)           //same

list.filter(x => x % 2 == 1) //return a new List (1,3)
list.filter(_ % 2 == 1)      //same

list.reduce((x,y) => x + y)  // => 6
list.reduce(_ + _)           // same
```

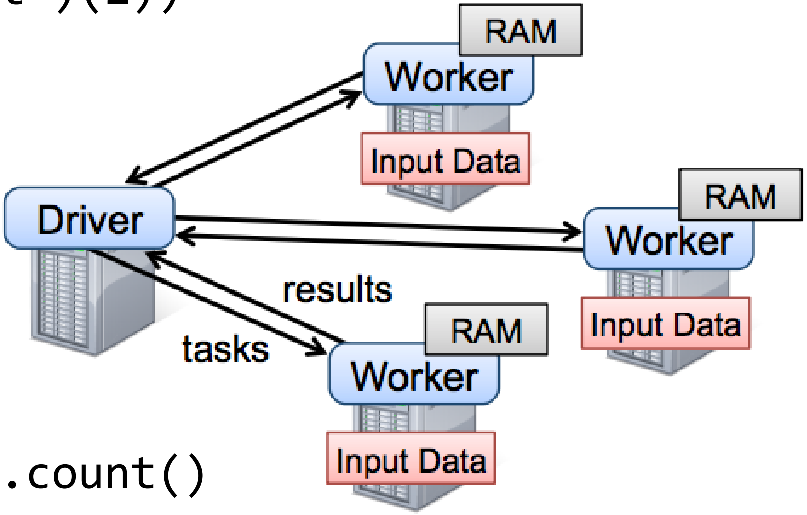
How do we communicate with Spark?

- SparkContext
  - Main entry point to Spark functionality
  - Created for you in Spark shells as variable sc
  - In standalone programs, you’d make your own

Example: Log Mining

```
ERROR    12:46:00         mysql: Unknown database ‘bd_IAA'
INFO     12:46:01         php: Aborting task
```

```
val lines = sc.textFile("hdfs://... ")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()
```

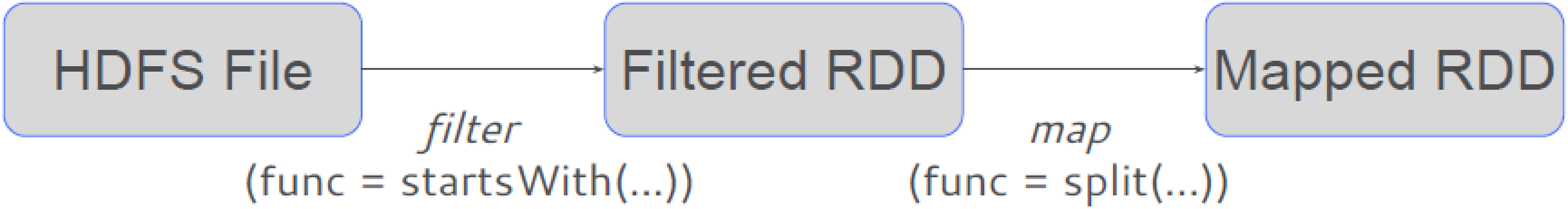


```
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

Fault Recovery

- RDDs track *lineage* information that can be used to efficiently recompute lost data

```
val msgs = lines.filter(_.startsWith("ERROR"))
                .map(_.split('\t')(2))
```



Creating RDDs

```
// Turn a local collection into an RDD
sc.parallelize(Array(1, 2, 3))

// Load text file from local FS, HDFS, or S3
sc.textFile("file:///home/user/file.txt")
sc.textFile("file:///home/user/*.txt")
sc.textFile("hdfs://namenode:8020/path/file")
```

Basic Transformations

```
val nums = sc.parallelize(Array(1, 2, 3))

// Pass each element through a function
val squares = nums.map(x => x*x) // => {1, 4, 9}

// Keep elements passing a predicate
val even = squares.filter(x => x % 2 == 0) // => {4}

// Map each element to zero or more others
nums.flatMap(x => Range(0, x)) // => {0, 0, 1, 0, 1, 2}
```

Basic Actions

```
val nums = sc.parallelize(Array(1, 2, 3))

// Retrieve RDD contents as a local collection
nums.collect() // => [1, 2, 3]

// Return first K elements
nums.take(2) // => [1, 2]

// Count number of elements
nums.count() // => 3

// Merge elements
nums.reduce(_+_) // => 6

// Write elements to a text file
nums.saveAsTextFile("file:///home/spark/file.txt")
```

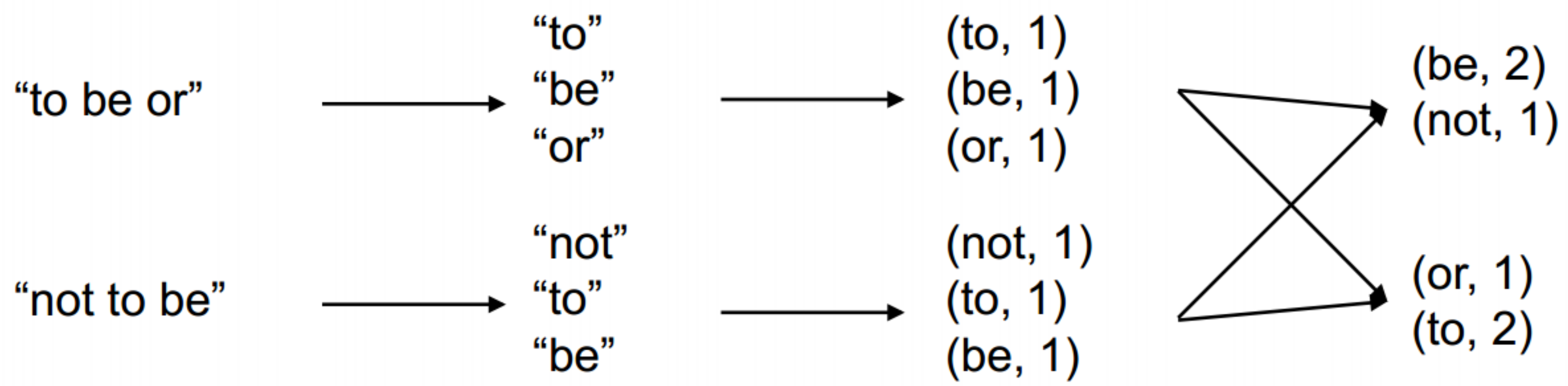
Key-Value Pairs

- Spark's "distributed reduce" transformations operate on RDDs of *key-value pairs*

```
val pair = ("a", "b")
pair._1 // => a
pair._2 // => b

val pets = sc.parallelize(Array(("cat", 1), ("dog", 1), ("cat", 2)))
pets.reduceByKey(_+_) // => {(cat, 3), (dog, 1)}
pets.groupByKey() // => {(cat, [1, 2]), (dog, [1])}
pets.sortByKey() // => {(cat,1), (cat, 2), (dog, 1)}
```

Example: Word Count

```
val textFile =
sc.textFile("file:///home/administrador/datasets/hamlet.txt")

val counts = textFile.flatMap(line => line.split(" "))
               .map(word => (word, 1))
               .reduceByKey(_ + _)
               .sortBy(_._2, ascending = false)
```



Multiple Datasets

```
val visits = sc.parallelize(Array(("index.html", "1.2.3.4"),
("about.html", "3.4.5.6"), ("index.html", "1.3.3.1")))

val pageNames = sc.parallelize(Array(("index.html", "Home"),
("about.html", "About")))

visits.join(pageNames)
// (“index.html”, (“1.2.3.4”, “Home”))
// (“index.html”, (“1.3.3.1”, “Home”))
// (“about.html”, (“3.4.5.6”, “About”))

visits.cogroup(pageNames)
// (“index.html”, (Seq(“1.2.3.4”, “1.3.3.1”), Seq(“Home”)))
// (“about.html”, (Seq(“3.4.5.6”), Seq(“About”)))
```

Level of Parallelism

- All the pair operations take an optional second parameter for number of tasks

```
pets.reduceByKey(_+_, 5)
pets.groupByKey(5)
visits.join(pageViews, 5)
```

- Or you can specify the number of partitions

```
sc.textFile(path, 5)
pets.repartition(5)
pets.coalesce(5)
```

Shared Variables

- **Broadcast variables** allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks
- **Accumulators** are variables that are only “added” to through an associative and commutative operation and can therefore be efficiently supported in parallel

Local Variables

- External variables you use in a closure will automatically be shipped to the cluster:

```
val query = raw_input("Enter a query:")
pages.filter(x => x.startswith(query)).count()
```

- Some caveats:
  - Each task gets a new copy (updates aren't sent back)
  - Variable must be Serializable (Java/Scala) or Pickle-able (Python)
  - Don't use fields of an outer object (ships all of it!)

