Introduction to Big Data Analytics

Chap 4 – Lab: Spark and Data Analytics

Yanwei Fu

School of Data Science, Fudan University

Reference


O'REILLY
Learning
Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

What's wrong with MapReduce?

从MapReduce说起

- 最大缺陷：网络、磁盘I/O

怎么办？

- 放内存：分布式内存计算

MapReduce其他不足：

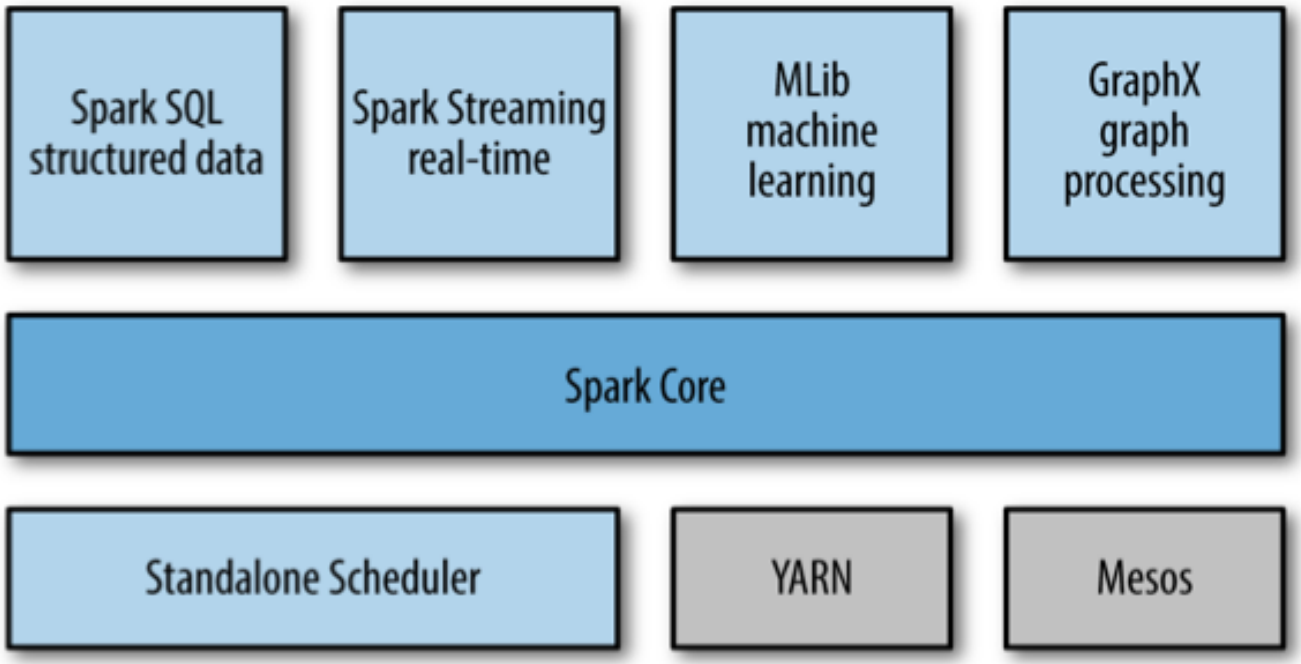
- 抽象层次低，需要手工代码
- 只提供Map和Reduce，表达力欠缺
- 复杂的计算需要大量的Job完成，Job之间的依赖关系是由开发者自己管理的
- 处理逻辑隐藏在代码细节中，没有整体逻辑
- ReduceTask需要等待所有MapTask都完成后才可以开始；
- 时延高，只适用Batch数据处理，不适合交互式数据、实时数据，迭代式数据处理性能比较差。

Who's using Spark?

Spark经典案例

- 腾讯 – 广点通pCTR投放系统。基于日志的快速查询系统构建于Spark之上的Shark，比Hive快2-100倍
- Yahoo – 定向广告业务。基于广告者提供的观看广告并购买产品的样本客户，学习并寻找更多可能转化的用户
- 淘宝 – 阿里搜索和广告业务。从Mahout或自己写的MR解决复杂机器学习到使用Spark、Graphx运用于相关推荐算法
- 优酷土豆 – 视频推荐(图计算)、广告业务等。Spark相比于Hadoop交互查询响应快；模拟广告投放计算效率高、延迟小；机器学习、图计算等迭代计算，大大减少了网络传输、数据落地等

Spark Stack


Spark SQL
structured data
Spark Streaming
real-time
MLib
machine
learning
GraphX
graph
processing
Spark Core
Standalone Scheduler
YARN
Mesos

Spark Core

Basic functionality of Spark, including components for:

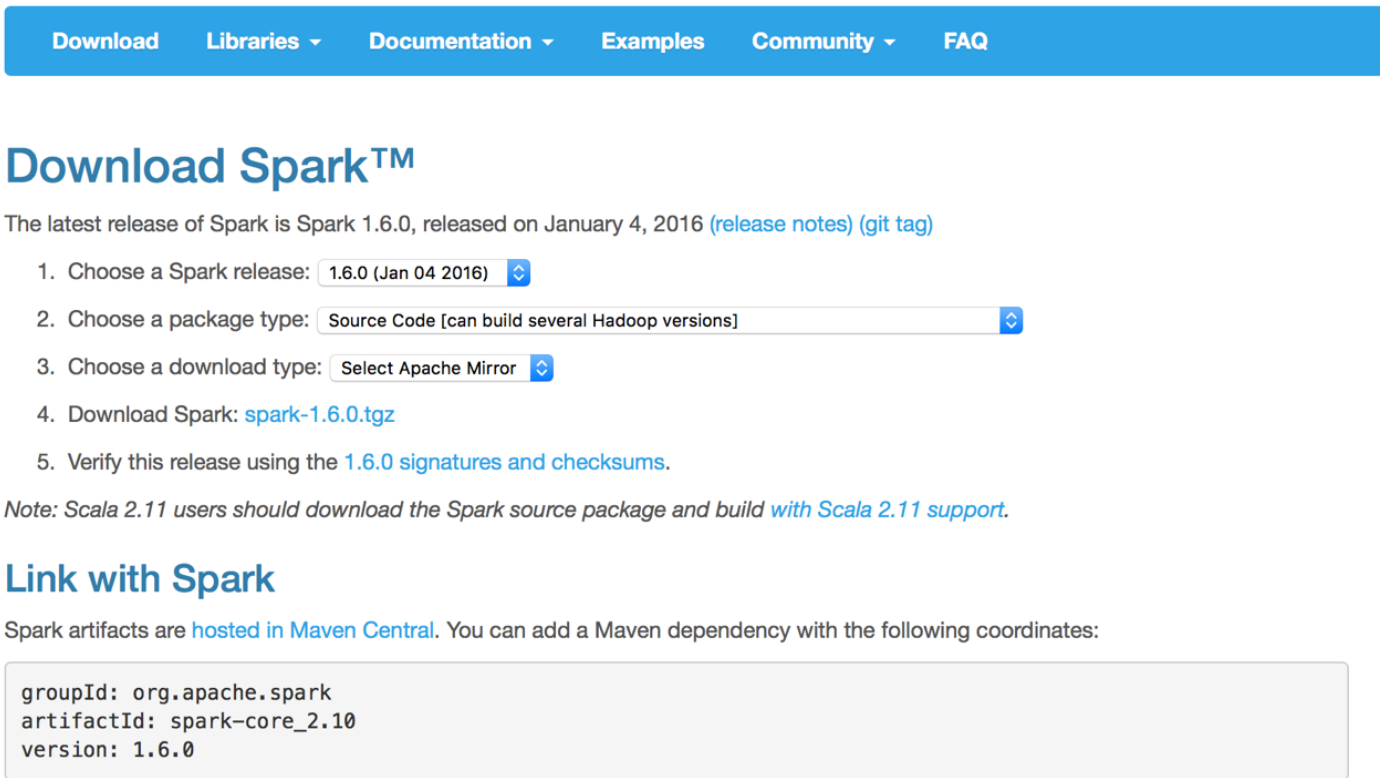
- Task Scheduling
- Memory Management
- Fault Recovery
- Interacting with Storage Systems
- and more

Home to the API that defines resilient distributed datasets (RDDs) - Spark's main programming abstraction.

