
DAMLAS Part 2

Spark Introduction

+ Homework for Tuesday, 7/13



James G. Shanahan ^{1,2}

¹Church and Duncan Group, ²iSchool UC Berkeley, CA

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Part 2 Lecture 2

July 13, 2016

Homework for 7/13/2016

- **Finish NaiveBayes stateless**
- **Write STAR proposal for any project**
 - * STAR (Situation, Task, Action, Result) - 100 words regarding your proposed projects
- **Install Spark on your local computers**
 - See section later in these slides
 - Please try the following notebook on your local Spark
 - <http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/ya26nezbyz0d1ly/1-Spark-conf-test.ipynb>
 - WordCount:
<https://www.dropbox.com/s/uI0I3q98w54dr8x/WordCount.ipynb?dl=0>
 - In Spark word count example, play around by adding features. (For example, filter words out, etc.)
 - Familiarize with lambda functions
- **Read Chapters 2, 3, and 4 in the following text book**
 - https://www.dropbox.com/s/m4xxds3po5byyg6/Learning_Spark.pdf?dl=0

Quiz1: Word Count

WordCount Quiz Notebook

- <http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/ul0l3q98w54dr8x/WordCountQuiz.ipynb>
-
- **WordCount Solution Notebook**
 - <http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/m5nj01q50b6gg8b/WordCountSolution.ipynb>

Quiz 2: Prime Numbers

- **Quiz notebook**
 - <https://www.dropbox.com/s/nahywnjvnhl6g/4-Spark-conf-test-QUIZ-PrimeNums.ipynb?dl=0>
- **Solution notebook**
 - <https://www.dropbox.com/s/vb7hwfn8iqu6r1y/4-Spark-conf-test-QUIZ-PrimeNums-Solution.ipynb?dl=0>

Quiz 3:

- **Quiz notebook**

- <https://www.dropbox.com/s/7fugb8ieuyh4rxo/3-Spark-example-logs-QUIZ.ipynb?dl=0>

Quiz # 4 Review hashjoins in Spark and some EDA:

- **Quiz Notebook**

- <https://www.dropbox.com/s/gchzblskg7bhzxw/1-Airlines-HashsideJoinVia-Broadcast-and-EDA-Quiz.ipynb?dl=0>

Fill in the missing code for Kmeans

- **Given**
 - Linear Regression with a Doodle factor:
 - <https://www.dropbox.com/s/uahs0pny9slzs95/LinearRegression-Notebook-Challenge.ipynb?dl=0>
 - Linear regression
 - <https://www.dropbox.com/s/zu05h38etpee6np/data.csv?dl=0>
 - And a helper Notebook for Kmeans:
 - <https://www.dropbox.com/s/pn1kzcib4iylwke/EM-KmeansQuiz.ipynb?dl=0>
 - Kmeans Data
<https://www.dropbox.com/s/lsz50we5sqtqssj/data.csv?dl=0>
- **Solution**
 - <https://www.dropbox.com/s/tcsoxkn2t4t33yz/EM-Kmeans.ipynb?dl=0>

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistence
- Animated Example
- Pair RDDs
- Word count example

Part 3: Machine Learning in Spark

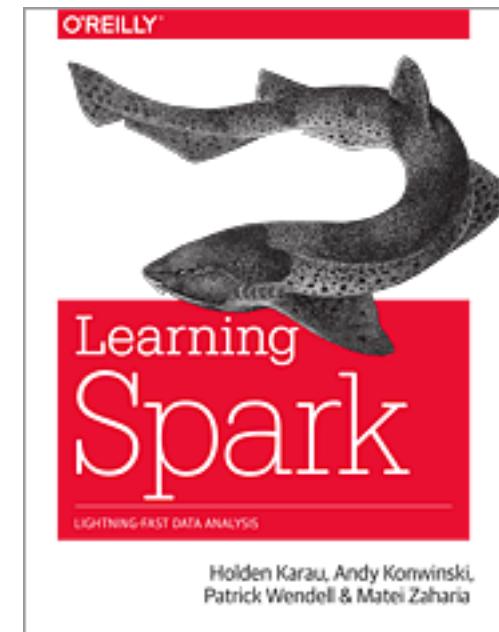
- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Reference material

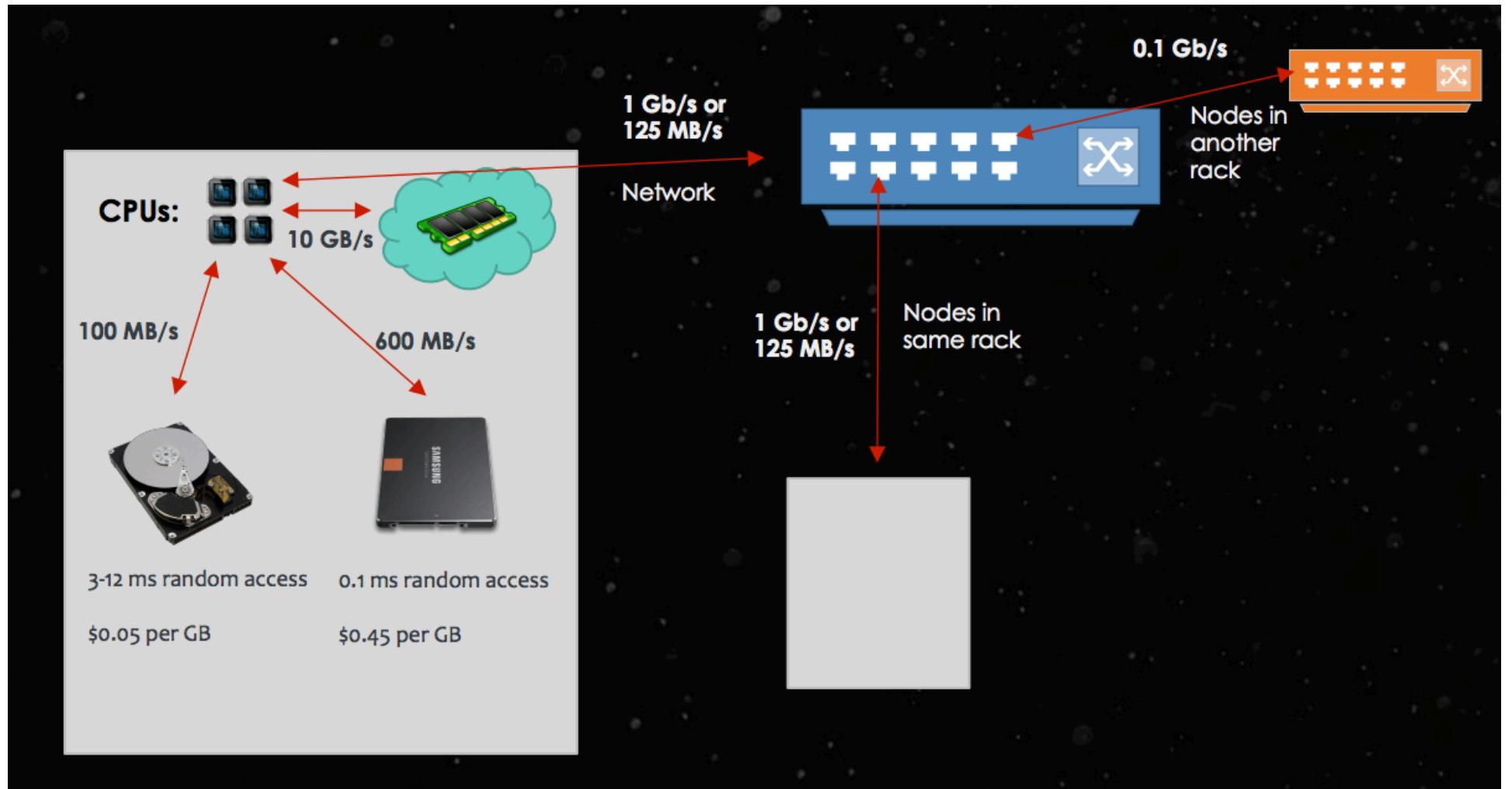
- **Book: Learning Spark: Lightning-Fast Big Data Analysis**
 - By Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia
 - Chapters 2, 3, and 4, and beyond
- <https://spark.apache.org/docs/latest/programming-guide.html>



Spark Online Resources

- <http://spark.apache.org/research.html>
- [Spark 1.4.1 released \(Jul 15, 2015\)](#)
- [Spark Summit 2015 Videos Posted\(Jun 29, 2015\)](#)
- **Reza Zadeh's presentation on Machine Learning in Spark @ Spark Summit 2015 in San Francisco (assumes strong understanding of programming in Spark)**