MLlib

MLlib: Machine Learning on Spark

- MLlib is a scalable machine learning library
  - Easy to deploy
  - Take advantage of Hadoop environment
  - Contains may algorithms and utilities

https://spark.apache.org/docs/latest/mllib-guide.html

Algorithms and Utilities

- Classification
  - Naïve Bayes, Decision Trees, Random Forests, …
- Regression
  - Linear regression, Decision Trees, Random Forests,…
- Clustering
  - K-means, LDA, …
- Statistics
  - Summary Statistics, correlations, random data generation, …

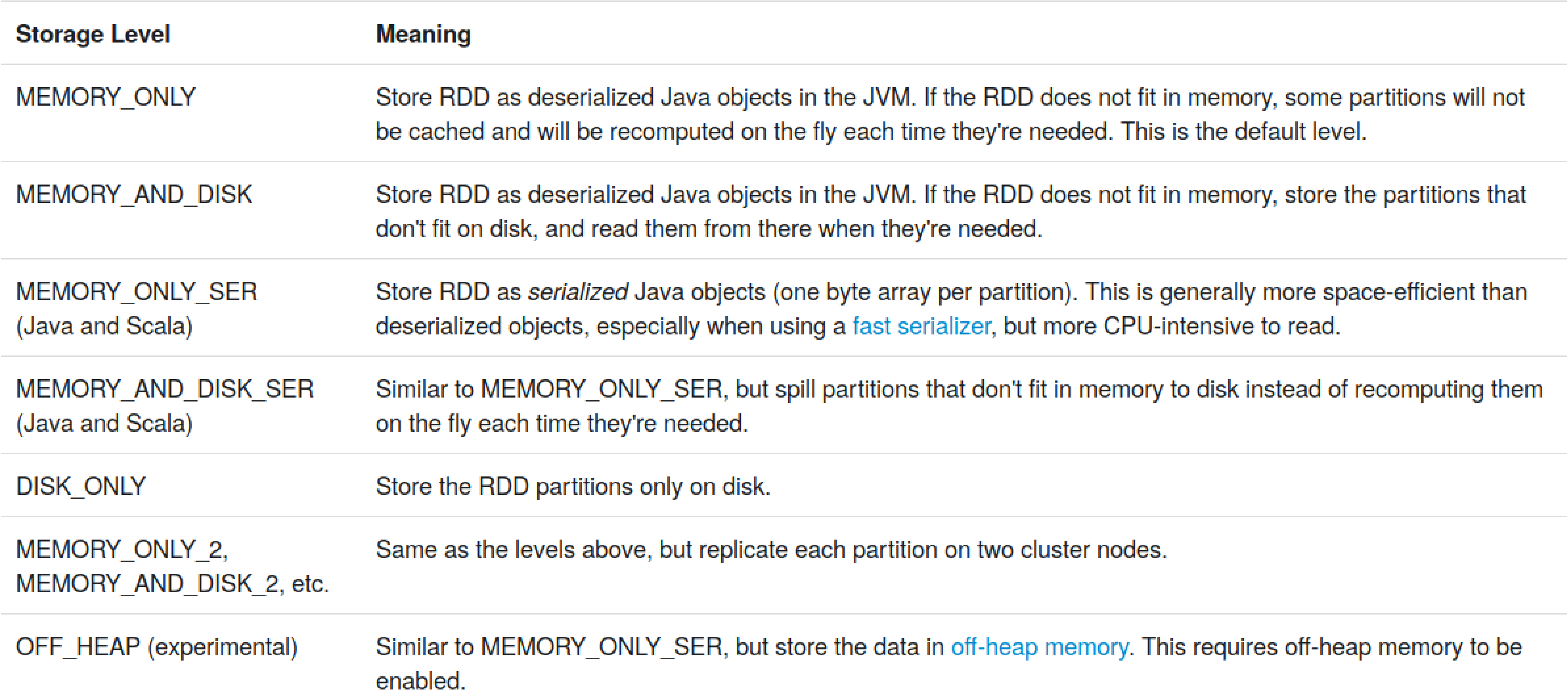
MLlib Data Types

- Local vector
  - 0-based indices and double-typed values, stored on a single machine
- LabeledPoint
  - Main MLlib data type
  - Local vector with a label (label, features)
- Distributed Matrix
  - RowMatrix: backed by an RDD of local rows
  - IndexedRowMatrix
  - CoordinateMatrix: formed by (i: Long, j: Long, value: Double)
  - Block Matrix

Cache

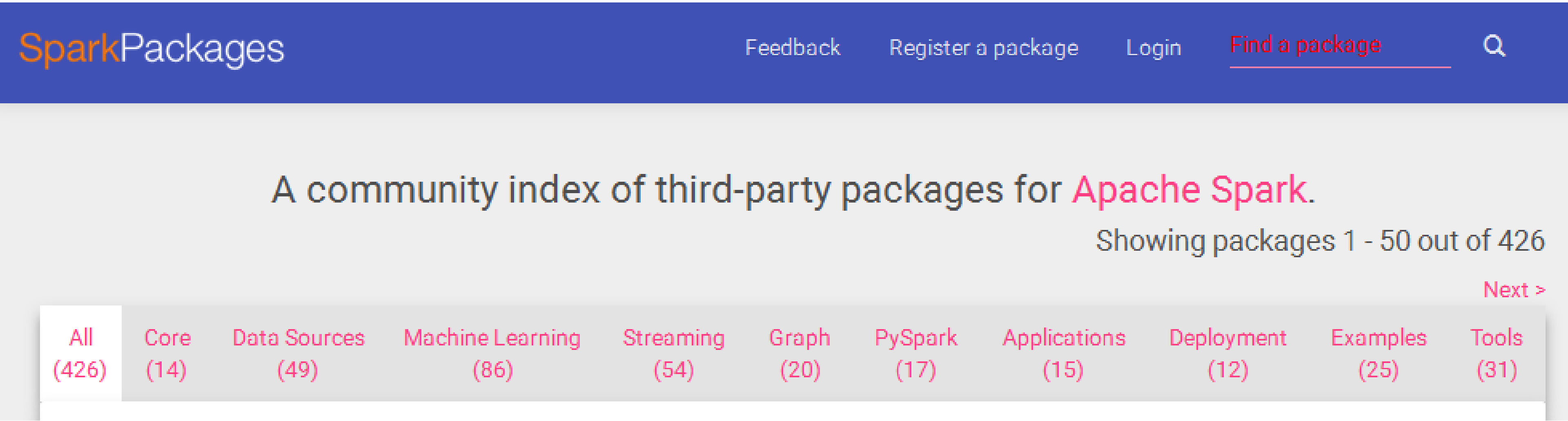
- RDD.cache() or RDD.persist(level)
- RDDs will be kept in node's memory **when an action is performed**
- Spark automatically persists some intermediate data in certain operations

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.

Spark Packages



A community index of third-party packages for Apache Spark.

Showing packages 1 - 50 out of 426

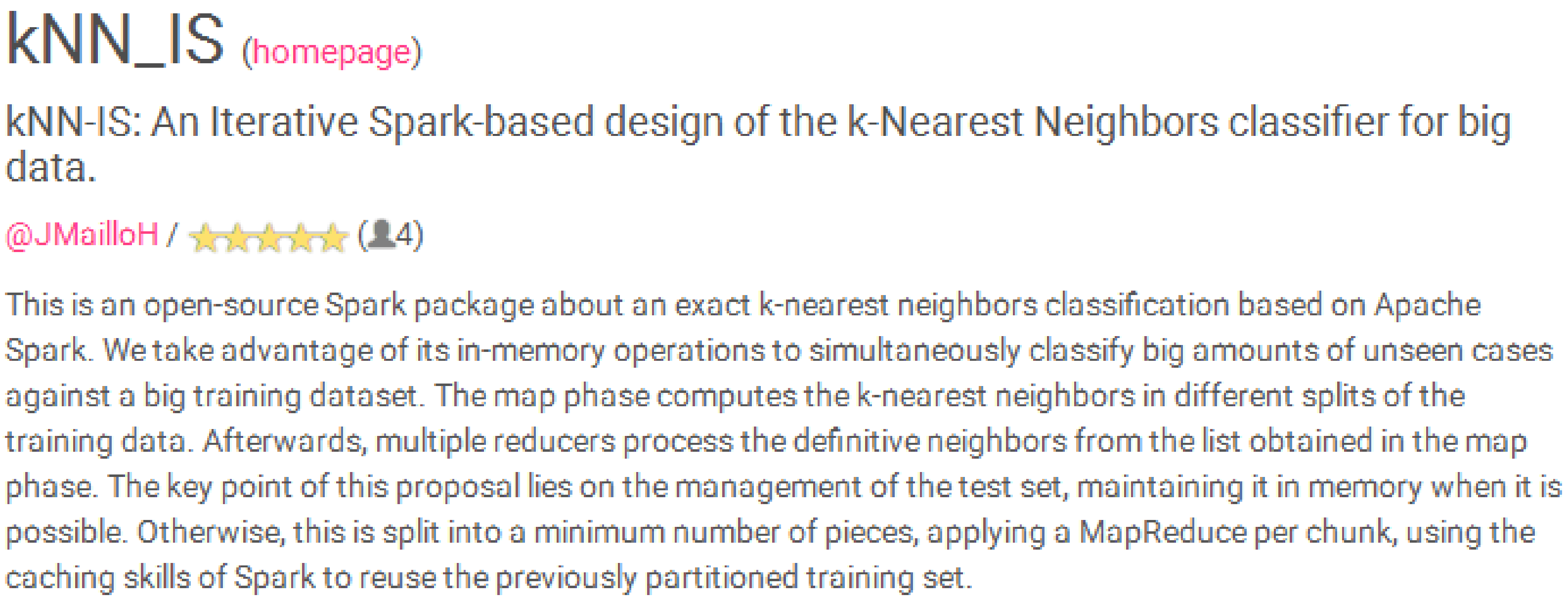
Large number of “utils” implementations

Machine Learning algorithms

Open source

https://spark-packages.org/

kNN_IS



kNN_IS (homepage)

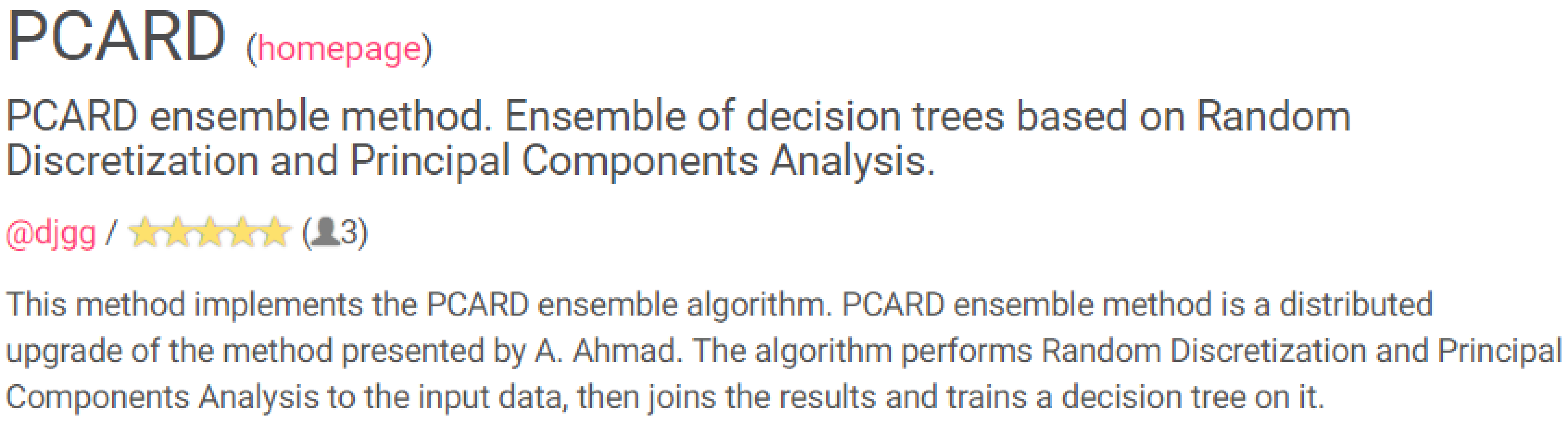
kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data.