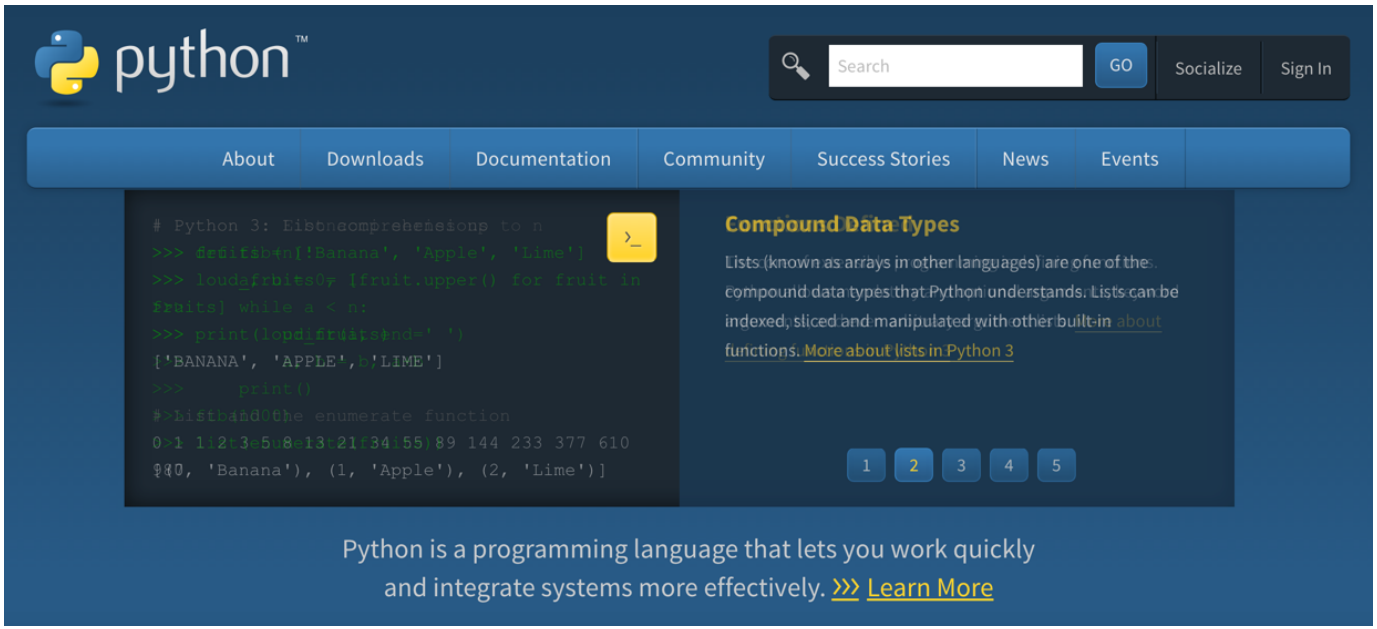
RDD represents a collection of items distributed across many compute nodes that can be manipulated in parallel.