CPU $\leftarrow \rightarrow$ Memory 100X faster than HD Across network: 10X to 100X

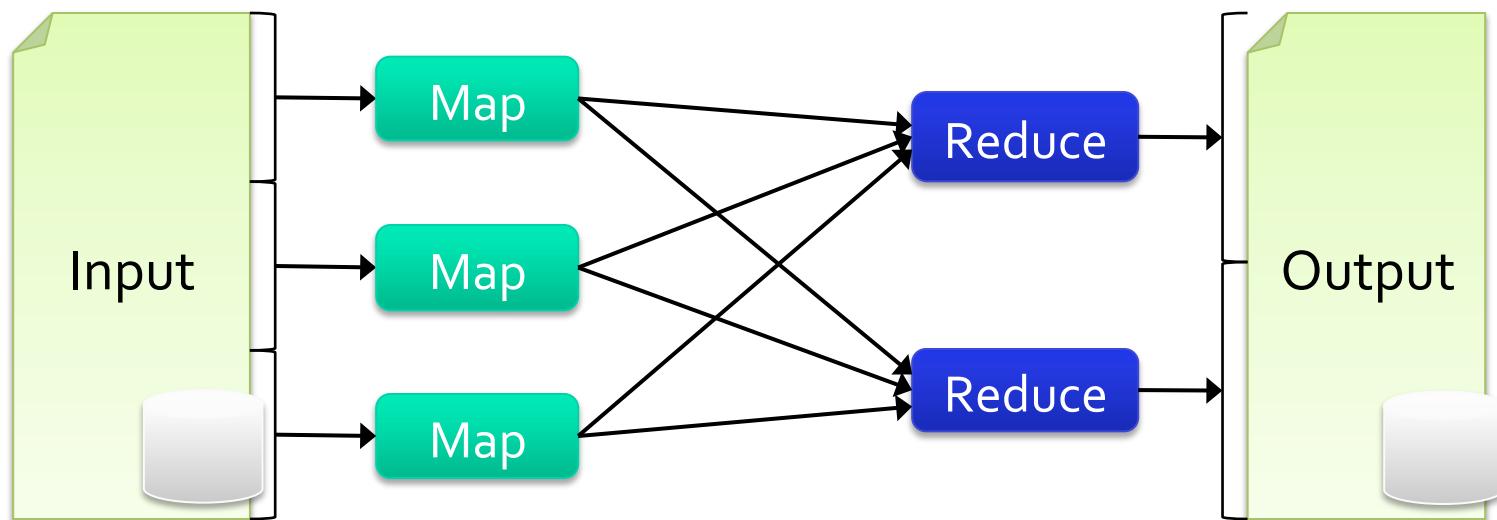


CPU $\leftarrow \rightarrow$ Memory 100X faster than HD
Across network: on same rack 10X; across the network 100X

Motivation: Read from file; Process; Write to file

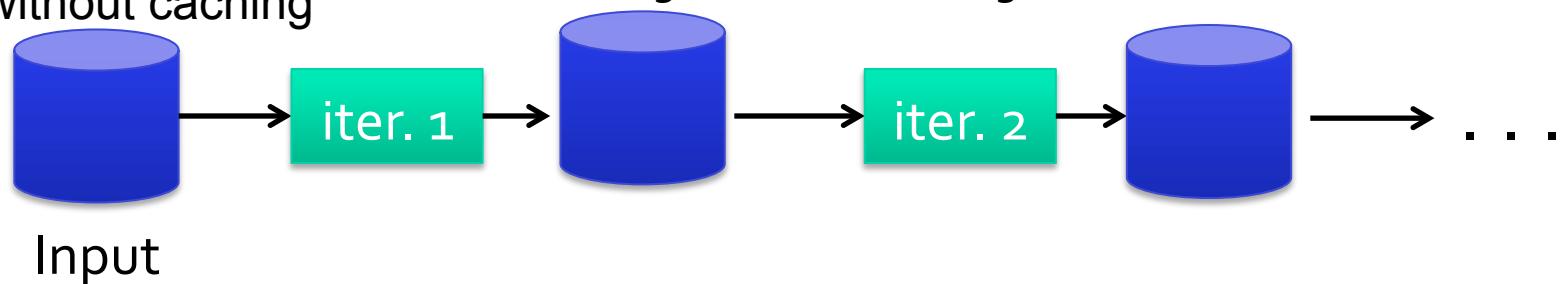
Current popular programming models for clusters transform data flowing from stable storage to stable storage (fault tolerant storage)

e.g., MapReduce:

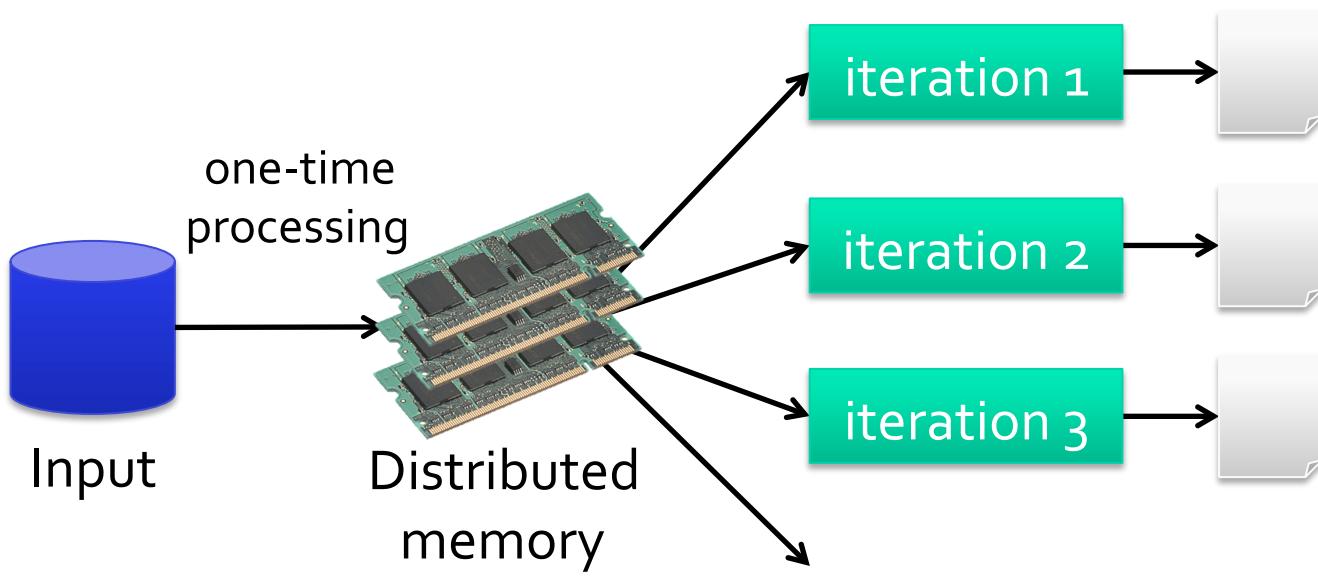


Goal: Keep Working Set in RAM

Hadoop MapReduce
Spark without caching



(Stable storage to stable storage)