@JMailloH / ★★★★★ (👤4)

This is an open-source Spark package about an exact k-nearest neighbors classification based on Apache Spark. We take advantage of its in-memory operations to simultaneously classify big amounts of unseen cases against a big training dataset. The map phase computes the k-nearest neighbors in different splits of the training data. Afterwards, multiple reducers process the definitive neighbors from the list obtained in the map phase. The key point of this proposal lies on the management of the test set, maintaining it in memory when it is possible. Otherwise, this is split into a minimum number of pieces, applying a MapReduce per chunk, using the caching skills of Spark to reuse the previously partitioned training set.

```
/opt/spark/bin/spark-shell --packages JMailloH:kNN_IS:3.0
```

PCARD



PCARD (homepage)

PCARD ensemble method. Ensemble of decision trees based on Random Discretization and Principal Components Analysis.

@djgg / ★★★★★ (👤3)

This method implements the PCARD ensemble algorithm. PCARD ensemble method is a distributed upgrade of the method presented by A. Ahmad. The algorithm performs Random Discretization and Principal Components Analysis to the input data, then joins the results and trains a decision tree on it.

```
/opt/spark/bin/spark-shell --packages djgg:PCARD:1.3,
JMailloH:kNN_IS:3.0
```

Dataset

- SUSY
  - Subset of 10,000 instances in the VM (5M original)
  - Number of instances: 10,000 + 10,000 (18 features)

    train test

  - Classification problem
  - Real features

Machine Learning Repository. SUSY Data Set
https://archive.ics.uci.edu/ml/datasets/SUSY

SUSY dataset

- The first eight features are kinematic properties measured by the particle detectors at the Large Hadron Collider. The last ten are functions of the first eight features
- The task is to distinguish between a signal process which produces supersymmetric (SUSY) particles and a background process which does not

P. Baldi , P. Sadowski , D. Whiteson , Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (2014) 4308

Dataset

- Path
  - Local (Virtual Machine)
    - Training: /home/administrador/datasets/susy-10k-tra.data
    - Test: /home/administrador/datasets/susy-10k-tst.data
    - Header: /home/administrador/datasets/susy.header

Load data

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


//Load Train & Test

val pathTrain = "file:///home/administrador/datasets/susy-10k-tra.data"
val rawDataTrain = sc.textFile(pathTrain)

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

RDD[String] → RDD[LabeledPoint]

- Each line is a String:

  “0.643,1.361,0.682,0.484,-0.325,-0.852,1.145,1.0”

LabeledPoint(1.0, 0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145)

RDD[String] → RDD[LabeledPoint]

- Each line is a String:

  “0.643,1.361,0.682,0.484,-0.325,-0.852,1.145,1.0”

- Cut each line into pieces:

  “0.643”, “1.361”, “0.682”, “0.484”, “-0.325”, “-0.852”, “1.145”, “1.0”

RDD[String] → RDD[LabeledPoint]

- Each line is a String:

  “0.643,1.361,0.682,0.484,-0.325,-0.852,1.145,1.0”

- Cut each line into pieces:

  “0.643”, “1.361”, “0.682”, “0.484”, “-0.325”, “-0.852”, “1.145”, “1.0”

- String → Double:

  0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145, 1.0

RDD[String] → RDD[LabeledPoint]

- Each line is a String:

  “0.643,1.361,0.682,0.484,-0.325,-0.852,1.145,1.0”

- Cut each line into pieces:

  “0.643”, “1.361”, “0.682”, “0.484”, “-0.325”, “-0.852”, “1.145”, “1.0”

- String → Double:

  0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145, 1.0

- Take Label & Features:

  0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145, 1.0

RDD[String] → RDD[LabeledPoint]

- Each line is a String:

  “0.643,1.361,0.682,0.484,-0.325,-0.852,1.145,1.0”

- Cut each line into pieces:

  “0.643”, “1.361”, “0.682”, “0.484”, “-0.325”, “-0.852”, “1.145”, “1.0”

- String → Double:

  0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145, 1.0

- Take Label & Features:

  0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145, 1.0

- Create LabeledPoint:

  LabeledPoint(1.0, 0.643, 1.361, 0.682, 0.484, -0.325, -0.852, 1.145)

Train & 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)
}

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)
}
```

Check Train & Test

```
train.persist
train.count     //Triggers all the previous transformations + cache
// 10000
train.first

test.persist
test.count
// 10000
test.first

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

Check Train & Test

```
train.persist
train.count
train.first

test.persist
test.count
test.first

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

```
1,0 -> 4907, 0,0 -> 5093
```

Balanced!

Statistics

```
import scala.collection.mutable.ListBuffer
import org.apache.spark.mllib.stat.{MultivariateStatisticalSummary,
Statistics}

val summaryTrain: MultivariateStatisticalSummary =
Statistics.colStats(train.map(_.features))

var outputString = new ListBuffer[String]
outputString += "******TRAIN******\n\n"
outputString += "@Max (0) --> " + summaryTrain.max(0) + "\n"
outputString += "@Min (0) --> " + summaryTrain.min(0) + "\n"
outputString += "@Mean (0) --> " + summaryTrain.mean(0) + "\n"
outputString += "@Variance (0) --> " + summaryTrain.variance(0) + "\n"
outputString += "@NumNonZeros (0) --> " + summaryTrain.numNonzeros(0) +
"\n"
```

Correlation

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.stat.Statistics

// calculate the correlation matrix using Pearson's method. Use
"spearman" for Spearman's method
// If a method is not specified, Pearson's method will be used by
default.

val correlMatrix: Matrix = Statistics.corr(train.map(_.features),
"pearson")

println(correlMatrix.toString)
```

Benchmark: Decision Tree

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

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 Prediction

```
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")
```

Decision Tree Prediction

```
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

Benchmark: Random Forest

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

// 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 Prediction

```
// 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")
```

Random Forest Prediction

```
// 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

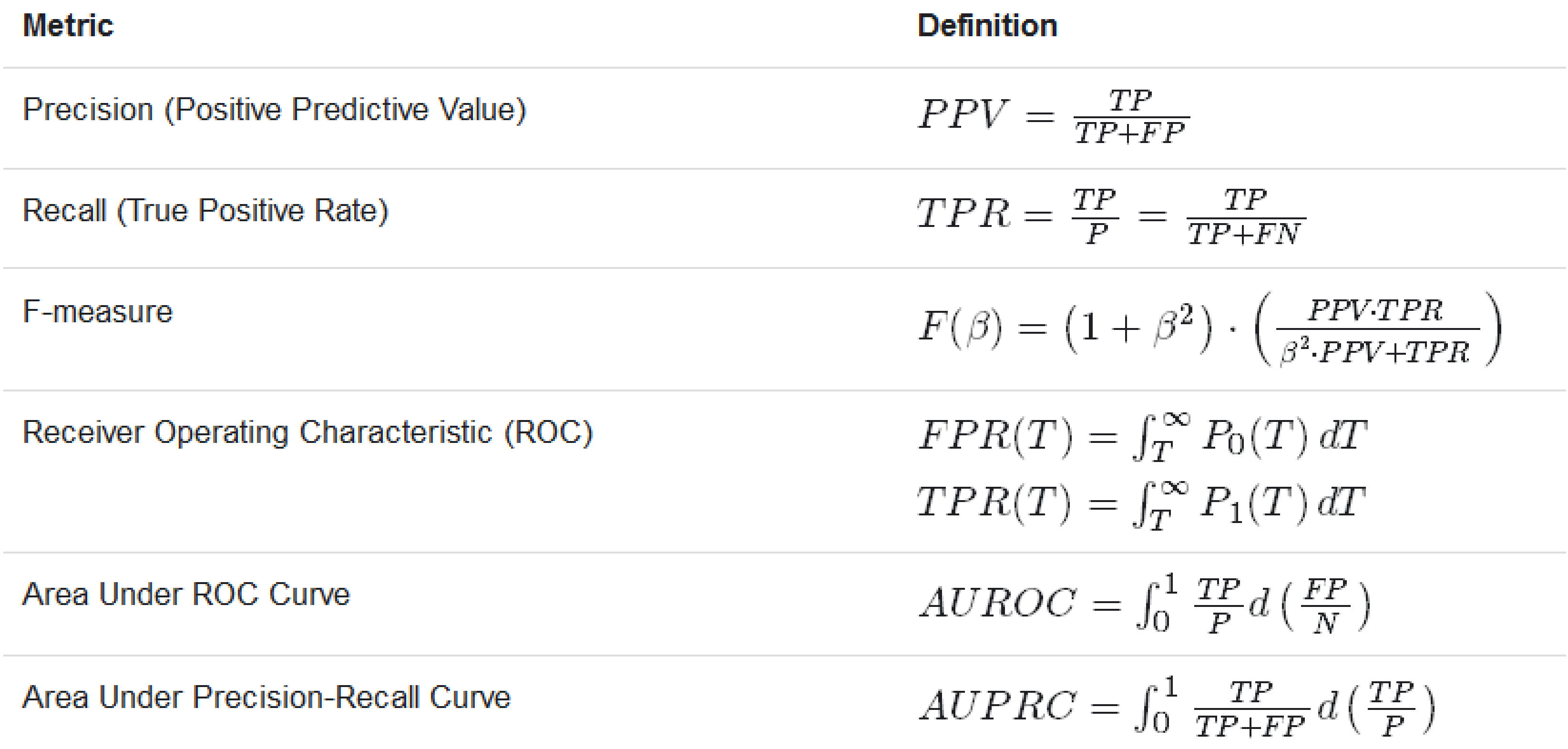
Benchmark: kNN

```
import org.apache.spark.mllib.classification.kNN_IS.kNN_IS

val k = 3

val numClass = train.map(_.label).distinct().collect().length
val numFeatures = train.first().features.size

val knn = kNN_IS.setup(train, test, k, 2, numClass, numFeatures,
train.getNumPartitions, 2, -1, 1)

val predictions = knn.predict(sc)
```