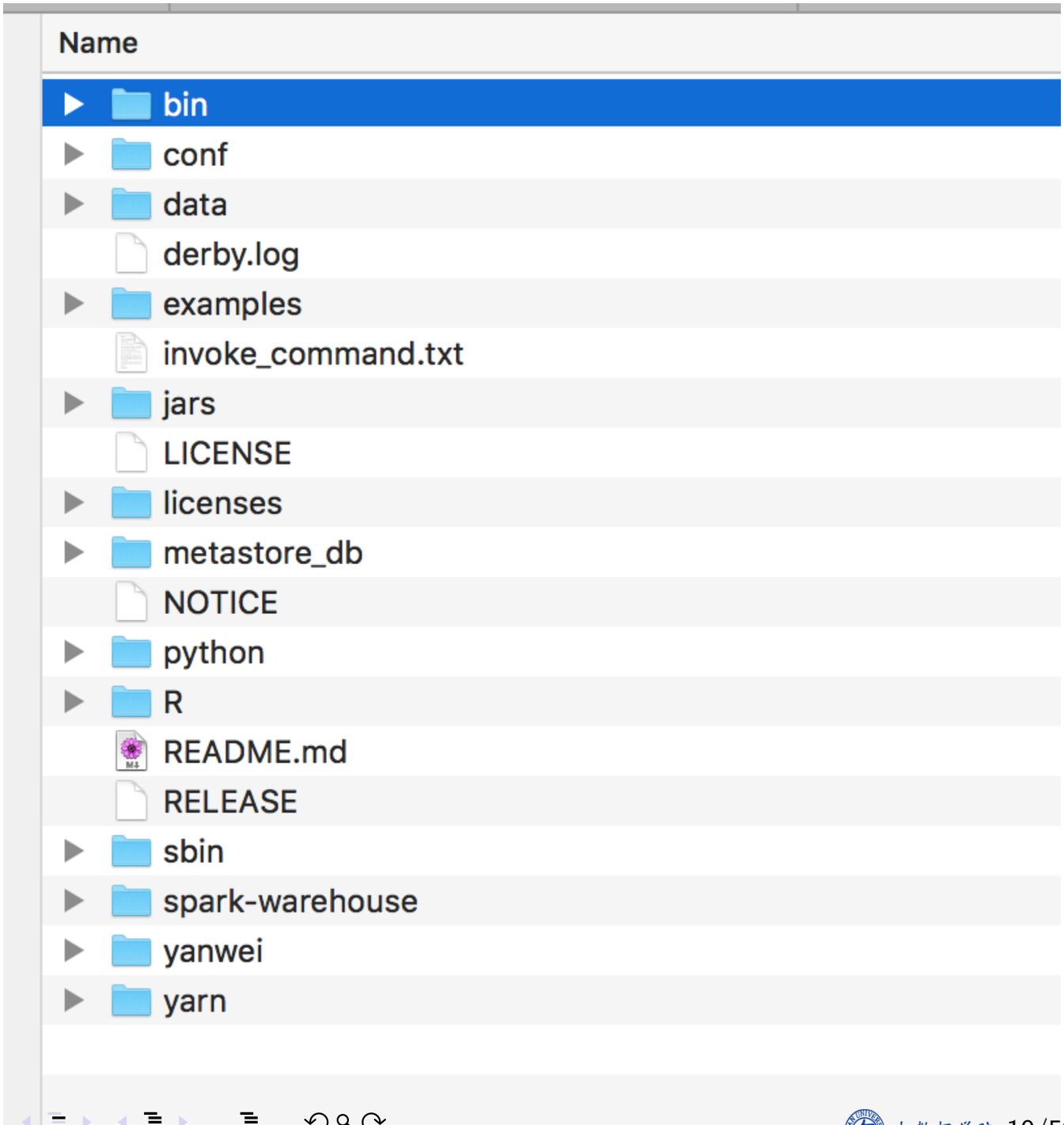
Download Spark

http://spark.apache.org/downloads.html



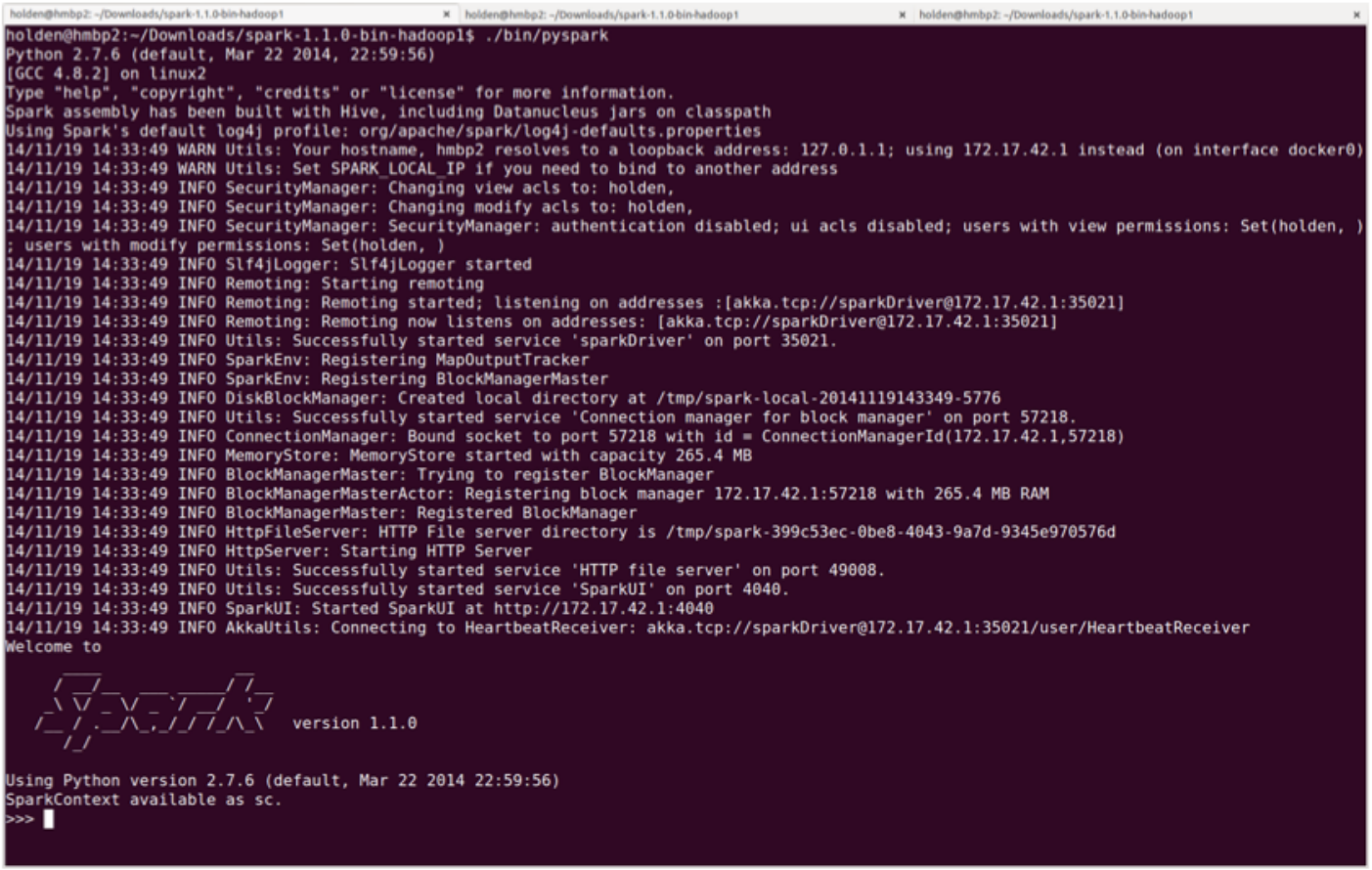
First language to use — Python



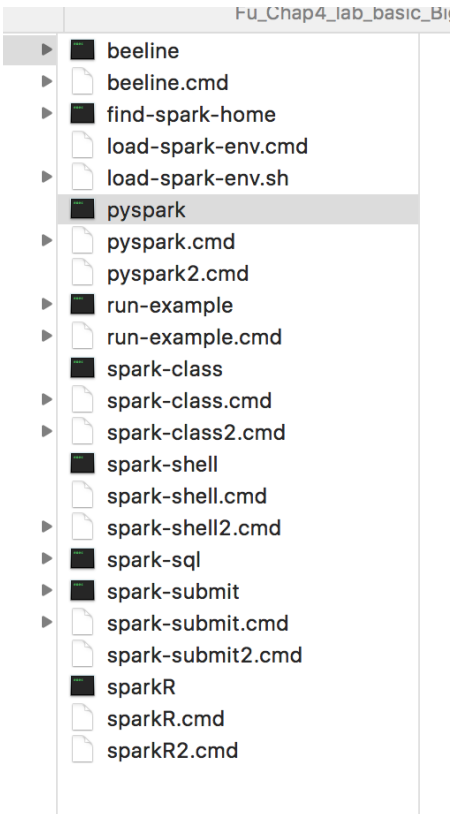

Name
bin
conf
data
derby.log
examples
invoke_command.txt
jars
LICENSE
licenses
metastore_db
NOTICE
python
R
README.md
RELEASE
sbin
spark-warehouse
yanwei
yarn

Spark's Python Shell (PySpark Shell)

bin/pyspark



Next Questions: what's inside of bin folder?




```
#!/usr/bin/env bash

#
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

if [ -z "${SPARK_HOME}" ]; then
  source "$(dirname "$0")"/find-spark-home
fi

source "${SPARK_HOME}"/bin/load-spark-env.sh
export _SPARK_CMD_USAGE="Usage: ./bin/pyspark [options]"

# In Spark 2.0, IPYTHON and IPYTHON_OPTS are removed and pyspark fails to launch if
  either option
# is set in the user's environment. Instead, users should set
  PYSPARK_DRIVER_PYTHON=ipython
# to use IPython and set PYSPARK_DRIVER_PYTHON_OPTS to pass options when starting the
  Python driver
# (e.g. PYSPARK_DRIVER_PYTHON_OPTS='notebook').  This supports full customization of
  the IPython
# and executor Python executables.

# Fail noisily if removed options are set
if [[ -n "$IPYTHON" || -n "$IPYTHON_OPTS" ]]; then
  echo "Error in pyspark startup:"
  echo "IPYTHON and IPYTHON_OPTS are removed in Spark 2.0+. Remove these from the enviro
  exit 1
fi

# Default to standard python interpreter unless told otherwise
if [[ -z "$PYSPARK_DRIVER_PYTHON" ]]; then
  PYSPARK_DRIVER_PYTHON="${PYSPARK_PYTHON:-"python"}"
fi
```

Hello Word in "Word Counting"

In the folder, please input: ./bin/pyspark

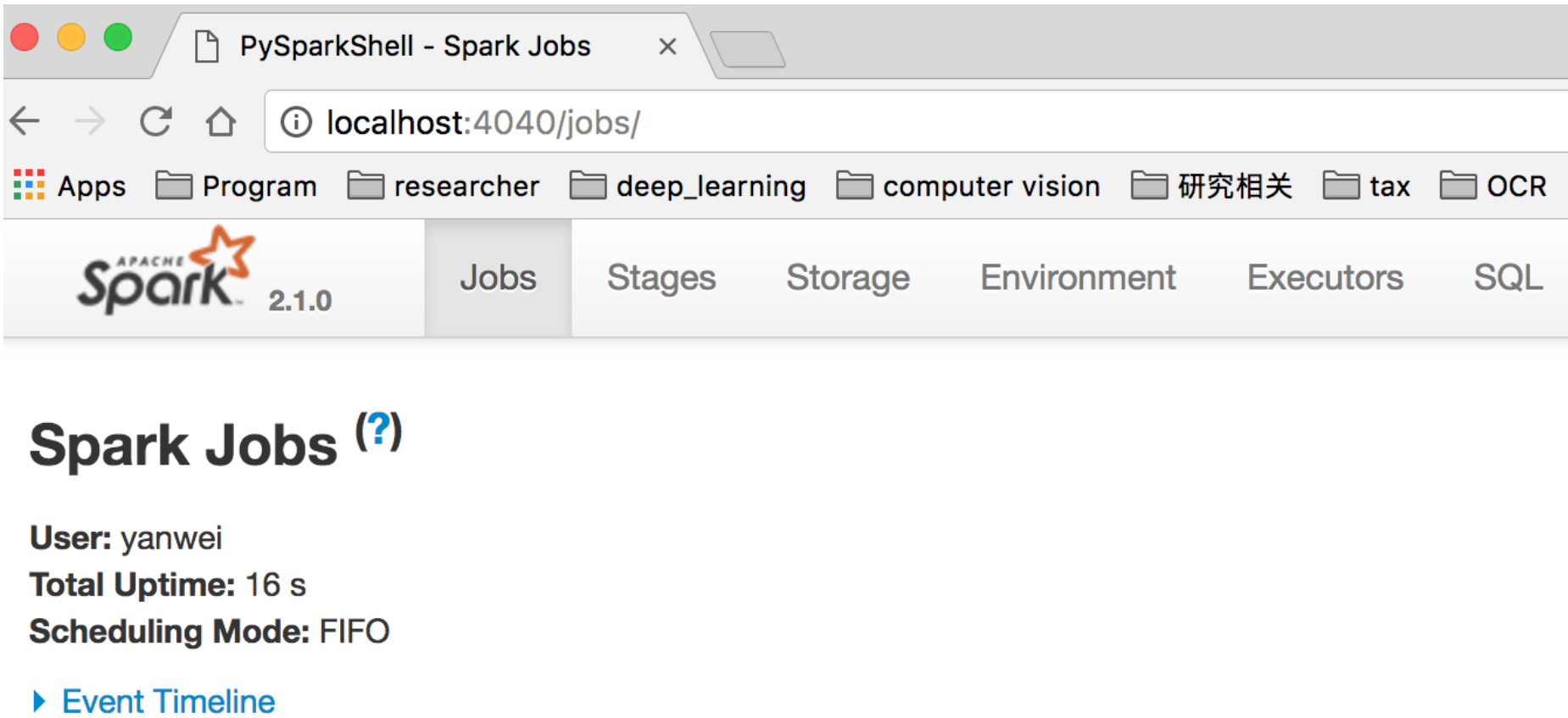
Test installation

Example 2-1. Python line count

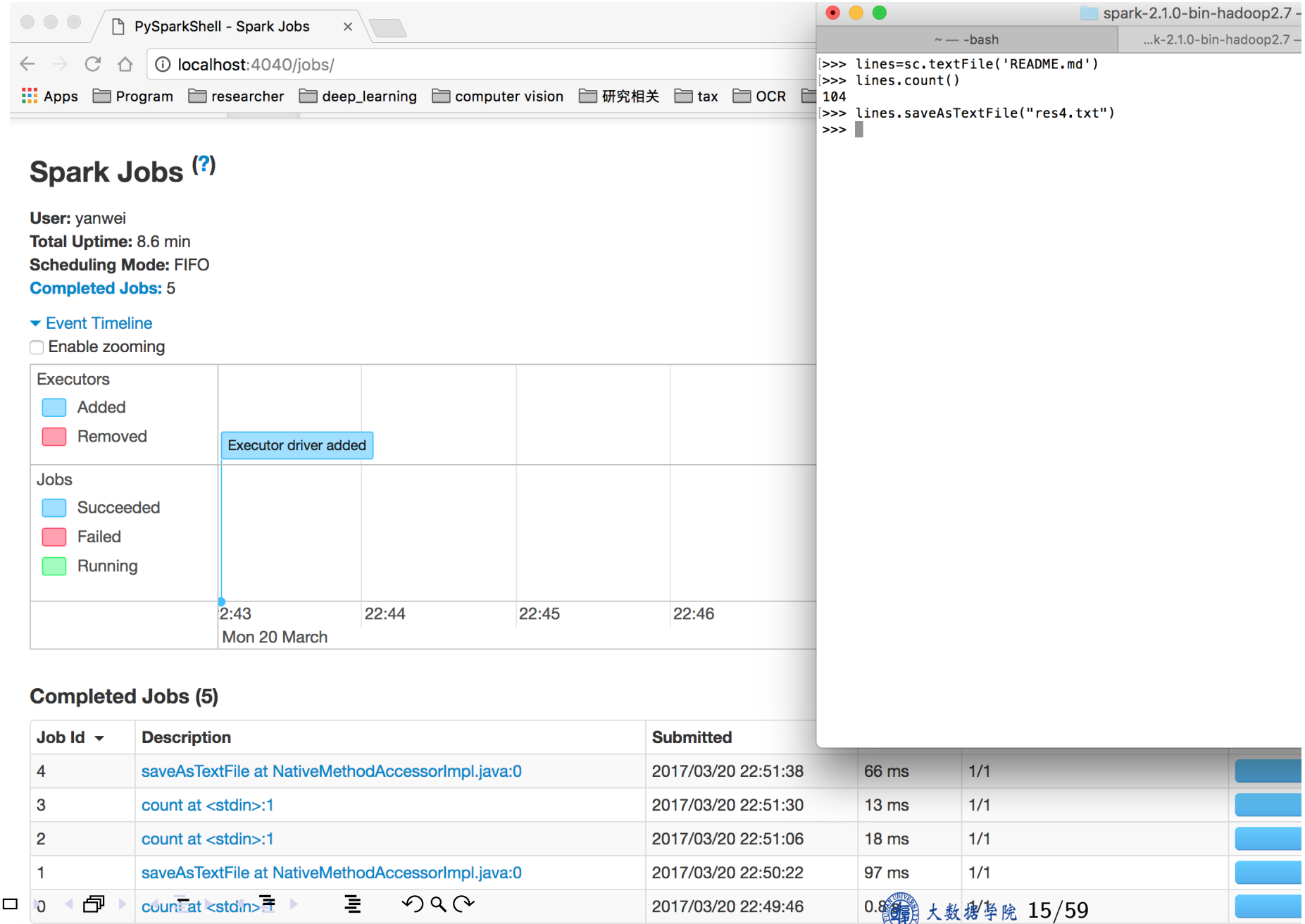
```
>>> lines = sc.textFile("README.md") # Create an RDD called lines

>>> lines.count() # Count the number of items in this RDD
127
>>> lines.first() # First item in this RDD, i.e. first line of README.md
u'# Apache Spark'
```