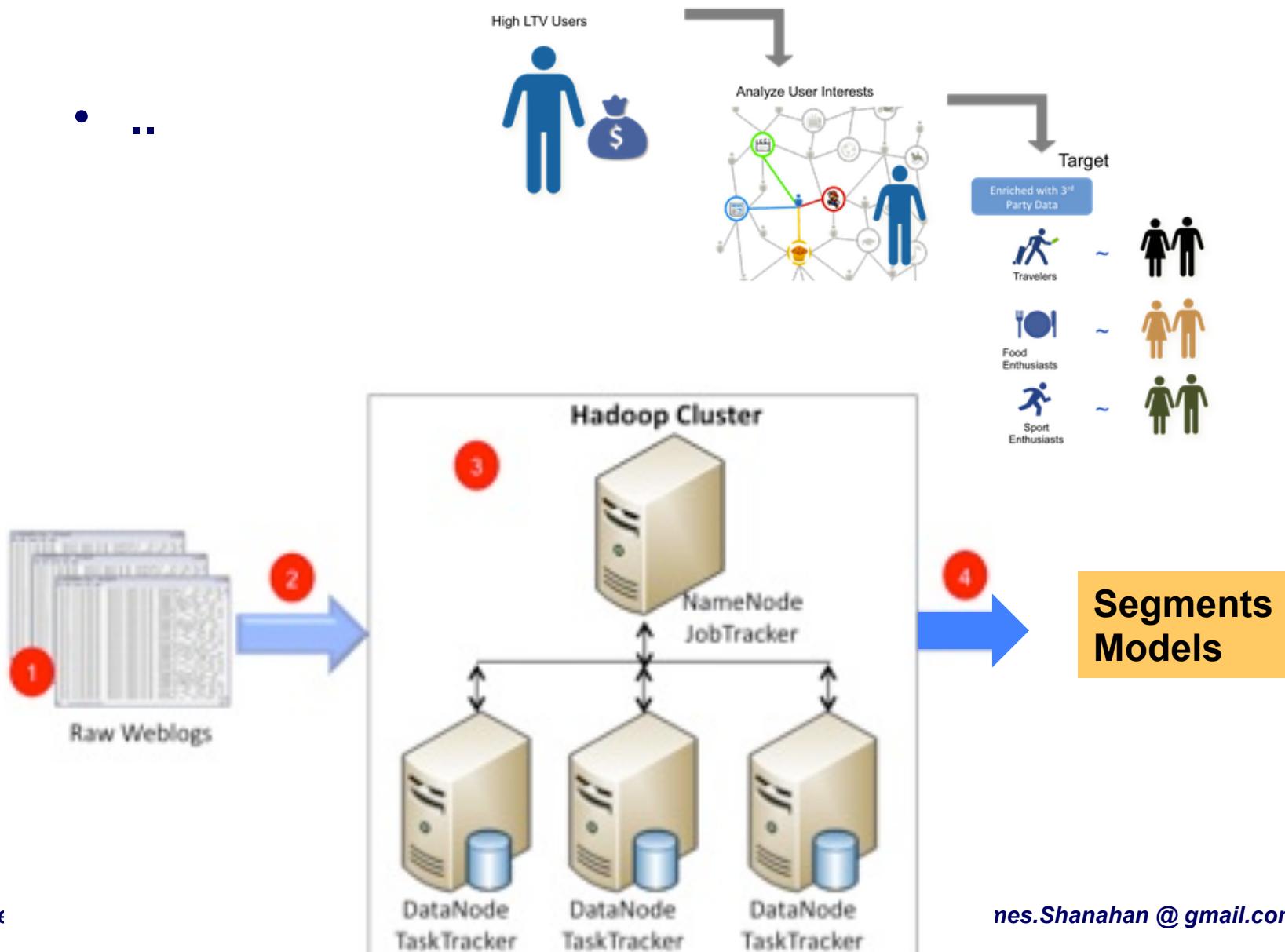
one-time
processing

Input

Distributed
memory

Spend 90% of time do I/O

MapReduce: Typical usecase



Hadoop is a black box and lacks REPL; Complex Jobs not possible

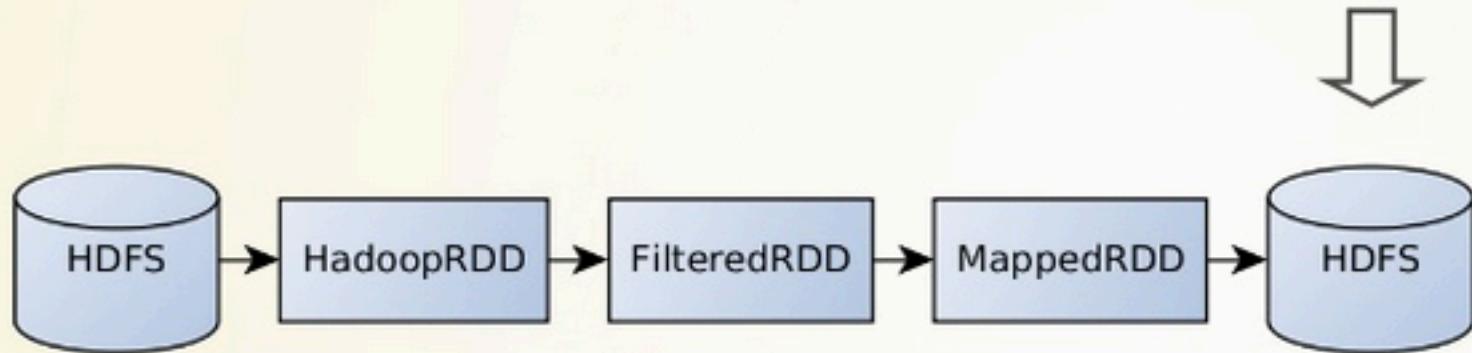
Read–eval–print loop

- Run the WordCount example job written in Java.
 - `hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar
hadoop-examples*.jar wordcount /user/hduser/
gutenberg /user/hduser/gutenberg-output`
- This command will read all the files in the HDFS directory `/user/hduser/gutenberg`, process it, and store the result in the HDFS directory `/user/hduser/gutenberg-output`.
- Hadoop is a black box
 - provide input, process, get output
 - Difficult to manipulate/access the data in the stream

Hadoop not good with pipeline jobs

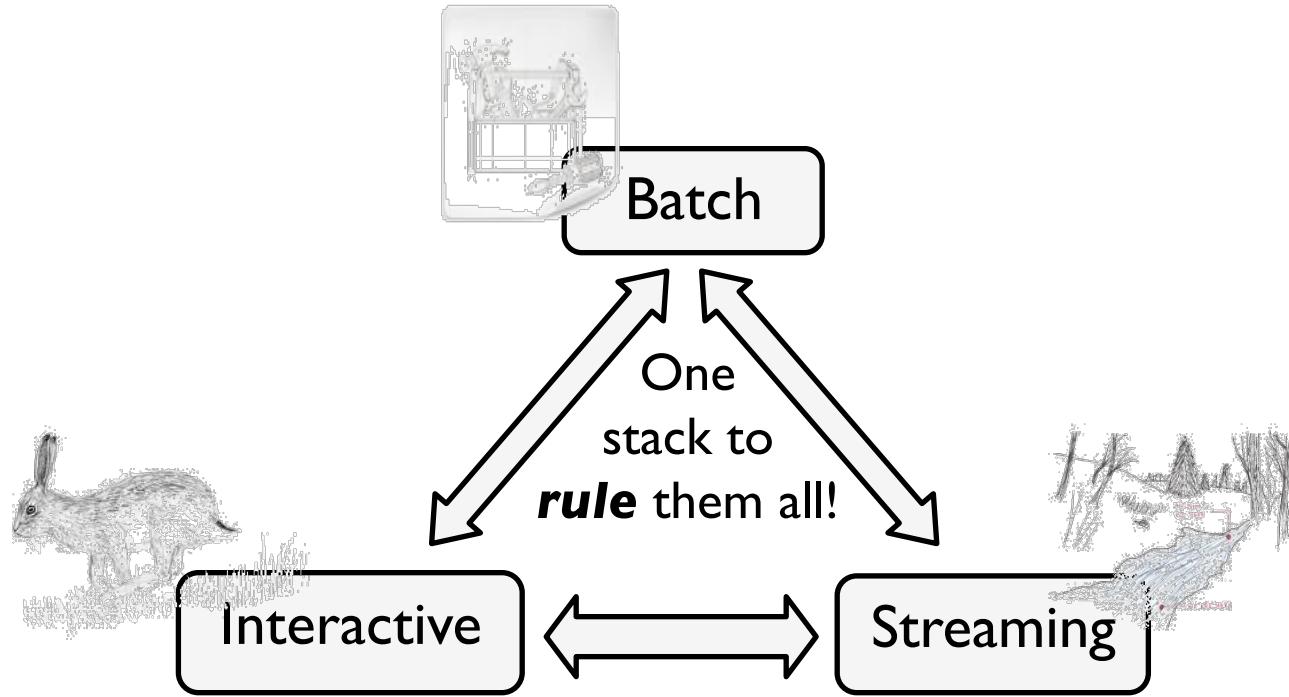
Step by Step

RDD: each row is a key, value pair



The `RDD.saveAsTextFile()` action triggers a job. Tasks are started on scheduled executors.

Goals for (Distributed) Systems for Big Data Science



- **Easy** to combine *batch*, *streaming*, and *interactive* computations
- **Easy** to develop *sophisticated* algorithms
- **Compatible** with existing open source ecosystem (Hadoop/HDFS)

Spark

In-Memory Cluster Computing for Iterative and Interactive Applications

[Zaharia, 2009, UC Berkeley]

Became an top level Apache project in 2013



What is Spark?