Metrics

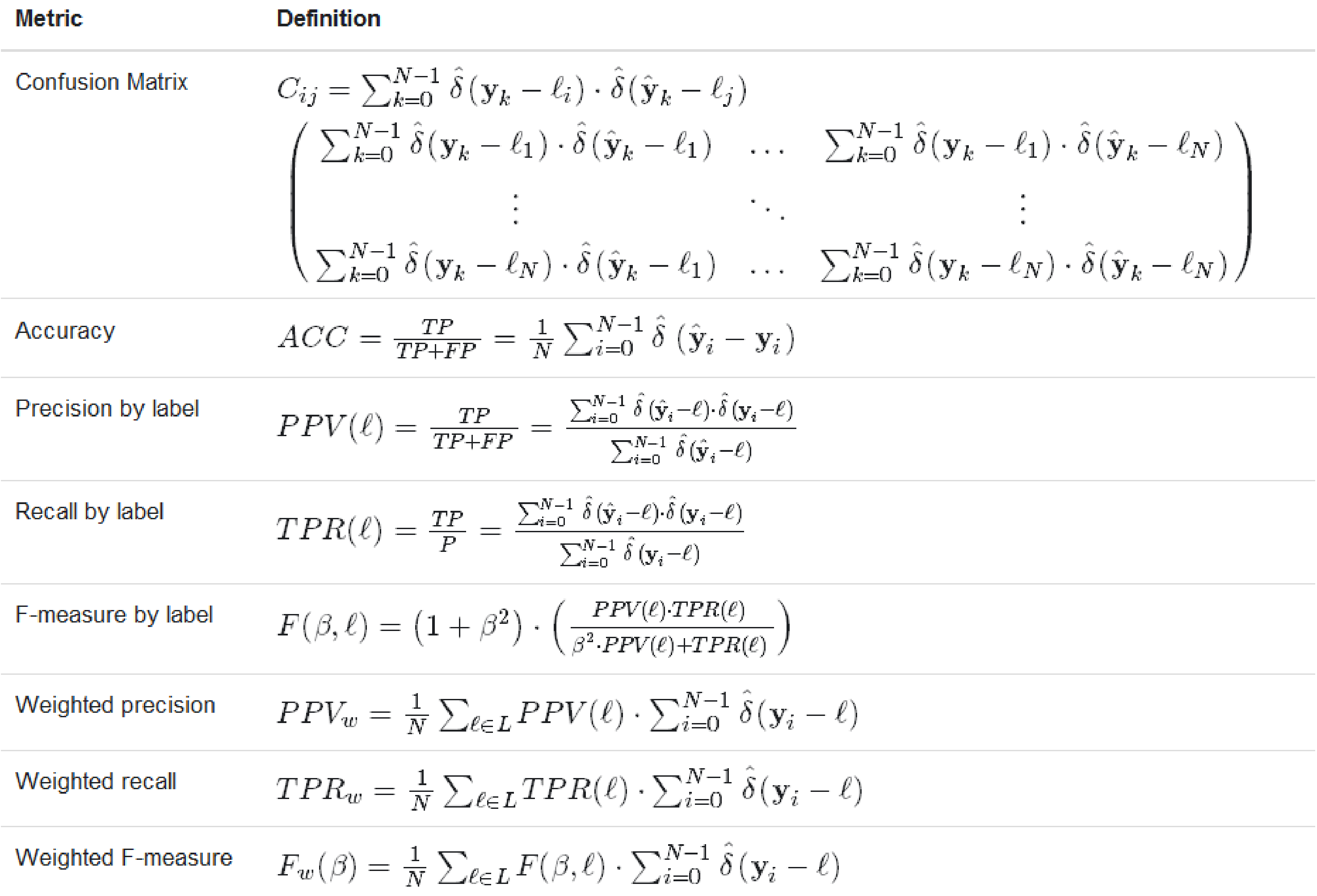
- predictions is an RDD[(Double, Double)]
- How can we easily calculate accuracy, confusion matrix…?
    - Spark’s metrics

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	$AUPRC = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{P}\right)$

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)

```
import org.apache.spark.mllib.evaluation._

val metrics = new MulticlassMetrics(predictions)
val precision = metrics.precision
val cm = metrics.confusionMatrix



val binaryMetrics = new BinaryClassificationMetrics(predictions)
val AUC = binaryMetrics.areaUnderROC
```

Metrics

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

```
import org.apache.spark.mllib.evaluation._

val metrics = new MulticlassMetrics(predictions)
val precision = metrics.precision           // 0.7411
val cm = metrics.confusionMatrix
// 5200       1434
// 1155       2211

val binaryMetrics = new BinaryClassificationMetrics(predictions)
val AUC = binaryMetrics.areaUnderROC        // 0.7204
```

Benchmark: PCARD

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

val cuts = 5 // bins for discretization
val trees = 10 // iterations

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

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

PCARD Prediction

```
val labels = test.map(_.label).collect()
var cont = 0

for (i <- labels.indices) {
  if (labels(i) == pcard(i)) {
    cont += 1
  }
}

val testAcc = cont / labels.length.toFloat

println(s"Test Accuracy = $testAcc")
```

PCARD Prediction

```
val labels = test.map(_.label).collect()
var cont = 0

for (i <- labels.indices) {
  if (labels(i) == pcard(i)) {
    cont += 1
  }
}

val testAcc = cont / labels.length.toFloat

println(s"Test Accuracy = $testAcc")
```

Test Accuracy: 0.8020

Imperfect Data

Data Preprocessing

- Imperfect Data
  - Transformations
  - Missing Values
  - Noise Filtering

Spark Shell

```
/opt/spark/bin/spark-shell --packages
djgarcia:NoiseFramework:1.2,
djgarcia:RandomNoise:1.0,
djgarcia:SmartFiltering:1.0,
JMailloH:Smart_Imputation:1.0,
JMailloH:kNN_IS:3.0
```

Load data

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


//Load Train & Test

val pathTrain = "file:///home/administrador/datasets/susy-10k-tra.data"
val rawDataTrain = sc.textFile(pathTrain)

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

Train & 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)
}

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)
}
```

Encapsulate Learning Algorithms

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

def trainDT(train: RDD[LabeledPoint], test: RDD[LabeledPoint], maxDepth: Int = 5): Double = {
    val numClasses = 2
    val categoricalFeaturesInfo = Map[Int, Int]()
    val impurity = "gini"
    val maxBins = 32

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

    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()
    testAcc
}
```

Encapsulate Learning Algorithms

```
import org.apache.spark.mllib.classification.kNN_IS.kNN_IS
import org.apache.spark.mllib.evaluation._
import org.apache.spark.rdd.RDD


def trainKNN(train: RDD[LabeledPoint], test: RDD[LabeledPoint], k: Int = 3): Double = {

    val numClass = train.map(_.label).distinct().collect().length
    val numFeatures = train.first().features.size

    val knn = kNN_IS.setup(train, test, k, 2, numClass, numFeatures, train.getNumPartitions, 2,
-1, 1)
    val predictions = knn.predict(sc)
    val metrics = new MulticlassMetrics(predictions)
    val precision = metrics.precision

    precision

}
```

Transformations

Normalization

- Scale to [0, 1]

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

How do we do it?

- We need maximum and minimum values for each feature
- Spark’s statistics
- Normalize train and then test?