http://localhost:4040



Running commands changing status of WebUI



Disable logging

You may find the logging statements that get printed in the shell distracting. You can control the verbosity of the logging. To do this, you can create a file in the *conf* directory called *log4j.properties*. The Spark developers already include a template for this file called *log4j.properties.template*. To make the logging less verbose, make a copy of *conf/log4j.properties.template* called *conf/log4j.properties* and find the following line:

```
log4j.rootCategory=INFO, console
```

Then lower the log level so that we show only the WARN messages, and above by changing it to the following:

```
log4j.rootCategory=WARN, console
```

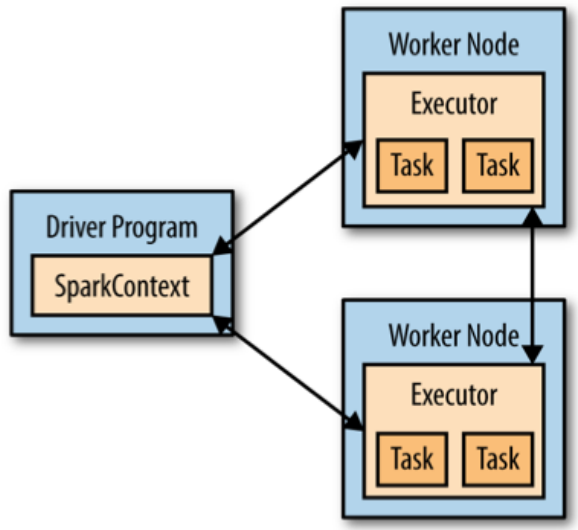
Core Spark Concepts

- At a high level, every Spark application consists of a **driver program** that launches various parallel operations on a cluster.
- The driver program contains your application's main function and defines distributed databases on the cluster, then applies operations to them.
- In the preceding example, the driver program was the Spark shell itself.
- Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster.
- In the shell, a SparkContext is automatically created as the variable called sc.

Driver Programs

Driver programs typically manage a number of nodes called **executors**.

If we run the count() operation on a cluster, different machines might count lines in different ranges of the file.



Example filtering

```
>>> lines = sc.textFile("README.md")

>>> pythonLines = lines.filter(lambda line: "Python" in line)

>>> pythonLines.first()
u'## Interactive Python Shell'
```

lambda —> define functions inline in Python.

```
def hasPython(line):
    return "Python" in line

pythonLines = lines.filter(hasPython)
```

Running as a Standalone Application

In Python, you simply write applications as Python scripts, but you must run them using the `bin/spark-submit` script included in Spark. The `spark-submit` script includes the Spark dependencies for us in Python. This script sets up the environment for Spark's Python API to function. Simply run your script with the line given in Example 2-6.

Example 2-6. Running a Python script

```
bin/spark-submit my_script.py
```

Example — word count

```
import sys
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print >> sys.stderr, "Usage: wordcount <file>"
        exit(-1)
    sc = SparkContext(appName="PythonWordCount")
    lines = sc.textFile(sys.argv[1], 1)
    counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print "%s: %i" % (word, count)

    sc.stop()
```

Resilient Distributed Dataset (RDD) Basics

- An RDD in Spark is an immutable distributed collection of objects.
- Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster.
- Users create RDDs in two ways: by loading an external dataset, or by distributing a collection of objects in their driver program.
- Once created, RDDs offer two types of operations: **transformations** and **actions**.

```
>>> lines = sc.textFile("README.md")
```
<== create RDD

```
>>> pythonLines = lines.filter(lambda line: "Python" in line)
```
<== transformation

```
>>> pythonLines.first()
u'## Interactive Python Shell'
```
<== action

Transformations and actions are different because of the way Spark computes RDDs.
==> Only computes when something is, the first time, in an action.

[What] 什么是Resilient Distributed Datasets （RDD）？

- 不变的、容错的、并行的数据结构
- 显式地将数据存储到磁盘和内存中，并能控制数据的分区

[Why] 为什么要RDD？

- 数据处理常见模型：1. Iterative Algorithms（迭代算法）; 2.Relational Queries（关联查询）; 3.MapReduce; 4. Stream Processing（流式处理）
- RDD支持四种模型

[How] 如何构建RDD？

- 从文件系统
- 通过Scala集合对象并行化生成
- 通过对已存在的RDD transform生成
- 通过改变其他RDD的持久化状态

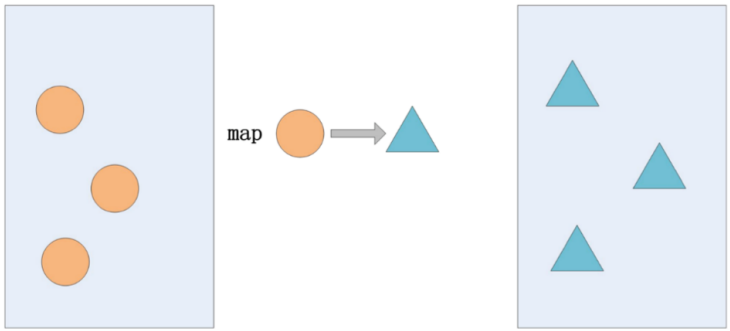
RDD Transformations和Actions

核心API：

- Map：集合元素A=>B，可以变化类型
- Reduce/ReduceByKey：归约计算，得到RDD(K,V)
- Filter：元素个数可能减少，类型不变
- Flattern：把嵌套的集合做成非嵌套

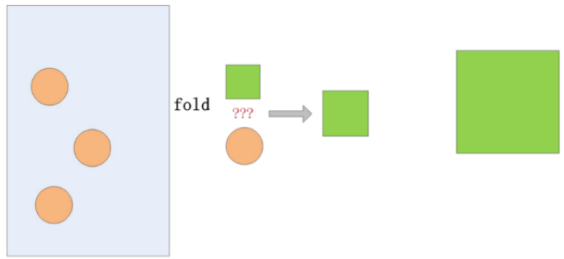
What's that?

MAP运算




```
scala> List(1,2,3).map(_.toString)
res1: List[String] = List(1, 2, 3)
```

FOLD/REDUCE运算

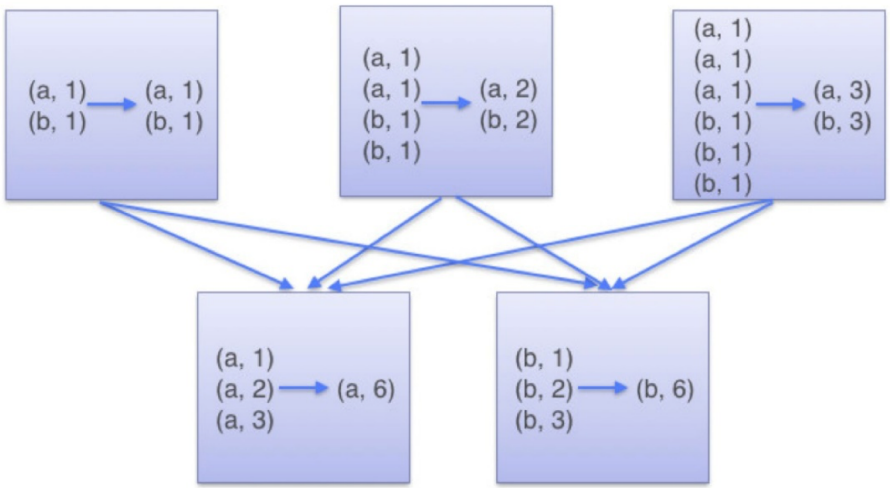



```
scala> List(1,2,3).foldLeft(0)(_ + _)
res3: Int = 6
```

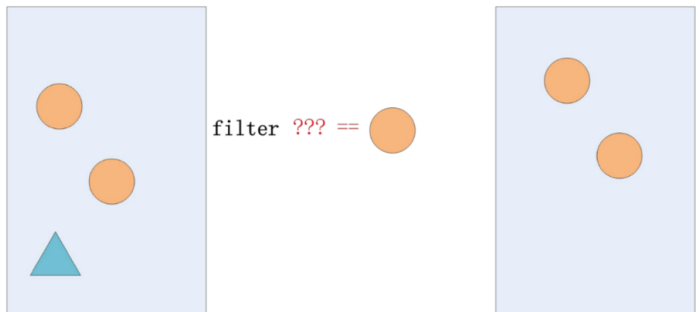
```
scala> List(1,2,3).foldLeft(List[Int]())((acc,i) => i :: acc)
res8: List[Int] = List(3, 2, 1)
```

```
scala> List(1,2,3).reduce(_ * _)
res11: Int = 6
```