Fast and Expressive Cluster Computing System
Compatible with Apache Hadoop

Up to **10x** faster on disk,
100x in memory

2-5x less code

Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python, R (1.4)
- Interactive shell

Spark Metrics

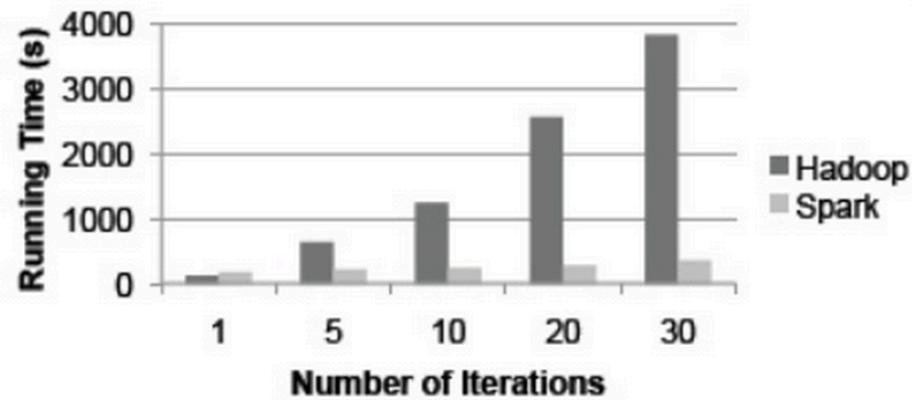
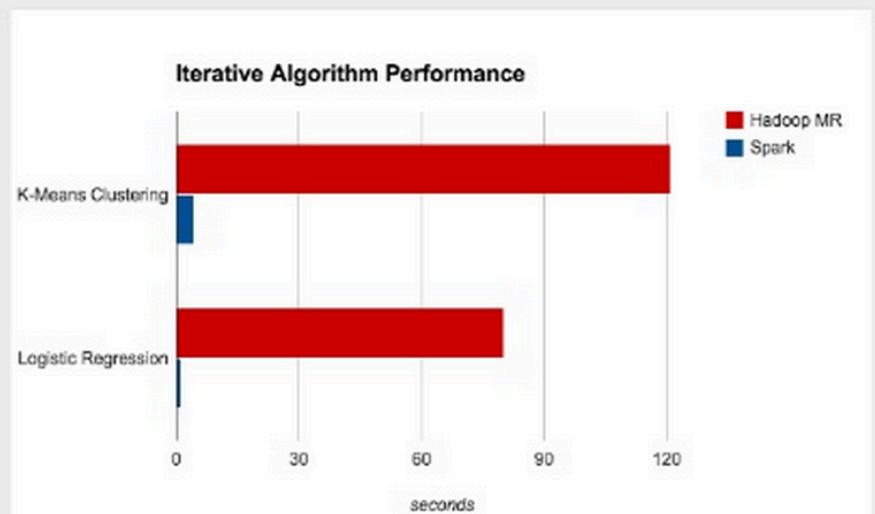


Figure 2: Logistic regression performance in Hadoop and Spark.



Code Size

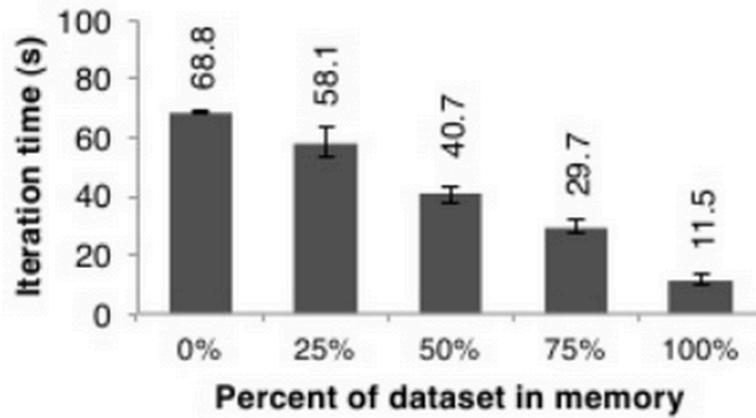
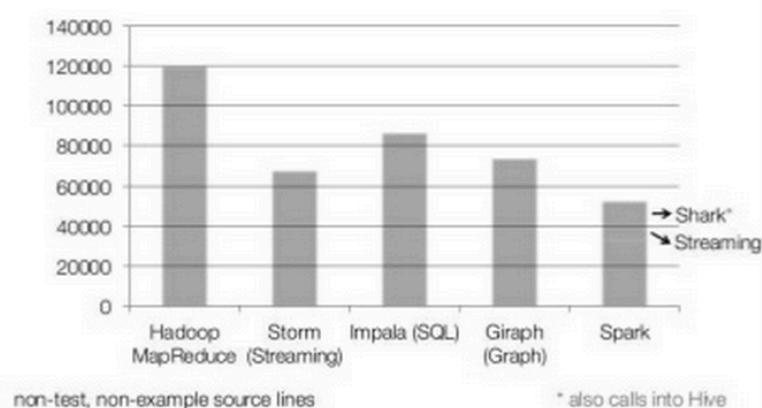
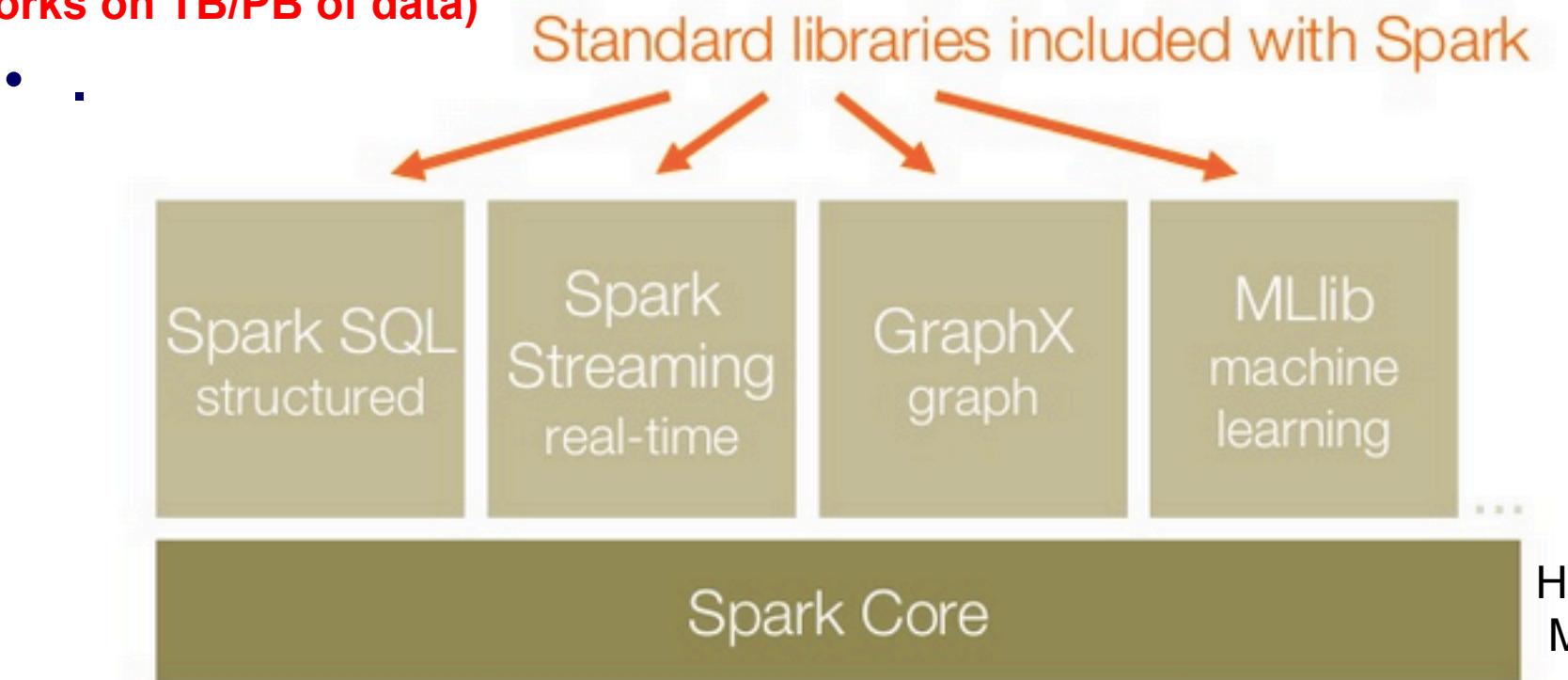


Figure 12: Performance of logistic regression using 100 GB data on 25 machines with varying amounts of data in memory.