Min & Max Values

```
import org.apache.spark.mllib.stat.{MultivariateStatisticalSummary,
Statistics}

val fullDataset = train.union(test)

val summary = Statistics.colStats(fullDataset.map(_.features))

summary.min

summary.max
```

Normalize Train & Test

```
val normalizedTrain = train.map{l =>
  val featuresArray = l.features.toArray.zipWithIndex.map{case (v,k) =>
    (v - summary.min(k)) / (summary.max(k) - summary.min(k))
  }
  LabeledPoint(l.label, Vectors.dense(featuresArray))
}

val normalizedTest = test.map{l =>
  val featuresArray = l.features.toArray.zipWithIndex.map{case (v,k) =>
    (v - summary.min(k)) / (summary.max(k) - summary.min(k))
  }
  LabeledPoint(l.label, Vectors.dense(featuresArray))
}
```

Check Train & Test

```
val summaryTrain = Statistics.colStats(normalizedTrain.map(_.features))
summaryTrain.min
summaryTrain.max

val summaryTest = Statistics.colStats(normalizedTest.map(_.features))
summaryTest.min
summaryTest.max

val summaryUnion =
Statistics.colStats(normalizedTrain.union(normalizedTest).map(_.features))
summaryUnion.min
summaryUnion.max
```

DT & kNN Results

```
trainDT(normalizedTrain, normalizedTest)
```

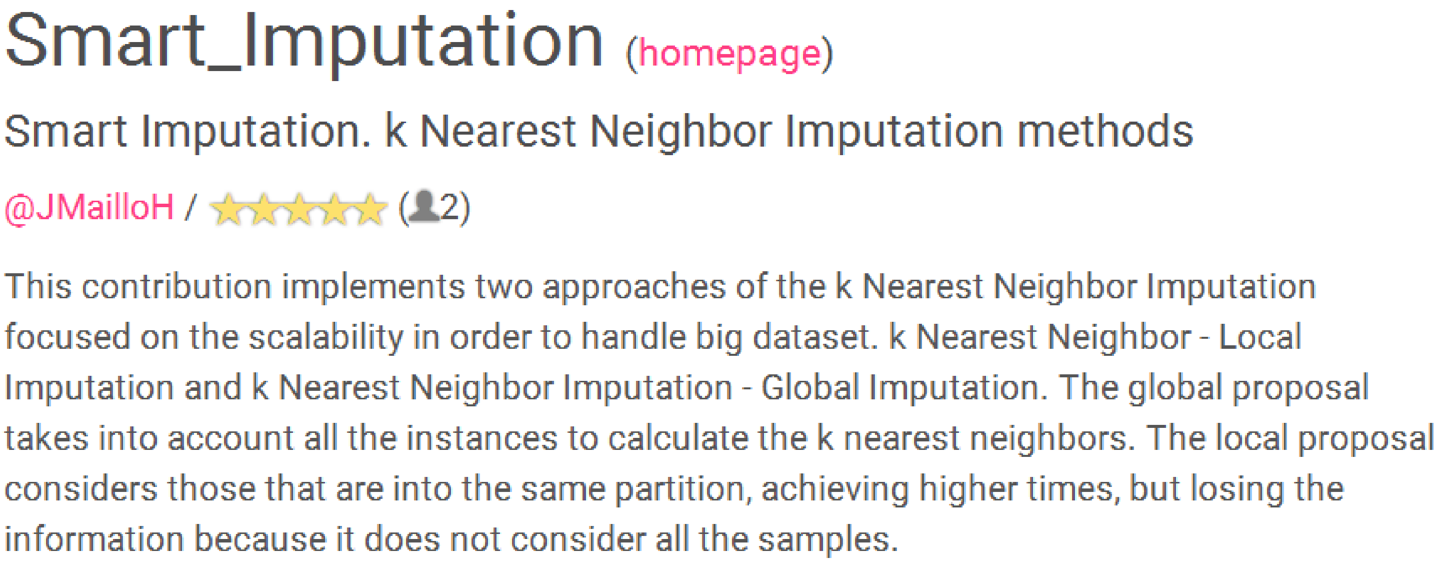
```
trainKNN(normalizedTrain, normalizedTest)
```

DT & kNN Results

```
trainDT(normalizedTrain, normalizedTest)

// 0.7876


trainKNN(normalizedTrain, normalizedTest)

// 0.7118
```

Missing Values

kNNI



Smart_Imputation (homepage)

Smart Imputation. k Nearest Neighbor Imputation methods

@JMailloH / ★★★★★ (👤2)

This contribution implements two approaches of the k Nearest Neighbor Imputation focused on the scalability in order to handle big dataset. k Nearest Neighbor - Local Imputation and k Nearest Neighbor Imputation - Global Imputation. The global proposal takes into account all the instances to calculate the k nearest neighbors. The local proposal considers those that are into the same partition, achieving higher times, but losing the information because it does not consider all the samples.

- Imputation using kNN

https://spark-packages.org/package/JMailloH/Smart_Imputation

Add Missing Values

```
val mv_pct = 30 // 30% of MVs

val tam = rawDataTrain.count.toInt // Number of instances

val num = math.round(tam * (mv_pct.toDouble / 100)) // Number of MVs

val range = util.Random.shuffle(0 to tam - 1) // Random number gen.

val indices = range.take(num.toInt) // Random instances

val broadcastInd = rawDataTrain.sparkContext.broadcast(indices)
```

Add MVs to Train Data

```
import scala.util.Random

val mvData = rawDataTrain.zipWithIndex.map {
  case (v, k) =>
    if (broadcastInd.value contains (k)) {
      val features = v.split(",").init
      val label = v.split(",").last
      val mv = features.indexOf(Random.shuffle(features.toList).head)
      features(mv) = "?"
      features.mkString(",").concat("," + label)
    } else {
      v
    }
}
mvData.persist

val mv_num = mvData.filter(_.contains("?")).count // 3000
```

Remove MVs

```
val train_without_mv = mvData.filter(!_.contains("?"))

val trainMV = train_without_mv.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)
}
```

Train DT & kNN

```
trainMV.persist
trainMV.count // 7000

trainDT(trainMV, test)





trainKNN(trainMV, test)
```

Train DT & kNN

```
trainMV.persist
trainMV.count // 7000

trainDT(trainMV, test)

// 0.7694


trainKNN(trainMV, test)

// 0.7367
```

Mean Imputation

```
val numFeatures = train.first().features.size
var means: Array[Double] = new Array(numFeatures)

for(x <- 0 to numFeatures-1){
  means(x) = mvData.map(_.split(",")(x)).filter(v =>
!v.contains("?")).map(_.toDouble).mean
}

val meanImputedData = mvData.map(_.split(",").zipWithIndex.map{case (v,k)=> if (v
== "?") means(k) else v.toDouble})

val mv_num = meanImputedData.filter(_.contains("?")).count // 0

val trainMean = meanImputedData.map{arrayDouble =>
    val featureVector = Vectors.dense(arrayDouble.init)
    val label = arrayDouble.last
    LabeledPoint(label, featureVector)
}
```

Train DT & kNN

```
trainMean.persist
trainMean.count // 10000
```

```
trainDT(trainMean, test)
```

```
trainKNN(trainMean, test)
```

Train DT & kNN

```
trainMean.persist
trainMean.count // 10000


trainDT(trainMean, test)

// 0.7857


trainKNN(trainMean, test)

// 0.7458
```

kNNI

```
import org.apache.spark.mllib.preprocessing.kNNI_IS.KNNI_IS

val k = 3
val pathHeader = "/home/USER/datasets/susy.header"

val knni = KNNI_IS.setup(mvData, k, 2, pathHeader, mvData.getNumPartitions,
"local")
val imputedData = knni.imputation(sc)

val mv_num_knni = imputedData.filter(_.contains("?")).count // 0

val trainKNNI = imputedData.map{array =>
  val arrayDouble = array.map(f => f.toDouble)
  val featureVector = Vectors.dense(arrayDouble.init)
  val label = arrayDouble.last
  LabeledPoint(label, featureVector)
}
```

Train DT & kNN

```
trainKNNI.persist
trainKNNI.count // 10000
```

```
trainDT(trainKNNI, test)
```

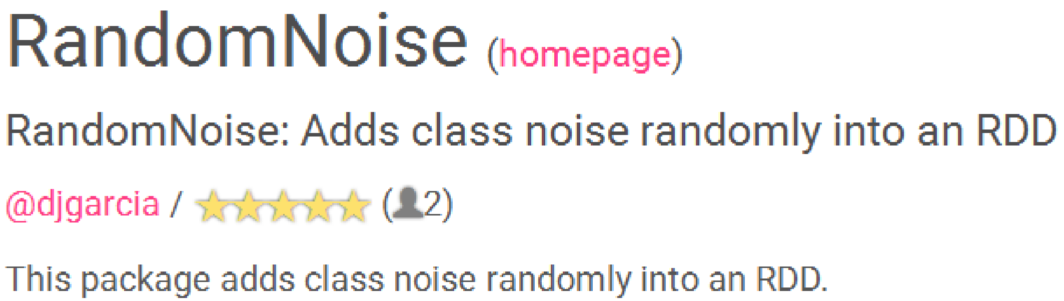
```
trainKNN(trainKNNI, test)
```

Train DT & kNN

```
trainKNNI.persist
trainKNNI.count // 10000


trainDT(trainKNNI, test)

// 0.7908


trainKNN(trainKNNI, test)

// 0.7448
```

Noise Filtering

Random Noise



RandomNoise (homepage)

RandomNoise: Adds class noise randomly into an RDD

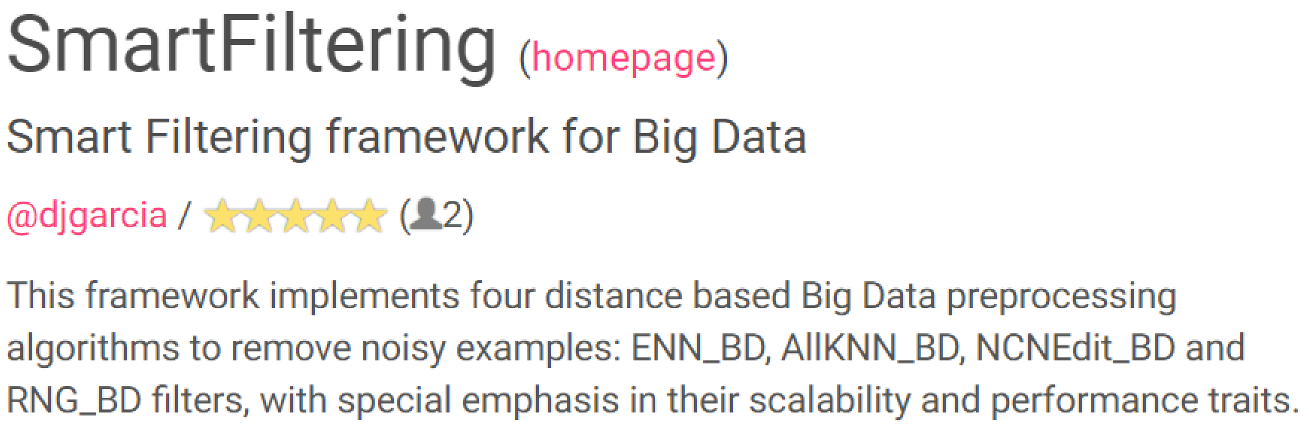
@djgarcia / ★★★★★ (👤2)