ReduceByKey

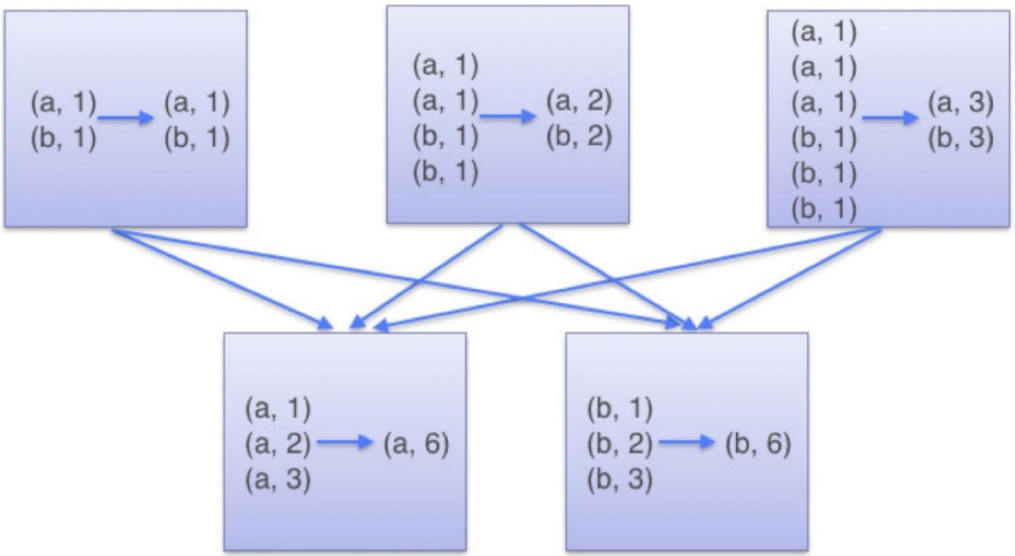


FILTER运算




```
scala> List(1,2,2).filter(_ == 2)
res2: List[Int] = List(2, 2)
```

ReduceByKey



FLATTEN运算


```
scala> List(List(1,2),Nil,List(3)).flatten
res10: List[Int] = List(1, 2, 3)
```

RDD Transformations和Actions

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Persistance in Spark

- By default, RDDs are computed each time you run an action on them.
- If you like to reuse an RDD in multiple actions, you can ask Spark to persist it using RDD.persist().
- RDD.persist() will then store the RDD contents in memory and reuse them in future actions.
- Persisting RDDs on disk instead of memory is also possible.
- The behavior of not persisting by default seems to be unusual, but it makes sense for big data.

Example 3-4. Persisting an RDD in memory

```
>>> pythonLines.persist

>>> pythonLines.count()
2

>>> pythonLines.first()
u'## Interactive Python Shell'
```

To summarize, every Spark program and shell session will work as follows:

1. Create some input RDDs from external data.
2. Transform them to define new RDDs using transformations like `filter()`.
3. Ask Spark to `persist()` any intermediate RDDs that will need to be reused.
4. Launch actions such as `count()` and `first()` to kick off a parallel computation, which is then optimized and executed by Spark.

Parallelizing in Spark

The simplest way to create RDDs is to take an existing collection in your program and pass it to SparkContext's parallelize() method. But, that needs all dataset in memory on one machine.

Example 3-5. parallelize() method in Python

```
lines = sc.parallelize(["pandas", "i like pandas"])
```

Spark file loading

Table 5-1. Common supported file formats

Format name	Structured	Comments
Text files	No	Plain old text files. Records are assumed to be one per line.
JSON	Semi	Common text-based format, semistructured; most libraries require one record per line.
CSV	Yes	Very common text-based format, often used with spreadsheet applications.
SequenceFiles	Yes	A common Hadoop file format used for key/value data.
Protocol buffers	Yes	A fast, space-efficient multilanguage format.
Object files	Yes	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.

Loading and Saving

Example 5-1. Loading a text file in Python

```
input = sc.textFile("file:///home/holden/repos/spark/README.md")
```

Example 5-5. Saving as a text file in Python

```
result.saveAsTextFile(outputFile)
```

Example 5-6. Loading unstructured JSON in Python

```
import json
data = input.map(lambda x: json.loads(x))
```

Example 5-9. Saving JSON in Python

```
(data.filter(lambda x: x['lovesPandas']).map(lambda x: json.dumps(x))
  .saveAsTextFile(outputFile))
```

Loading and Saving (II)

Example 5-12. Loading CSV with textFile() in Python

```
import csv
import StringIO
...
def loadRecord(line):
    """Parse a CSV line"""
    input = StringIO.StringIO(line)
    reader = csv.DictReader(input, fieldnames=["name", "favouriteAnimal"])
    return reader.next()
input = sc.textFile(inputFile).map(loadRecord)
```

Example 5-18. Writing CSV in Python

```
def writeRecords(records):
    """Write out CSV lines"""
    output = StringIO.StringIO()
    writer = csv.DictWriter(output, fieldnames=["name", "favoriteAnimal"])
    for record in records:
        writer.writerow(record)
    return [output.getvalue()]

pandaLovers.mapPartitions(writeRecords).saveAsTextFile(outputFile)
```

Loading and Saving (III)

Example 5-20. Loading a SequenceFile in Python

```
val data = sc.sequenceFile(inFile,
  "org.apache.hadoop.io.Text", "org.apache.hadoop.io.IntWritable")
```

File compression options

Table 5-3. Compression options

Format	Splittable	Average compression speed	Effectiveness on text	Hadoop compression codec	Pure Java	Native	Comments
gzip	N	Fast	High	`org.apache.hadoop.io.compress.GzipCodec`	Y	Y	
lzo	Y[6]	Very fast	Medium	`com.hadoop.compression.lzo.LzoCodec`	Y	Y	LZO requires installation on every worker node
bzip2	Y	Slow	Very high	`org.apache.hadoop.io.compress.BZip2Codec`	Y	Y	Uses pure Java for splittable version
zlib	N	Slow	Medium	`org.apache.hadoop.io.compress.DefaultCodec`	Y	Y	Default compression codec for Hadoop
Snappy	N	Very Fast	Low	`org.apache.hadoop.io.compress.SnappyCodec`	N	Y	There is a pure Java port of Snappy but it is not yet available in Spark/ Hadoop

Example of Spark Programming

Example 6-1. Sample call log entry in JSON, with some fields removed


```
{"address":"address here", "band":"40m","callsign":"KK6JLK","city":"SUNNYVALE",
"contactlat":"37.384733","contactlong":"-122.032164",
"county":"Santa Clara","dxcc":"291","fullname":"MATTHEW McPherrin",
"id":57779,"mode":"FM","mylat":"37.751952821","mylong":"-122.4208688735",...}
```


Example 6-2. Accumulator empty line count in Python

```
file = sc.textFile(inputFile)
# Create Accumulator[Int] initialized to 0
blankLines = sc.accumulator(0)

def extractCallSigns(line):
    global blankLines   # Make the global variable accessible
    if (line == ""):
        blankLines += 1
    return line.split(" ")

callSigns = file.flatMap(extractCallSigns)
callSigns.saveAsTextFile(outputDir + "/callsigns")
print "Blank lines: %d" % blankLines.value
```

Numeric RDD Operations

Table 6-2. Summary statistics available from StatsCounter

Method	Meaning
`count()`	Number of elements in the RDD
`mean()`	Average of the elements
`sum()`	Total
`max()`	Maximum value
`min()`	Minimum value
`variance()`	Variance of the elements
`sampleVariance()`	Variance of the elements, computed for a sample
`stdev()`	Standard deviation
`sampleStdev()`	Sample standard deviation

Removing outliers

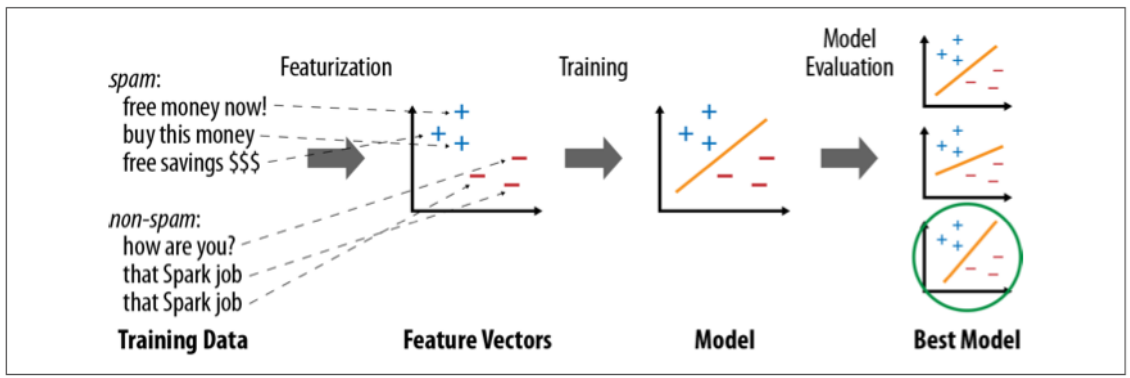
Example 6-19. Removing outliers in Python

```
# Convert our RDD of strings to numeric data so we can compute stats and
# remove the outliers.
distanceNumerics = distances.map(lambda string: float(string))
stats = distanceNumerics.stats()
stddev = std.stdev()
mean = stats.mean()
reasonableDistances = distanceNumerics.filter(
  lambda x: math.fabs(x - mean) < 3 * stddev)
print reasonableDistances.collect()
```

Machine Learning Library in Spark — MLlib

An example of using MLlib for text classification task, e.g., identifying spammy emails.