- **Data Management**
 - Split data
 - Ship data
 - Replicate data
 - Run tasks
- **Task management**
 - Distribute, track
- **Fault tolerance (nodes crash 1-5 in a 1000 per day)**
- **First class programming framework for distributing programming over huge volumes of data that leverages memory, disk and network resources and constraints**
- **REPL (Read–eval–print loop)**

Spark

Apache Spark is an open-source cluster computing framework for big data (works on TB/PB of data)



Cluster Manager (Yarn/Mesos)

Machines (cores and storage)

Spark in local mode on a single computer or on a Spark Cluster

- Apache Spark is an open-source cluster computing framework for big data (works on TB/PB of data)

```
df= sc.textFile("hdfs://mydocs")
dfLen= df.map(length)
strLen= dfLen.reduce(sum)
dfLen.saveAsTextFile("strLens") #Action
```

INPUT	Key	Value
	d1	the quick brown fox
	d2	the fox ate the mouse
	d3	how now brown cow

OUTPUT	Key	Value
	d1	20
	d2	22
	d3	17

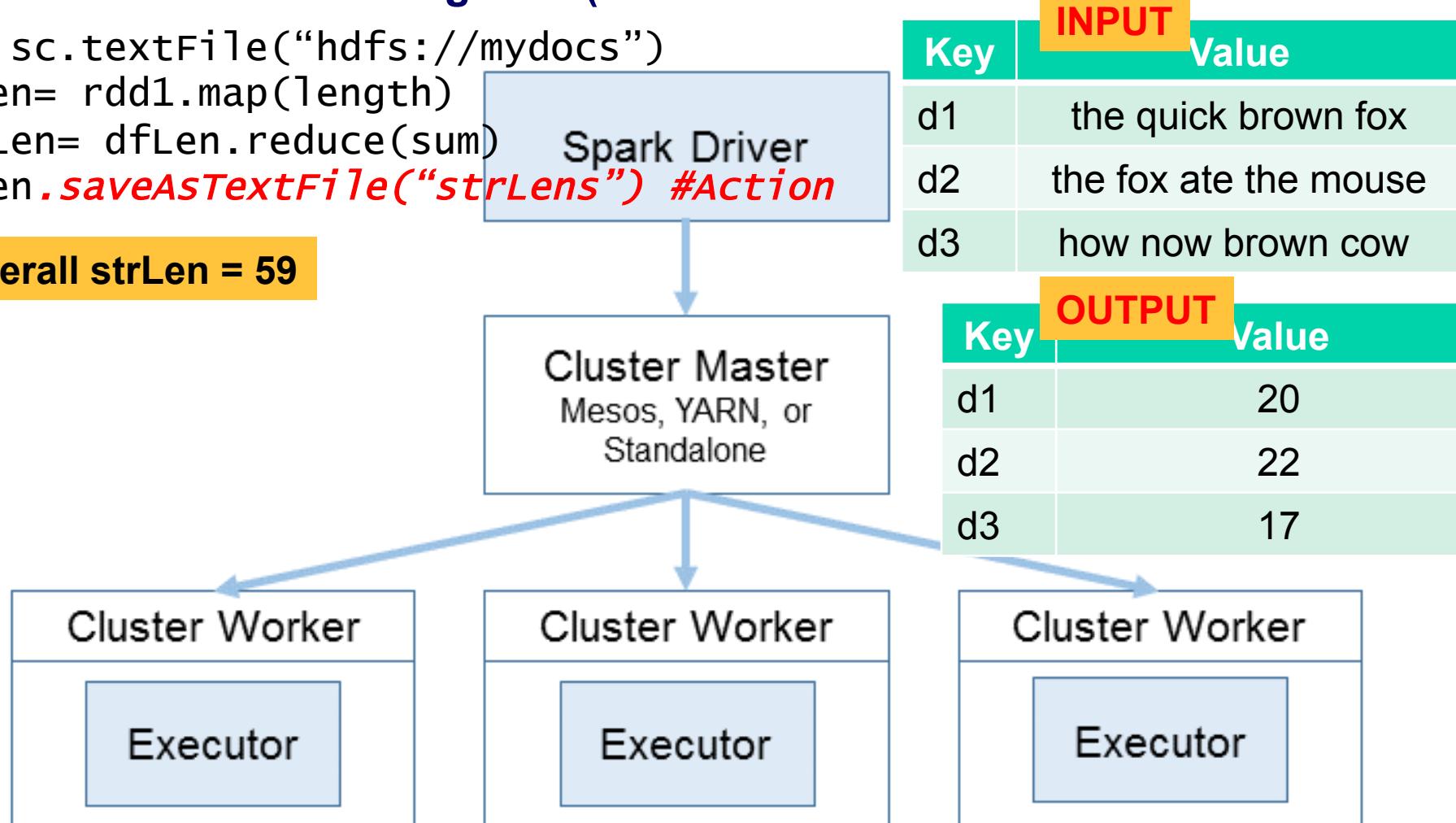
Overall strLen = 59

Spark in local mode on a single computer or on a Spark Cluster

- Apache Spark is an open-source cluster computing framework for big data (works on TB/PB of data)

```
df= sc.textFile("hdfs://mydocs")
dfLen= rdd1.map(length)
strLen= dfLen.reduce(sum)
dfLen.saveAsTextFile("strLens") #Action
```

Overall strLen = 59



Life of a Spark Program

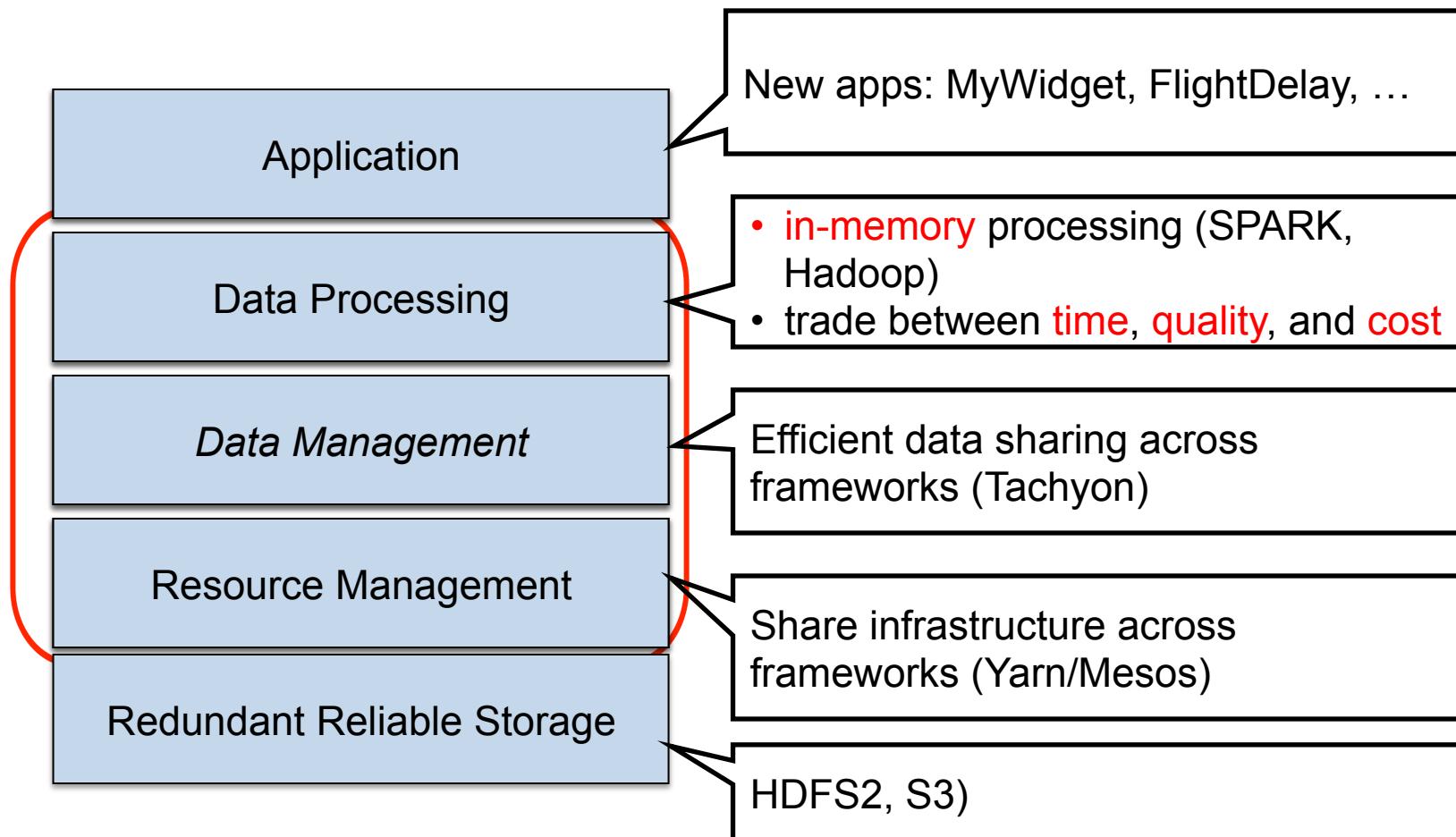
- 1) Create some input RDDs from external data or parallelize a collection in your driver program.
- 2) Lazily *transform* them to define new RDDs using transformations like `filter()` or `map()`
- 3) Ask Spark to `cache()` any intermediate RDDs that will need to be reused.
- 4) Launch *actions* such as `count()` and `collect()` to kick off a parallel computation, which is then optimized and executed by Spark.