This package adds class noise randomly into an RDD.

https://spark-packages.org/package/djgarcia/RandomNoise

Noise Filtering with kNN

- ENN_BD, AllKNN_BD, NCNEdit_BD & RNG_BD



SmartFiltering (homepage)

Smart Filtering framework for Big Data

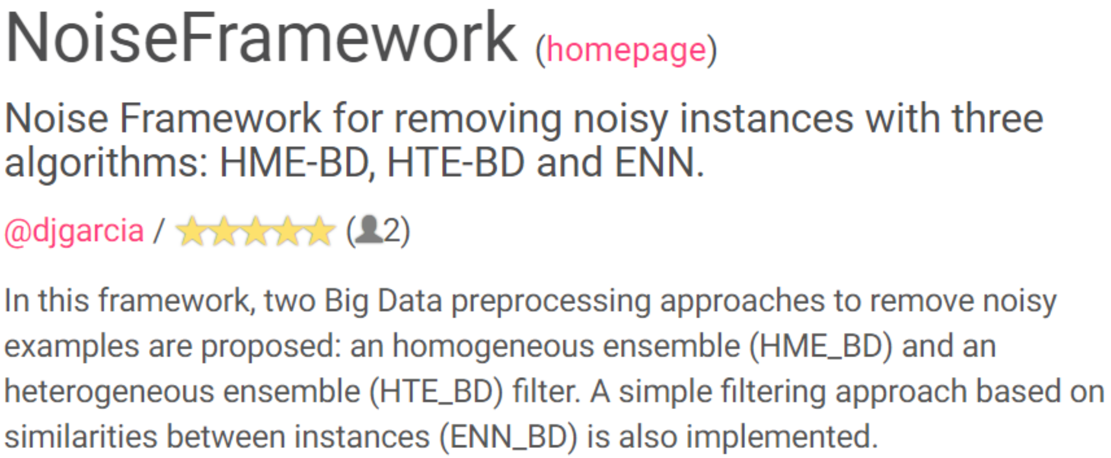
@djgarcia / ★★★★★ (👤2)

This framework implements four distance based Big Data preprocessing algorithms to remove noisy examples: ENN_BD, AllKNN_BD, NCNEdit_BD and RNG_BD filters, with special emphasis in their scalability and performance traits.

https://spark-packages.org/package/djgarcia/SmartFiltering

Noise Filtering

- HME_BD & HTE_BD



NoiseFramework (homepage)

Noise Framework for removing noisy instances with three algorithms: HME-BD, HTE-BD and ENN.

@djgarcia / ★★★★★ (👤2)

In this framework, two Big Data preprocessing approaches to remove noisy examples are proposed: an homogeneous ensemble (HME_BD) and an heterogeneous ensemble (HTE_BD) filter. A simple filtering approach based on similarities between instances (ENN_BD) is also implemented.

https://spark-packages.org/package/djgarcia/NoiseFramework

Add Noise

```
import org.apache.spark.mllib.util._

val noise = 20 //(in %)

val noisyModel = new RandomNoise(train, noise)

val noisyData = noisyModel.runNoise()

noisyData.persist()

noisyData.count()
```

Decision Tree & kNN Clean Data

```
trainDT(train, test, 20)
```

```
trainKNN(train, test)
```

Decision Tree & kNN Clean Data

```
trainDT(train, test, 20)

// 0.7058


trainKNN(train, test)

// 0.7411
```

Decision Tree & kNN Noisy Data

```
trainDT(noisyData, test, 20)
```

```
trainKNN(noisyData, test)
```

Decision Tree & kNN Noisy Data

```
trainDT(noisyData, test, 20)

// 0.6197


trainKNN(noisyData, test)

// 0.6642
```

ENN_BD

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val enn_bd_model = new ENN_BD(noisyData, k)

val enn_bd = enn_bd_model.runFilter()

enn_bd.persist()

enn_bd.count()
```

ENN_BD

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val enn_bd_model = new ENN_BD(noisyData, k)

val enn_bd = enn_bd_model.runFilter()

enn_bd.persist()

enn_bd.count() // 5072
```

DT & kNN Accuracy

```
trainDT(enn_bd, test, 20)
```

```
trainKNN(enn_bd, test)
```

DT & kNN Accuracy

```
trainDT(enn_bd, test, 20)

// 0.6388


trainKNN(enn_bd, test)

// 0.6866
```

NCNEdit_BD

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val ncnedit_bd_model = new NCNEdit_BD(noisyData, k)

val ncnedit_bd = ncnedit_bd_model.runFilter()

ncnedit_bd.persist()

ncnedit_bd.count()
```

NCNEdit_BD

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val ncnedit_bd_model = new NCNEdit_BD(noisyData, k)

val ncnedit_bd = ncnedit_bd_model.runFilter()

ncnedit_bd.persist()

ncnedit_bd.count() // 5831
```

DT & kNN Accuracy

```
trainDT(ncnedit_bd, test, 20)
```

```
trainKNN(ncnedit_bd, test)
```

DT & kNN Accuracy

```
trainDT(ncnedit_bd, test, 20)

// 0.7152


trainKNN(ncnedit_bd, test)

// 0.7412
```

RNG_BD

```
import org.apache.spark.mllib.feature._

val order = true // Order of the graph (true = first, false = second)
val selType = true // Selection type (true = edition, false =
condensation)

val rng_bd_model = new RNG_BD(noisyData, order, selType)

val rng_bd = rng_bd_model.runFilter()

rng_bd.persist()

rng_bd.count()
```

RNG_BD

```
import org.apache.spark.mllib.feature._

val order = true // Order of the graph (true = first, false = second)
val selType = true // Selection type (true = edition, false =
condensation)

val rng_bd_model = new RNG_BD(noisyData, order, selType)

val rng_bd = rng_bd_model.runFilter()

rng_bd.persist()

rng_bd.count() // 7504
```

DT & kNN Accuracy

```
trainDT(rng_bd, test, 20)
```

```
trainKNN(rng_bd, test)
```

DT & kNN Accuracy

```
trainDT(rng_bd, test, 20)

// 0.7135


trainKNN(rng_bd, test)

// 0.7444
```

HME_BD

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4

val hme_bd_model = new HME_BD(noisyData, nTrees, partitions, maxDepthRF,
48151623)

val hme_bd = hme_bd_model.runFilter()

hme_bd.persist()

hme_bd.count()
```

HME_BD

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4

val hme_bd_model = new HME_BD(noisyData, nTrees, partitions, maxDepthRF,
48151623)

val hme_bd = hme_bd_model.runFilter()

hme_bd.persist()

hme_bd.count() // 6624
```

DT & kNN Accuracy

```
trainDT(hme_bd, test, 20)
```

```
trainKNN(hme_bd, test)
```

DT & kNN Accuracy

```
trainDT(hme_bd, test, 20)

// 0.7862


trainKNN(hme_bd, test)

// 0.7913
```

HME_BD on Clean Data

```
val hme_bd_model_clean = new HME_BD(train, nTrees, partitions, maxDepthRF, 48151623)

val hme_bd_clean = hme_bd_model_clean .runFilter()

hme_bd_clean.persist()

hme_bd_clean.count()
```

HME_BD on Clean Data

```
val hme_bd_model_clean = new HME_BD(train, nTrees, partitions,
maxDepthRF, 48151623)

val hme_bd_clean = hme_bd_model_clean.runFilter()

hme_bd_clean.persist()

hme_bd_clean.count() // 7814
```

DT & kNN Accuracy

```
trainDT(hme_bd_clean, test, 20)
```

```
trainKNN(hme_bd_clean, test)
```

DT & kNN Accuracy

```
trainDT(hme_bd_clean, test, 20)

// 0.7947


trainKNN(hme_bd_clean, test)

// 0.7962
```

HTE_BD

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4
val vote = 0 // 0 = majority, 1 = consensus
val k = 1

val hte_bd_model = new HTE_BD(noisyData, nTrees, partitions, vote, k,
maxDepthRF, 48151623)

val hte_bd = hte_bd_model.runFilter()

hte_bd.persist()
hte_bd.count()
```

HTE_BD

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4
val vote = 0 // 0 = majority, 1 = consensus
val k = 1

val hte_bd_model = new HTE_BD(noisyData, nTrees, partitions, vote, k,
maxDepthRF, 48151623)

val hte_bd = hte_bd_model.runFilter()

hte_bd.persist()
hte_bd.count() // 6588
```

DT & kNN Accuracy

```
trainDT(hte_bd, test, 20)
```

```
trainKNN(hte_bd, test)
```

DT & kNN Accuracy

```
trainDT(hte_bd, test, 20)

// 0.7949


trainKNN(hte_bd, test)

// 0.7957
```

HTE_BD on Clean Data

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4
val vote = 0 // 0 = majority, 1 = consensus
val k = 1

val hte_bd_model_clean = new HTE_BD(train, nTrees, partitions, vote, k,
maxDepthRF, 48151623)

val hte_bd_clean = hte_bd_model_clean.runFilter()

hte_bd_clean.persist()
hte_bd_clean.count()
```

HTE_BD on Clean Data

```
import org.apache.spark.mllib.feature._

val nTrees = 100
val maxDepthRF = 10
val partitions = 4
val vote = 0 // 0 = majority, 1 = consensus
val k = 1

val hte_bd_model_clean = new HTE_BD(train, nTrees, partitions, vote, k,
maxDepthRF, 48151623)

val hte_bd_clean = hte_bd_model_clean.runFilter()

hte_bd_clean.persist()
hte_bd_clean.count() // 7817
```

DT & kNN Accuracy

```
trainDT(hte_bd_clean, test, 20)
```

```
trainKNN(hte_bd_clean, test)
```

DT & kNN Accuracy

```
trainDT(hte_bd_clean, test, 20)