1. Start with an RDD of strings representing your messages.
2. Run one of MLlib's *feature extraction* algorithms to convert text into numerical features (suitable for learning algorithms); this will give back an RDD of vectors.
3. Call a classification algorithm (e.g., logistic regression) on the RDD of vectors; this will give back a model object that can be used to classify new points.
4. Evaluate the model on a test dataset using one of MLlib's evaluation functions.



Example: Spam Detection

Example 11-1. Spam classifier in Python

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.feature import HashingTF
from pyspark.mllib.classification import LogisticRegressionWithSGD

spam = sc.textFile("spam.txt")
normal = sc.textFile("normal.txt")

# Create a HashingTF instance to map email text to vectors of 10,000 features.
tf = HashingTF(numFeatures = 10000)
# Each email is split into words, and each word is mapped to one feature.
spamFeatures = spam.map(lambda email: tf.transform(email.split(" ")))
normalFeatures = normal.map(lambda email: tf.transform(email.split(" ")))

# Create LabeledPoint datasets for positive (spam) and negative (normal) examples.

positiveExamples = spamFeatures.map(lambda features: LabeledPoint(1, features))
negativeExamples = normalFeatures.map(lambda features: LabeledPoint(0, features))
trainingData = positiveExamples.union(negativeExamples)
trainingData.cache() # Cache since Logistic Regression is an iterative algorithm.

# Run Logistic Regression using the SGD algorithm.
model = LogisticRegressionWithSGD.train(trainingData)

# Test on a positive example (spam) and a negative one (normal). We first apply
# the same HashingTF feature transformation to get vectors, then apply the model.
posTest = tf.transform("O M G GET cheap stuff by sending money to ...".split(" "))
negTest = tf.transform("Hi Dad, I started studying Spark the other ...".split(" "))
print "Prediction for positive test example: %g" % model.predict(posTest)
print "Prediction for negative test example: %g" % model.predict(negTest)
```

Term Frequency — Inverse Document Frequency (TF-IDF)

The value of word is reduced more if it is used frequently across all the documents in the dataset.

To calculate the inverse document frequency, the document frequency (DF) for each word is first calculated. Document frequency is the number of documents the word occurs in. The number of times a word occurs in a document isn't counted in document frequency. Then, the inverse document frequency or IDF_i for a word, w_i, is

$$IDF_i = \frac{1}{DF_i}$$

$$W_i = TF_i \cdot IDF_i = TF_i \cdot \frac{N}{DF_i} \quad \text{or} \quad W_i = TF_i \cdot \log\frac{N}{DF_i}$$

Feature Extraction Example — TF-IDF

Example 11-7. Using HashingTF in Python

```
>>> from pyspark.mllib.feature import HashingTF

>>> sentence = "hello hello world"
>>> words = sentence.split()  # Split sentence into a list of terms
>>> tf = HashingTF(10000)  # Create vectors of size S = 10,000
>>> tf.transform(words)
SparseVector(10000, {3065: 1.0, 6861: 2.0})

>>> rdd = sc.wholeTextFiles("data").map(lambda (name, text): text.split())
>>> tfVectors = tf.transform(rdd)   # Transforms an entire RDD
```

Example 11-8. Using TF-IDF in Python

```
from pyspark.mllib.feature import HashingTF, IDF

# Read a set of text files as TF vectors
rdd = sc.wholeTextFiles("data").map(lambda (name, text): text.split())
tf = HashingTF()
tfVectors = tf.transform(rdd).cache()

# Compute the IDF, then the TF-IDF vectors
idf = IDF()
idfModel = idf.fit(tfVectors)
tfIdfVectors = idfModel.transform(tfVectors)
```

Linear Regression

Example 11-10. Linear regression in Python

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.regression import LinearRegressionWithSGD

points = # (create RDD of LabeledPoint)
model = LinearRegressionWithSGD.train(points, iterations=200, intercept=True)
print "weights: %s, intercept: %s" % (model.weights, model.intercept)
```

Classifiers

Logistic regression

Logistic regression is a binary classification method that identifies a linear separating plane between positive and negative examples. In MLlib, it takes `LabeledPoints` with label 0 or 1 and returns a `LogisticRegressionModel` that can `predict` new points.

Support Vector Machines

Support Vector Machines, or SVMs, are another binary classification method with linear separating planes, again expecting labels of 0 or 1. They are available through the `SVMWithSGD` class, with similar parameters to linear and logisitic regression. The returned `SVMModel` uses a threshold for prediction like `LogisticRegressionModel`.

Naive Bayes

Naive Bayes is a multiclass classification algorithm that scores how well each point belongs in each class based on a linear function of the features. It is commonly used in text classification with TF-IDF features, among other applications. MLlib implements Multinomial Naive Bayes, which expects nonnegative frequencies (e.g., word frequencies) as input features.

Logistic regression

Logistic regression describes a kind of classification model in which the predictor variables are combined with linear weights and then passed through a soft-limit function that limits the output to the range from 0 to 1. Logistic regression is closely related to other models such as a perceptron (where the soft limit is replaced by a hard limit), neural networks (where multiple layers of linear combination and soft limiting are used) and naive Bayes (where the linear weights are determined strictly by feature frequencies assuming independence). Logistic regression can't separate all possible classes, but in very high dimensional problems or where you can introduce new variables by combining other predictors, this is much less of a problem. The mathematical simplicity of logistic regression allows very efficient and effective learning algorithms to be derived.

Stochastic Gradient Descent (SGD)

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \sum_{i=1}^{n} Q_i(w),$$

where the parameter w is to be estimated and where typically each summand function $Q_i()$ is associated with the i-th observation in the data set (used for training).

- Choose an initial vector of parameters w and learning rate α.
- Randomly shuffle examples in the training set.
- Repeat until an approximate minimum is obtained:
  - For $i = 1, 2, ..., n$, do:
    - $w := w - \alpha \nabla Q_i(w)$.

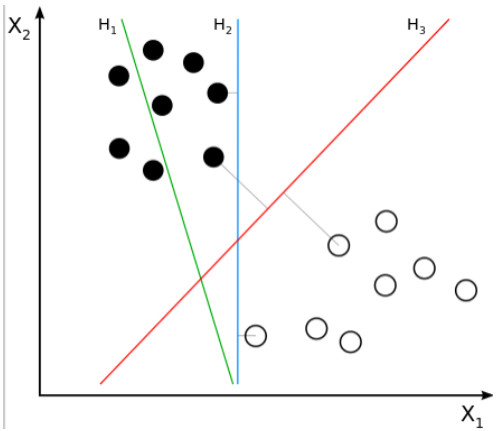
Let's suppose we want to fit a straight line $y = w_1 + w_2 x$ to a training set of two-dimensional points $(x_1, y_1), \ldots, (x_n, y_n)$ using least squares. The objective function to be minimized is:

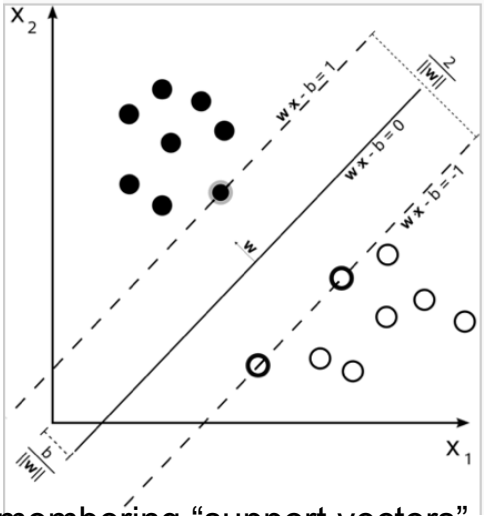
$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

The last line in the above pseudocode for this specific problem will become:

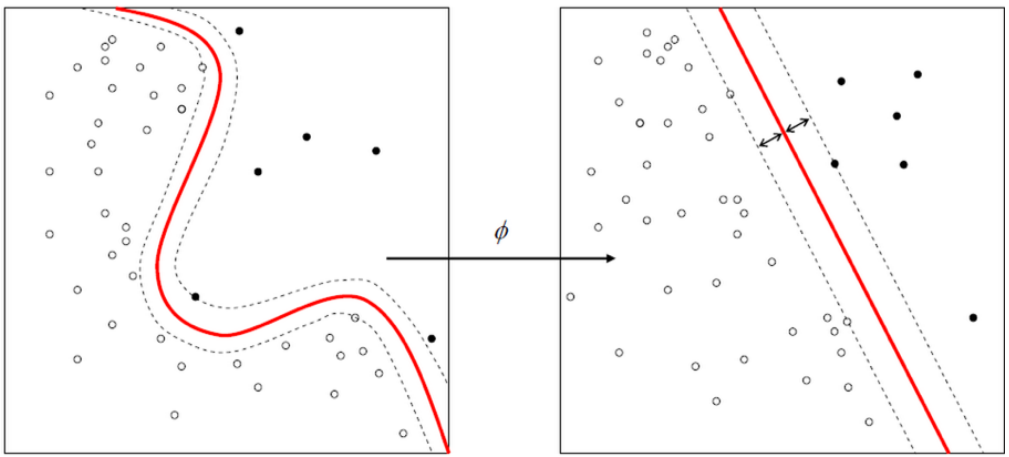
$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \begin{bmatrix} \sum_{i=1}^{n} 2(w_1 + w_2 x_i - y_i) \\ \sum_{i=1}^{n} 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