Spark APIs

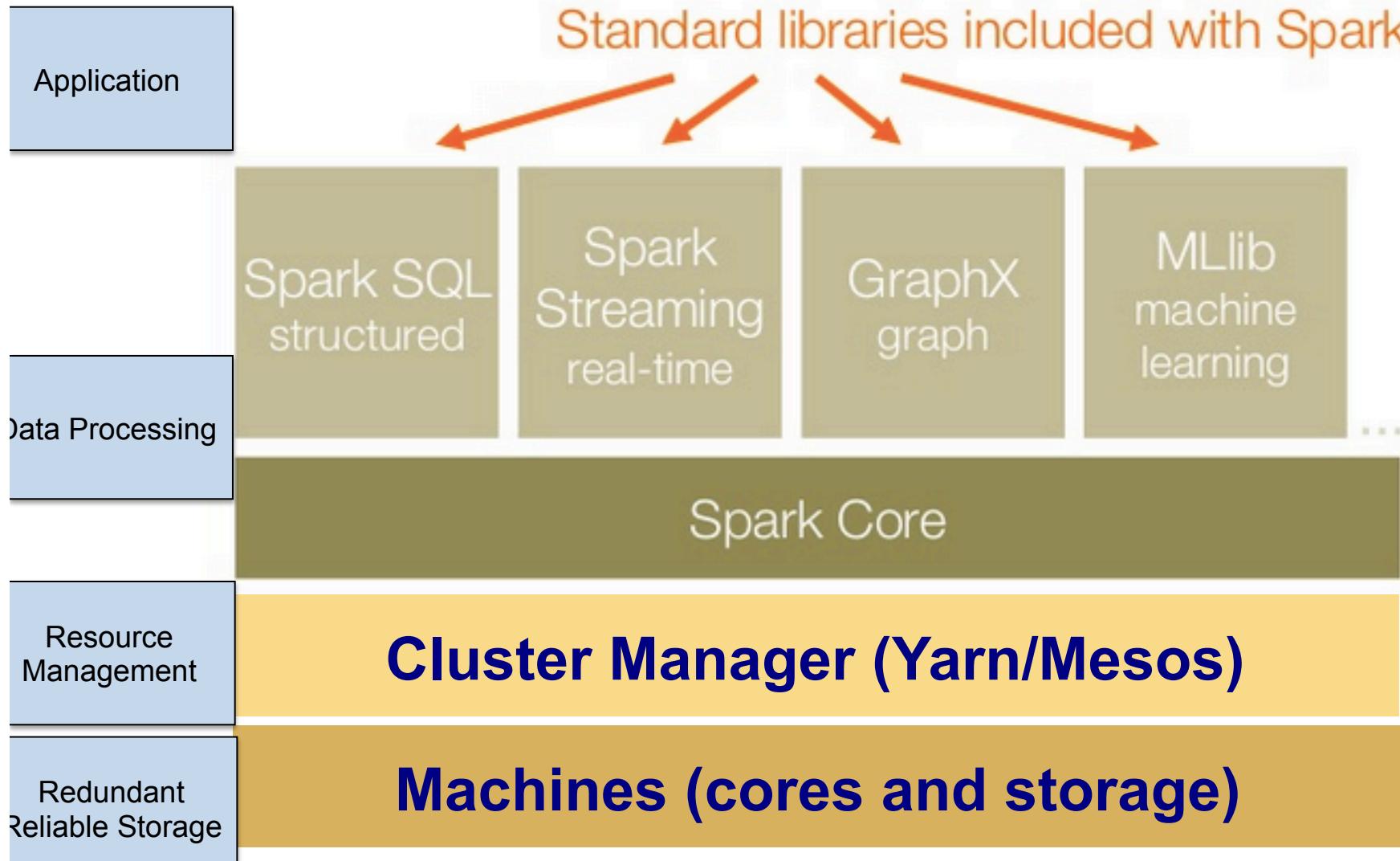
- **API**
 - In computing, a binding from a programming language to a library or operating system service is an application programming interface (API) providing glue code to use that library or service in a particular programming language.
- **Apache Spark** Apache Spark is an alternative big data computing system which can run on Yarn/Mesos and provides
 - An Elegant, Rich and Usable Core API
 - An Expansive set of ecosystem libraries built around the Core API
 - Hive compatibility via SparkSQL
 - Mature Python/R/Java/SQL/Scala support for both core APIs as well as the spark ecosystem
- **Spark has APIs for**
 - Scala, Java, Python, R, and SQL

(Ipython/Zeppelin Notebook
as a Unified Data Science
Interface to all of this)

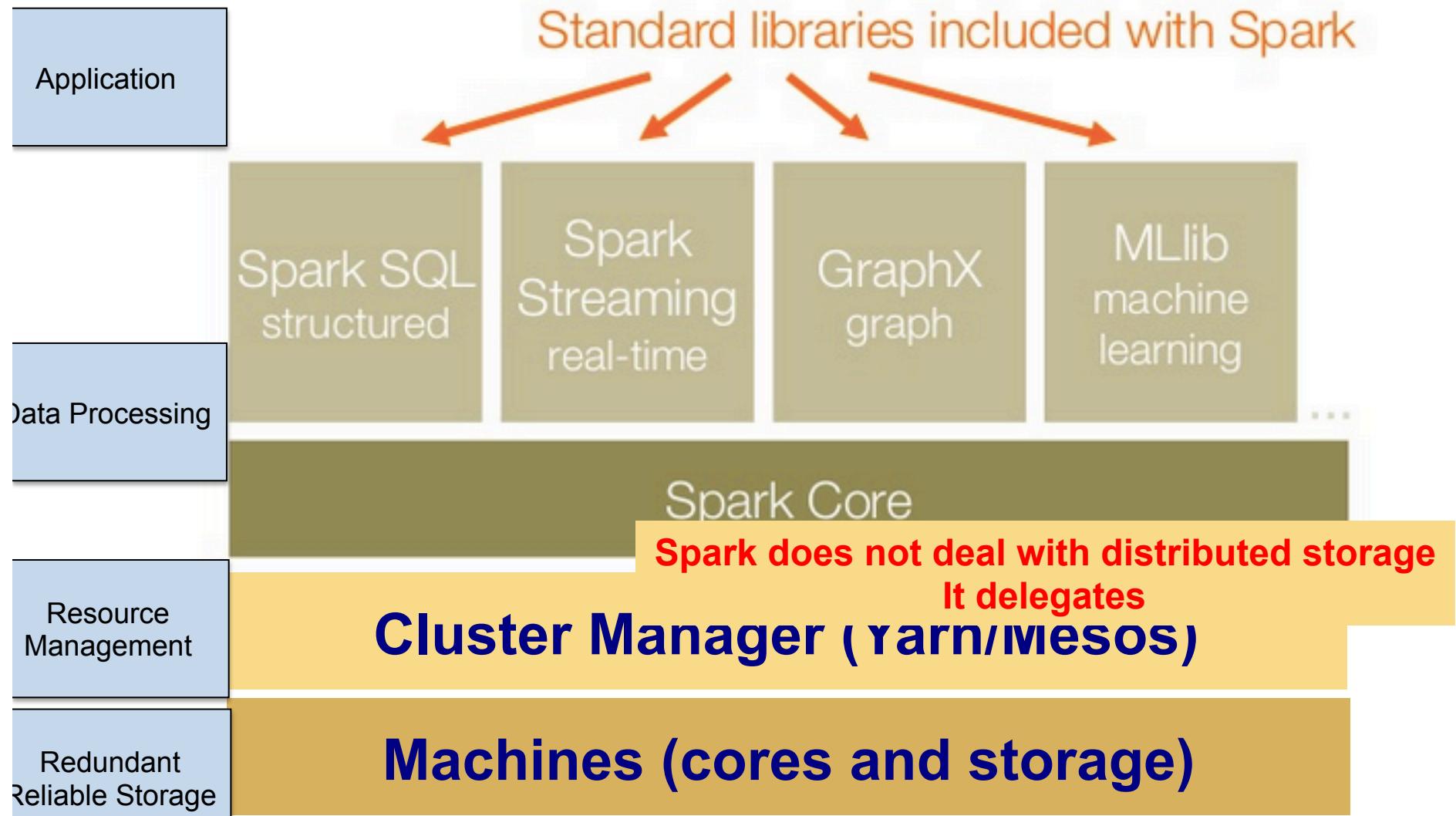
Berkeley Data Analytics Stack (BDAS)