// 0.7955


trainKNN(hte_bd_clean, test)

// 0.8012
```

Resume

Method	Accuracy DT	Accuracy kNN	Instances
Clean Data	0.7058	0.7411	10,000
Noisy Data	0.6197	0.6642	10,000
ENN_BD	0.6388	0.6866	5,072
NCNEdit_BD	0.7152	0.7412	5,831
RNG_BD	0.7135	0.7444	7,504
HME_BD	0.7862	0.7913	6,624
HME_BD Clean Data	0.7947	0.7962	7,814
HTE_BD	0.7949	0.7957	6,588
HTE_BD Clean Data	0.7955	0.8012	7,817

Data Reduction

Data Preprocessing

- Data Reduction
  - Feature Selection
  - Instance Reduction
  - Discretization

Spark Shell

```
/opt/spark/bin/spark-shell --packages
JMailloH:kNN_IS:3.0,
djgarcia:SmartReduction:1.0,
djgarcia:Equal-Width-Discretizer:1.0
--jars /home/administrador/datasets/mdlp-mrmr.jar
```

Load data

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


//Load Train & Test

val pathTrain = "file:///home/administrador/datasets/susy-10k-tra.data"
val rawDataTrain = sc.textFile(pathTrain)

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

Train & 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)
}

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)
}
```

Encapsulate Learning Algorithms

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

def trainDT(train: RDD[LabeledPoint], test: RDD[LabeledPoint], maxDepth: Int = 5): Double = {
    val numClasses = 2
    val categoricalFeaturesInfo = Map[Int, Int]()
    val impurity = "gini"
    val maxBins = 32

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

    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()
    testAcc
}
```

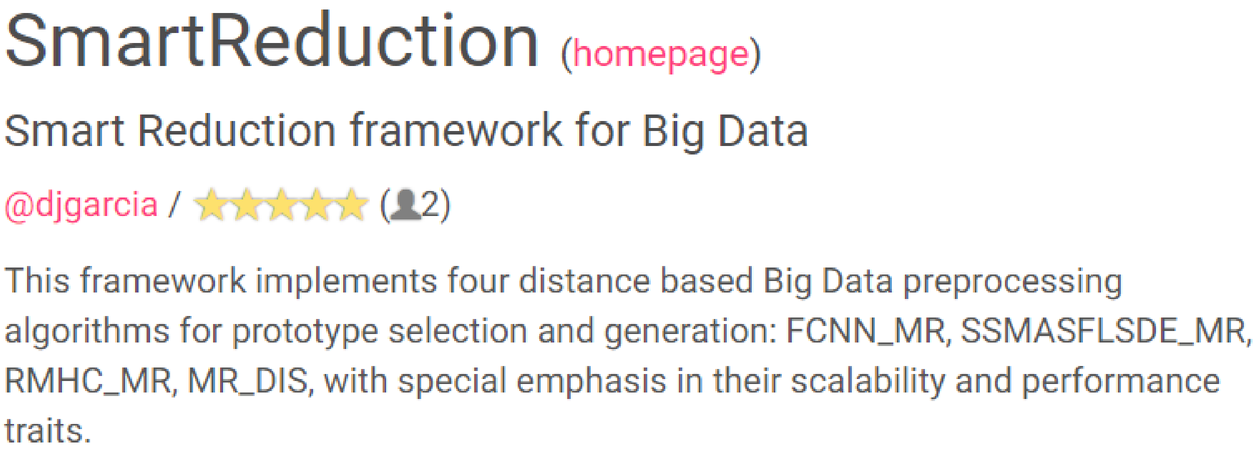
Encapsulate Learning Algorithms

```
import org.apache.spark.mllib.classification.kNN_IS.kNN_IS
import org.apache.spark.mllib.evaluation._
import org.apache.spark.rdd.RDD


def trainKNN(train: RDD[LabeledPoint], test: RDD[LabeledPoint], k: Int = 3): Double = {

    val numClass = train.map(_.label).distinct().collect().length
    val numFeatures = train.first().features.size

    val knn = kNN_IS.setup(train, test, k, 2, numClass, numFeatures, train.getNumPartitions, 2,
-1, 1)
    val predictions = knn.predict(sc)
    val metrics = new MulticlassMetrics(predictions)
    val precision = metrics.precision

    precision

}
```

Instance Reduction

Instance Reduction

- FCNN_MR, SSMASFLSDE_MR, RMHC_MR, MR_DIS



SmartReduction (homepage)

Smart Reduction framework for Big Data

@djgarcia / ★★★★★ (👤2)

This framework implements four distance based Big Data preprocessing algorithms for prototype selection and generation: FCNN_MR, SSMASFLSDE_MR, RMHC_MR, MR_DIS, with special emphasis in their scalability and performance traits.

https://spark-packages.org/package/djgarcia/SmartReduction

FCNN_MR

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val fcnn_mr_model = new FCNN_MR(train, k)

val fcnn_mr = fcnn_mr_model.runPR()

fcnn_mr.persist()

fcnn_mr.count()
```

FCNN_MR

```
import org.apache.spark.mllib.feature._

val k = 3 //number of neighbors

val fcnn_mr_model = new FCNN_MR(train, k)

val fcnn_mr = fcnn_mr_model.runPR()

fcnn_mr.persist()

fcnn_mr.count() // 5584
```

DT & kNN Accuracy

```
trainDT(fcnn_mr, test)
```

```
trainKNN(fcnn_mr, test)
```

DT & kNN Accuracy

```
trainDT(fcnn_mr, test)

// 0.7659


trainKNN(fcnn_mr, test)

// 0.6909
```

RMHC_MR

```
import org.apache.spark.mllib.feature._

val p = 0.1 // Percentage of instances (max 1.0)
val it = 100 // Number of iterations
val k = 3 // Number of neighbors

val rmhc_mr_model = new RMHC_MR(train, p, it, k, 48151623)

val rmhc_mr = rmhc_mr_model.runPR()

rmhc_mr.persist()

rmhc_mr.count()
```

RMHC_MR

```
import org.apache.spark.mllib.feature._

val p = 0.1 // Percentage of instances (max 1.0)
val it = 100 // Number of iterations
val k = 3 // Number of neighbors

val rmhc_mr_model = new RMHC_MR(train, p, it, k, 48151623)

val rmhc_mr = rmhc_mr_model.runPR()

rmhc_mr.persist()

rmhc_mr.count() // 960
```

DT & kNN Accuracy

```
trainDT(rmhc_mr, test)
```

```
trainKNN(rmhc_mr, test)
```

DT & kNN Accuracy

```
trainDT(rmhc_mr, test)

// 0.7282


trainKNN(rmhc_mr, test)

// 0.7229
```

SSMA-SFLSDE_MR

```
import org.apache.spark.mllib.feature._

val ssmasflsde_mr_model = new SSMASFLSDE_MR(train)

val ssmasflsde_mr = ssmasflsde_mr_model.runPR()

ssmasflsde_mr.persist()

ssmasflsde_mr.count()
```

SSMA-SFLSDE_MR

```
import org.apache.spark.mllib.feature._

val ssmasflsde_mr_model = new SSMASFLSDE_MR(train)

val ssmasflsde_mr = ssmasflsde_mr_model.runPR()

ssmasflsde_mr.persist()

ssmasflsde_mr.count() //222
```

DT & kNN Accuracy

```
trainDT(ssmasflsde_mr, test)
```

```
trainKNN(ssmasflsde_mr, test)
```

DT & kNN Accuracy

```
trainDT(ssmasflsde_mr, test)

// 0.7305


trainKNN(ssmasflsde_mr, test)

// 0.7625
```

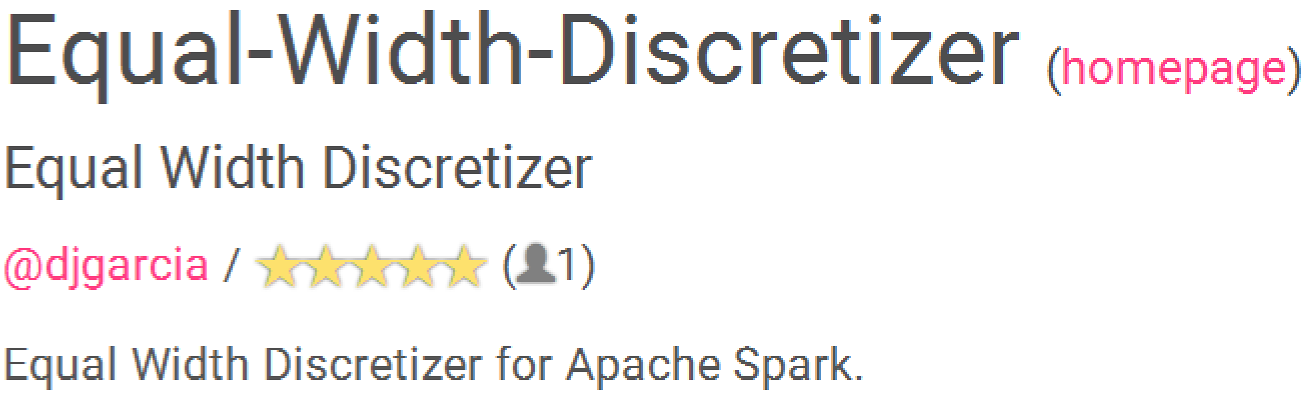
Resume

Method	Accuracy DT	Accuracy kNN	Instances	Reduction
Baseline	0.7876	0.7411	10,000	0.00%
FCNN_MR	0.7659	0.6909	5,584	44.16%
RMHC_MR	0.7282	0.7229	960	90.40%
SSMA-SFLSDE_MR	0.7305	0.7625	222	97.78%