Support Vector Machine (SVM)





maximize boundary distances; remembering “support vectors”



nonlinear kernels

Training set:

sex	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Classifier using Gaussian distribution assumptions:

sex	mean (height)	variance (height)	mean (weight)	variance (weight)	mean (foot size)	variance (foot size)
male	5.855	3.5033e-02	176.25	1.2292e+02	11.25	9.1667e-01
female	5.4175	9.7225e-02	132.5	5.5833e+02	7.5	1.6667e+00

$$posterior(male) = \frac{P(male)\, p(height|male)\, p(weight|male)\, p(footsize|male)}{evidence}$$

Test Set:

sex	height (feet)	weight (lbs)	foot size(inches)
sample	6	130	8

$$evidence = P(male)\, p(height|male)\, p(weight|male)\, p(footsize|male) + P(female)\, p(height|female)\, p(weight|female)\, p(footsize|female)$$

$$P(male) = 0.5$$

$$p(\text{height}|\text{male}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(6-\mu)^2}{2\sigma^2}\right) \approx 1.5789,$$

$$p(\text{weight}|\text{male}) = 5.9881 \cdot 10^{-6}$$
$$p(\text{foot size}|\text{male}) = 1.3112 \cdot 10^{-3}$$
$$\text{posterior numerator (male)} = \text{their product} = 6.1984 \cdot 10^{-9}$$

$$P(\text{female}) = 0.5$$
$$p(\text{height}|\text{female}) = 2.2346 \cdot 10^{-1}$$
$$p(\text{weight}|\text{female}) = 1.6789 \cdot 10^{-2}$$
$$p(\text{foot size}|\text{female}) = 2.8669 \cdot 10^{-1}$$
$$\text{posterior numerator (female)} = \text{their product} = 5.3778 \cdot 10^{-4}$$

==> female

Decision trees and Random Forests

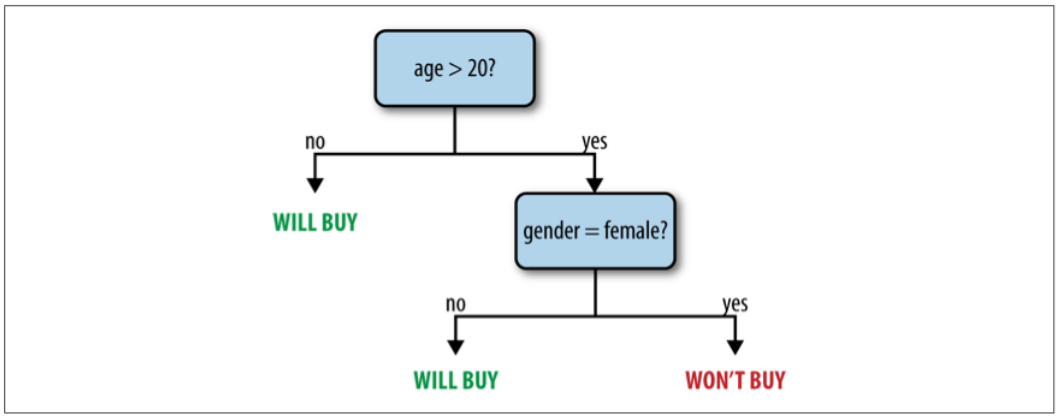


Figure 11-2. An example decision tree predicting whether a user might buy a product

In MLlib, you can train trees using the `mllib.tree.DecisionTree` class, through the static methods `trainClassifier()` and `trainRegressor()`.

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1$ through y_n, bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For b = 1 through B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a decision or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

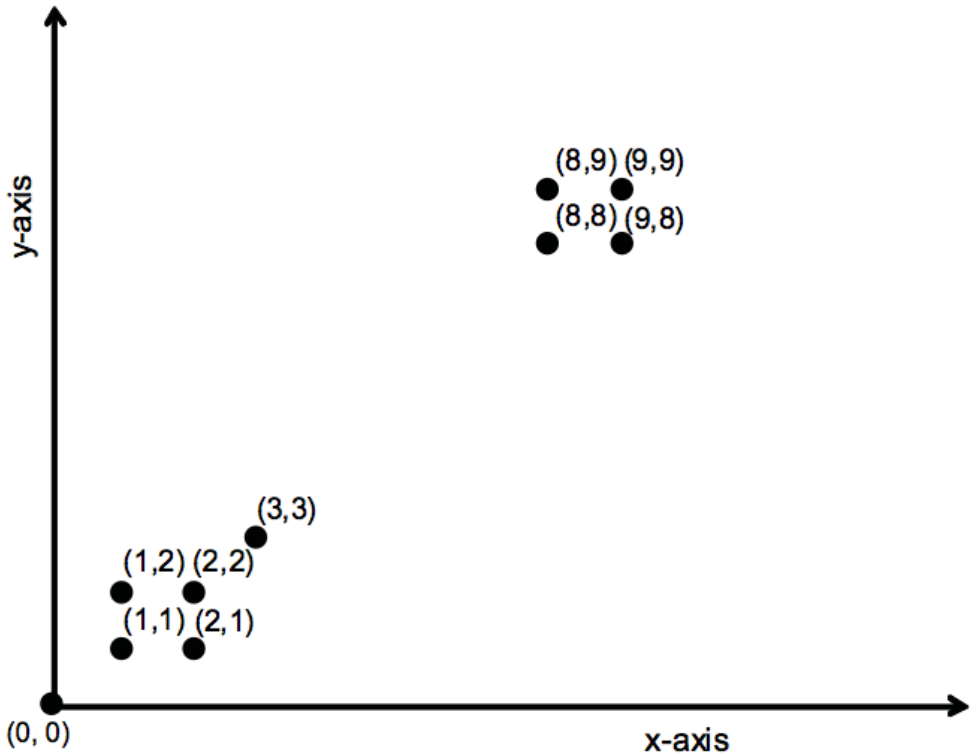
$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x')$$

or by taking the majority vote in the case of decision trees.

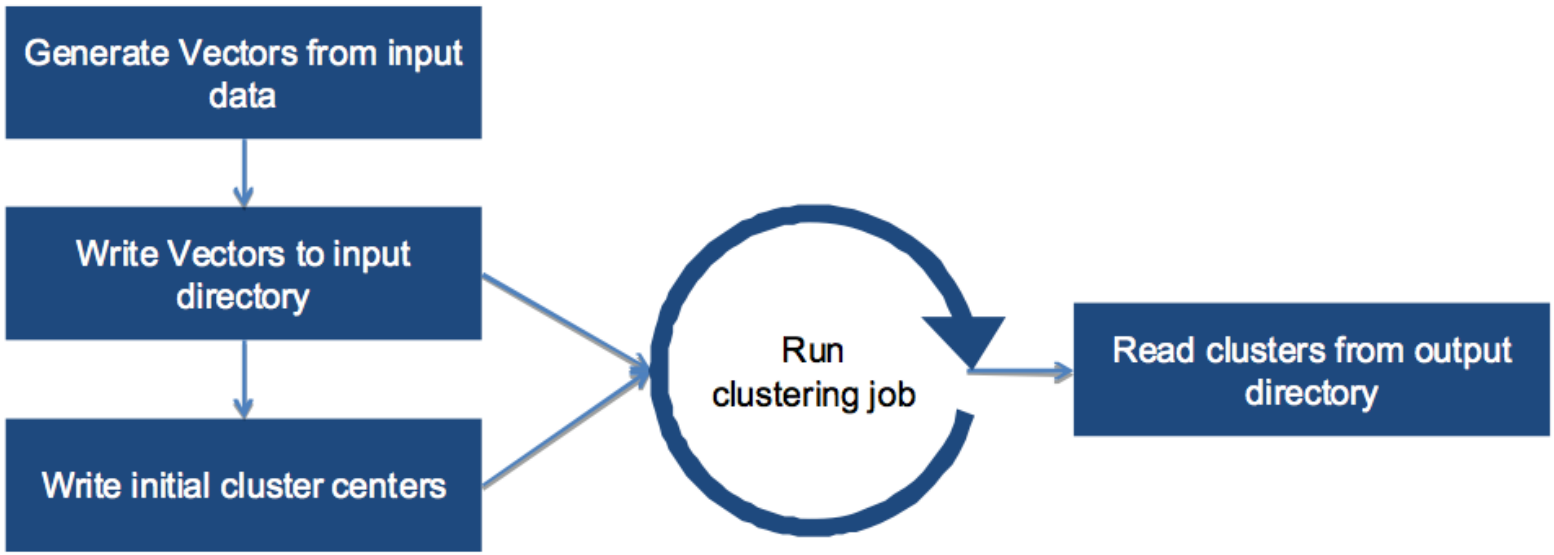
Random forest uses a modified tree learning algorithm that selects, at each candidate split in learning process, a random subset of the features.