Spark



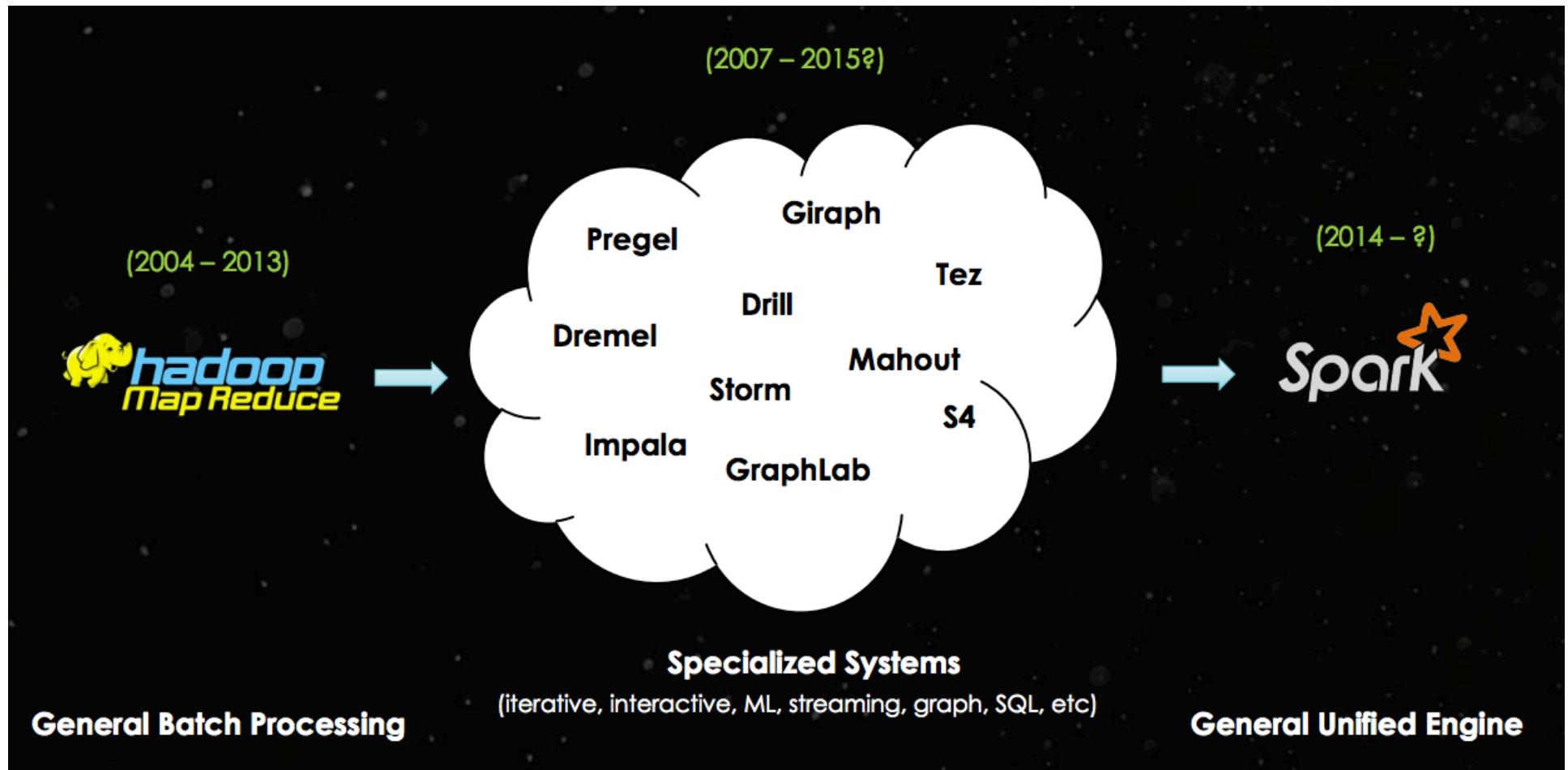
Spark



Divide and conquer with Closures

- **Decompose problems in non-overlapping sub-problems**
- **Spark's API relies heavily on passing functions in the driver program to run on the cluster. There are three recommended ways to do this:**
 - Lambda expressions, for simple functions that can be written as an expression. (Lambdas do not support multi-statement functions or statements that do not return a value.)
 - Local defs inside the function calling into Spark, for longer code.
 - Top-level functions in a module.
- **Lazy evaluation! Optimize execution graphs**

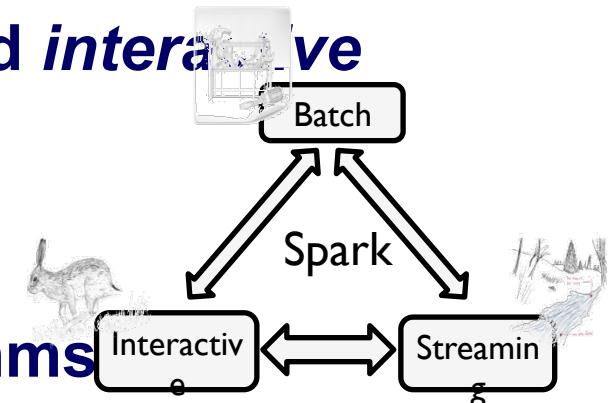
Spark builds on ...



Google, Yahoo, Facebook, Twitter, MetaMarkets, DataBricks and many more

Spark Summary

- **Support *interactive* and *streaming* computations**
 - In-memory, fault-tolerant storage abstraction, low-latency scheduling,...
- **Easy to combine *batch*, *streaming*, and *interactive* computations**
 - Spark execution engine supports all comp. models
- **Easy to develop *sophisticated* algorithms**
 - Scala interface, APIs for Java, Python, R, SQL, Hive QL, ...
 - New frameworks targeted to graph based and ML algorithms
- **Compatible with existing open source ecosystem**
- **Open source (Apache) and fully committed to release *high quality* software**



-
- End section

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Install the following:

Oracle Java JDK

Spark

Anaconda Python (+Jupyter notebooks)

Installing Hadoop

- **Windows:**
 - Hadoop 1.0
 - <http://saphanatutorial.com/hadoop-installation-on-windows-7-using-cygwin/>
 - Hadoop 2.0 (Hortonworks Data Platform 2.0 for Windows)
 - <http://hortonworks.com/blog/install-hadoop-windows-hortonworks-data-platform-2-0/>
- **Mac:**
 - <http://amodernstory.com/2014/09/23/installing-hadoop-on-mac-osx-yosemite/>
 - This link is for hadoop 2.6. I follow the instructions and easily get hadoop installed.
- **Linux (Ubuntu):**
 - http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php

On Mac install HomeBrew

- **Install HomeBrew**

- Download it from the website at <http://brew.sh/> or simply paste the script inside the terminal
- `$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

On Windows install CygWin

- **Cygwin is:**
 - a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.

Current Cygwin DLL version

The most recent version of the Cygwin DLL is [2.2.1](#). Install it by running [setup-x86.exe](#) (32-bit installation) or [setup-x86_64.exe](#) (64-bit installation).

Use the setup program to perform a [fresh install](#) or to [update](#) an existing installation.

Note that individual packages in the distribution are updated separately from the DLL so the Cygwin DLL version is not useful as a general Cygwin release number.