Discretization

Discretization

- Equal-Width-Discretizer



Equal-Width-Discretizer (homepage)

Equal Width Discretizer

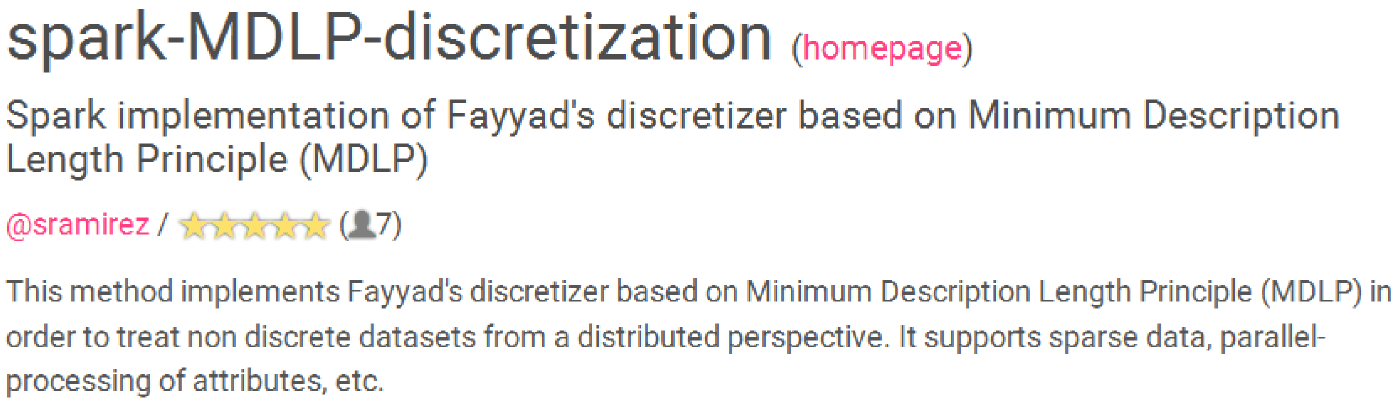
@djgarcia / ★★★★★ (👤1)

Equal Width Discretizer for Apache Spark.

https://spark-packages.org/package/djgarcia/Equal-Width-Discretizer

Discretization

- MDLP



spark-MDLP-discretization (homepage)

Spark implementation of Fayyad's discretizer based on Minimum Description Length Principle (MDLP)

@sramirez / ★★★★★ (👤7)

This method implements Fayyad's discretizer based on Minimum Description Length Principle (MDLP) in order to treat non discrete datasets from a distributed perspective. It supports sparse data, parallel-processing of attributes, etc.

https://spark-packages.org/package/sramirez/spark-MDLP-discretization

EWD

```
import org.apache.spark.mllib.feature._

val nBins = 25 // Number of bins

val discretizerModel = new EqualWidthDiscretizer(train,nBins).calcThresholds()

val discretizedTrain = discretizerModel.discretize(train)
val discretizedTest = discretizerModel.discretize(test)

discretizedTrain.first
discretizedTest.first
```

DT & kNN Accuracy

```
trainDT(discretizedTrain, discretizedTest)
```

```
trainKNN(discretizedTrain, discretizedTest)
```

DT & kNN Accuracy

```
trainDT(discretizedTrain, discretizedTest)

// 0.7704


trainKNN(discretizedTrain, discretizedTest)

// 0.7119
```

MDLP

```
import org.apache.spark.ml.feature.{MDLPDiscretizer, LabeledPoint =>
NewLabeledPoint}

val mdlpTrain = train.map(l => NewLabeledPoint(l.label, l.features.asML)).toDS()
val mdlpTest = test.map(l => NewLabeledPoint(l.label, l.features.asML)).toDS()

val bins = 25

val discretizer = new MDLPDiscretizer()
.setMaxBins(bins)
.setMaxByPart(10000)
.setInputCol("features")
.setLabelCol("label")
.setOutputCol("buckedFeatures")

val model = discretizer.fit(mdlpTrain)
```

MDLP

```
val trainDisc = model.transform(mdlpTrain).rdd.map(row => LabeledPoint(
  row.getAs[Double]("label"),
Vectors.dense(row.getAs[org.apache.spark.ml.linalg.Vector]("buckedFeatures").toAr
ray)
))

val testDisc = model.transform(mdlpTest).rdd.map(row => LabeledPoint(
  row.getAs[Double]("label"),
Vectors.dense(row.getAs[org.apache.spark.ml.linalg.Vector]("buckedFeatures").toAr
ray)
))
```

DT & kNN Accuracy

```
trainDT(trainDisc, testDisc)
```

```
trainKNN(trainDisc, testDisc)
```

DT & kNN Accuracy

```
trainDT(trainDisc, testDisc)

// 0.7967


trainKNN(trainDisc, testDisc)

// 0.7541
```

Feature Selection

ChiSq

```
import org.apache.spark.mllib.feature.ChiSqSelector

val numFeatures = 5
val selector = new ChiSqSelector(numFeatures)
val transformer = selector.fit(train)

val chisqTrain = train.map { lp =>
  LabeledPoint(lp.label, transformer.transform(lp.features))
}

val chisqTest = test.map { lp =>
  LabeledPoint(lp.label, transformer.transform(lp.features))
}

chisqTrain.first.features.size // 5
```

DT & kNN Accuracy

```
trainDT(chisqTrain, chisqTest)
```

```
trainKNN(chisqTrain, chisqTest)
```

DT & kNN Accuracy

```
trainDT(chisqTrain, chisqTest)

// 0.7535


trainKNN(chisqTrain, chisqTest)

// 0.7154
```

PCA

```
import org.apache.spark.mllib.feature.PCA

val numFeatures = 5

val pca = new PCA(5).fit(train.map(_.features))

val projectedTrain = train.map(p => p.copy(features =
pca.transform(p.features)))
val projectedTest = test.map(p => p.copy(features =
pca.transform(p.features)))

projectedTrain.first.features.size // 5
projectedTest.first.features.size // 5
```

DT & kNN Accuracy

```
trainDT(projectedTrain, projectedTest)
```

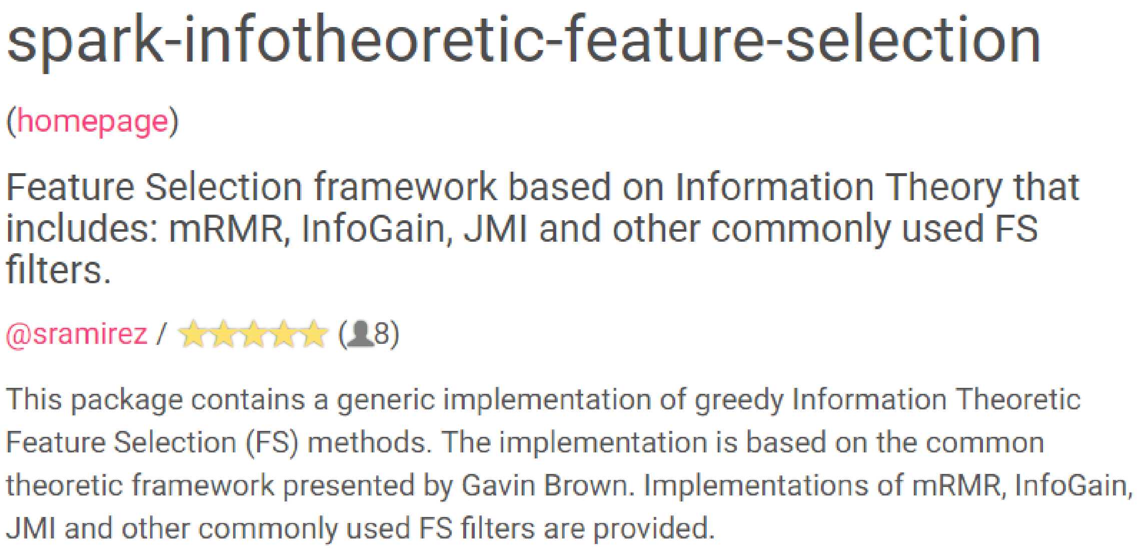
```
trainKNN(projectedTrain, projectedTest)
```

DT & kNN Accuracy

```
trainDT(projectedTrain, projectedTest)

// 0.7446


trainKNN(projectedTrain, projectedTest)

// 0.7085
```

Feature Selection

- mRMR, InfoGain, JMI



spark-infotheoretic-feature-selection

(homepage)

Feature Selection framework based on Information Theory that includes: mRMR, InfoGain, JMI and other commonly used FS filters.

@sramirez / ★★★★★ (👤8)

This package contains a generic implementation of greedy Information Theoretic Feature Selection (FS) methods. The implementation is based on the common theoretic framework presented by Gavin Brown. Implementations of mRMR, InfoGain, JMI and other commonly used FS filters are provided.

https://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection

mRMR

```
import org.apache.spark.mllib.feature._

val criterion = new InfoThCriterionFactory("mrmr")
val nToSelect = 5
val nPartitions = trainDisc.getNumPartitions

val featureSelector = new InfoThSelector(criterion, nToSelect,
nPartitions).fit(trainDisc)

val reducedTrain = testDisc.map(i => LabeledPoint(i.label,
featureSelector.transform(i.features)))
reducedTrain.first()

val reducedTest = mdlpTest.map(i => LabeledPoint(i.label,
featureSelector.transform(i.features)))
```

DT & kNN Accuracy

```
trainDT(reducedTrain, reducedTest)
```

```
trainKNN(reducedTrain, reducedTest)
```

DT & kNN Accuracy

```
trainDT(reducedTrain, reducedTest)

// 0.7923


trainKNN(reducedTrain, reducedTest)

// 0.7218
```

Resume

Method	Accuracy DT	Accuracy kNN
ChiSq	0.7535	0.7154
PCA	0.7085	0.7446
mRMR	0.7923	0.7218

Thank You!

Diego J. García Gil

djgarcia@decsai.ugr.es