Clustering example

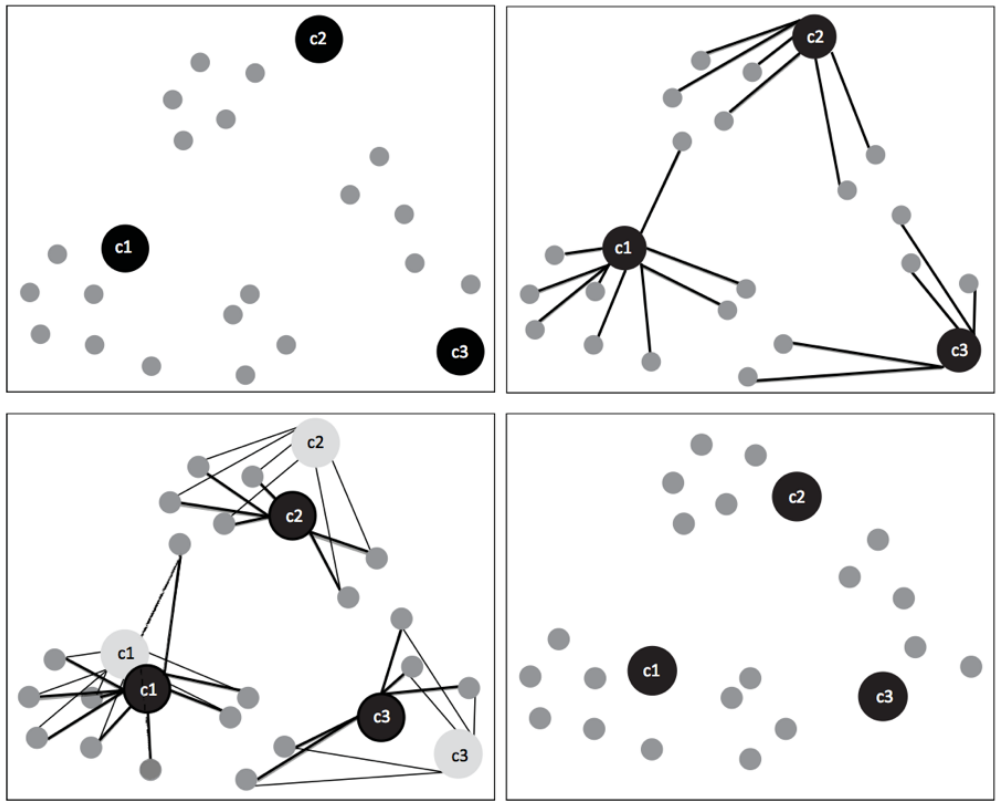
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



Steps on clustering



K-mean clustering



K-means clustering in action. Starting with three random points as centroids (top left), the map stage (top right) assigns each point to the cluster nearest to it. In the reduce stage (bottom left), the associated points are averaged out to produce the new location of the centroid, leaving you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

Clustering

K-means

MLlib includes the popular K-means algorithm for clustering, as well as a variant called K-means|| that provides better initialization in parallel environments.[5] K-means|| is similar to the K-means++ initialization procedure often used in single-node settings.

`initializationMode`
: The method to initialize cluster centers, which can be either "k-means||" or "random"; `k-means||` (the default) generally leads to better results but is slightly more expensive.

`maxIterations`
: Maximum number of iterations to run (default: `100`).

`runs`
: Number of concurrent runs of the algorithm to execute. MLlib's K-means supports running from multiple starting positions concurrently and picking the best result, which is a good way to get a better overall model (as K-means runs can stop in local minima).

Like other algorithms, you invoke K-means by creating a `mllib.clustering.KMeans` object (in Java/Scala) or calling `KMeans.train` (in Python).

Collaborative Filtering and Recommendation

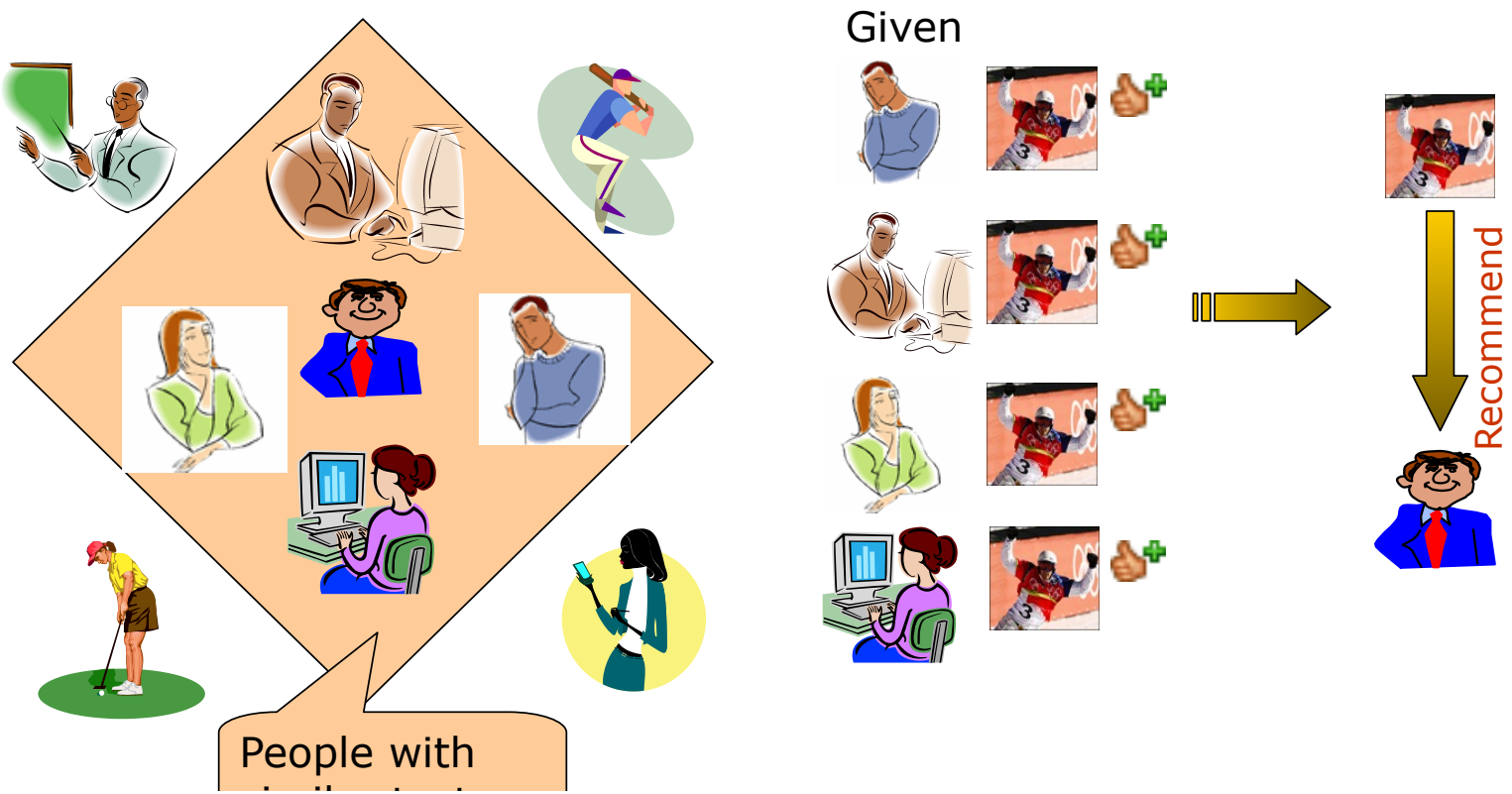
Collaborative filtering is a technique for recommender systems wherein users' ratings and interactions with various products are used to recommend new ones. Collaborative filtering is attractive because it only needs to take in a list of user/product interactions: either "explicit" interactions (i.e., ratings on a shopping site) or "implicit" ones (e.g., a user browsed a product page but did not rate the product).

Alternating Least Squares

MLlib includes an implementation of Alternating Least Squares (ALS), a popular algorithm for collaborative filtering that scales well on clusters.[6] It is located in the `mllib.recommendation.ALS` class.

Collaborative Filtering (CF)

Leveraging opinions of like-minded users



Dimensionality Reduction

Principal component analysis

Given a dataset of points in a high-dimensional space, we are often interested in reducing the dimensionality of the points so that they can be analyzed with simpler tools. For example, we might want to plot the points in two dimensions, or just reduce the number of features to train models more effectively.

Singular value decomposition

MLlib also provides the lower-level singular value decomposition (SVD) primitive. The SVD factorizes an $m \times n$ matrix A into three matrices $A \approx U\Sigma V^T$, where:

- U is an orthonormal matrix, whose columns are called left singular vectors.
- Σ is a diagonal matrix with nonnegative diagonals in descending order, whose diagonals are called singular values.
- V is an orthonormal matrix, whose columns are called right singular vectors.

应用场景

Hadoop – 极大数据量 > TB - PB级

- 单次海量数据的离线分析处理
- 大规模Web信息搜索
- 数据密集型并行计算

Spark – 内存可容纳 ≈TB级

- 多次操作特定数据集，迭代运算
- 搜索引擎 – PageRank
- 计算相似 – Single Source Shortest Path (单源最短路径)

性能

- Hadoop：适合不能全部读入内存；单次读取、类似 ETL （抽取、转换、加载）操作的任务，比如数据转化、数据整合等时
- Spark：适合数据不太大内存放得下，重复读取同样数据迭代计算

上手

- Hadoop：Java编写，需要学习语法，有一些工具（Pig、Hive等）简化
- Spark：Scala、Java和Python，还支持交互式命令模式

兼容性

- Spark兼容Hadoop数据源

容错

- Hadoop：硬盘静态数据，硬盘驱动失败处自动重启执行
- Spark：内存失效，需要手动设置checkpoint

数据处理

- Hadoop：批处理
- Spark：实时/批数据处理，迭代任务