Make sure Java JDK is installed

- **Click here:**
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- **On windows machine set up \$JAVA_HOME**
 - Right-click the My Computer icon on your desktop and select Properties
 - Click the Advanced tab
 - Click the Environment Variables button
 - Under System Variables, click New
 - Enter the variable name as JAVA_HOME
 - Enter the variable value as the installation path for the Java Development Kit



- Java SE
- Java EE
- Java ME
- Java SE Support
- Java SE Advanced & Suite
- Java Embedded
- Java DB
- Web Tier
- Java Card
- Java TV
- New to Java
- Community
- Java Magazine

Overview

Downloads

Documentation

Community

Technologies

Training

Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- [Java Developer Newsletter](#) (tick the checkbox under Subscription Center > Oracle Technology News)
- [Java Developer Day hands-on workshops \(free\)](#) and other events
- [Java Magazine](#)

JDK 8u51 Checksum

Looking for JDK 8 on ARM?

JDK 8 for ARM downloads have moved to the [JDK 8 for ARM download page](#).

Java SE Development Kit 8u51

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

Accept License Agreement

Decline License Agreement

Product / File Description	File Size	Download
Linux x64	146.9 MB	jdk-8u51-linux-i586.rpm
Linux x64	166.95 MB	jdk-8u51-linux-i586.tar.gz
Mac OS X x64	145.19 MB	jdk-8u51-linux-x64.rpm
Mac OS X x64	165.25 MB	jdk-8u51-linux-x64.tar.gz
Mac OS X x64	222.09 MB	jdk-8u51-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.36 MB	jdk-8u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.8 MB	jdk-8u51-solaris-sparcv9.tar.gz

On Mac: double click to install

Download python+Notebook

Contents

- Anaconda Install
 - OS X Install
 - OS X Uninstall
 - Linux Install
 - Linux Uninstall
 - Windows Install
 - Windows Uninstall
 - Updating from older Anaconda versions
 - What's next?
-

OS X Install

Download the Anaconda installer and double click it.

Download the [Anaconda installer](#) and double click it.

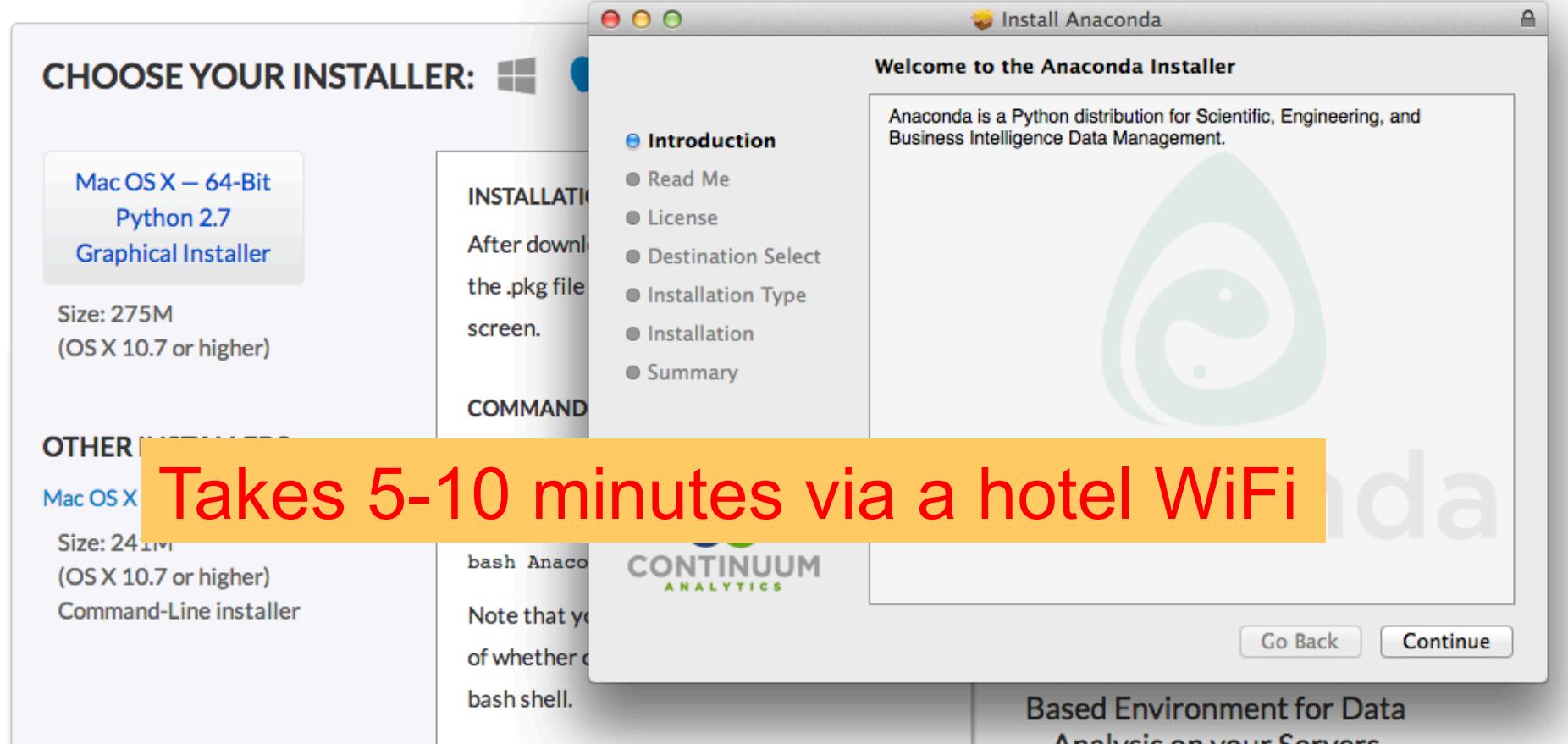
NOTE: You may see a screen that says "You cannot install Anaconda in this location. The Anaconda

Takes 5-10 minutes via a hotel WiFi



Anaconda Installer

engineering, data analysis.



```
conda update --prefix /Users/jshanahan/anaconda anaconda
```

conda update --prefix /Users/jshanahan/anaconda anaconda

```
boto-2.38.0-py 100% |#####| Time: 0:00:00 90.07 kB/s  
conda-env-2.4. 100% |#####| Time: 0:00:06 387.59 kB/s  
cython-0.22.1- 100% |#####| Time: 0:00:01 202.66 kB/s  
cytoolz-0.7.3- 100% |#####| Time: 0:00:00 1.17 MB/s  
decorator-3.4. 100% |#####| Time: 0:00:00 55.86 kB/s  
greenlet-0.4.7 100% |#####| Time: 0:00:00 145.75 kB/s  
idna-2.0-py27_ 100% |#####| Time: 0:00:00 91.65 kB/s  
ipaddress-1.0. 100% |#####| Time: 0:00:14 435.04 kB/s  
llvmlite-0.5.0 100% |#####| Time: 0:00:04 220.90 kB/s  
lxlxml-3.4.4-py2 100% |#####| Time: 0:00:01 167.80 kB/s  
mistune-0.5.1- 100% |#####| Time: 0:00:01 175.91 kB/s  
nose-1.3.7-py2 100% |#####| Time: 0:00:14 369.05 kB/s  
astropy-1.0.3- 100% |#####| Time: 0:00:01 245.33 kB/s  
bcolz-0.9.0-np 100% |#####| Time: 0:00:04 230.86 kB/s  
bottleneck-1.0 100% |#####| Time: 0:00:00 128.90 kB/s  
numba-0.19.1-n 100% |#####| Time: 0:00:01 221.19 kB/s  
numexpr-2.4.3- 100% |#####| Time: 0:00:01 165.68 kB/s  
blz-0.6.2-np19 100% |#####| Time: 0:00:00 3.96 MB/s  
pillow-2.8.2-p 100% |#####| Time: 0:00:01 141.94 kB/s  
ply-3.6-py27_0 100% |#####| Time: 0:00:01 158.43 kB/s  
py-1.4.27-py27 100% |#####| Time: 0:00:01 145.17 kB/s  
pycparser-2.14 100% |#####| Time: 0:00:00 83.32 kB/s  
cffi-1.1.0-py2 100% |#####| Time: 0:00:00 99.50 kB/s  
pycurl-7.19.5. 100% |#####| Time: 0:00:01 163.19 kB/s  
pyflakes-0.9.2 100% |#####| Time: 0:00:00 166.70 kB/s  
pytest-2.7.1-p 100% |#####| Time: 0:00:00 166.70 kB/s  
python-2.7.10- 100% |#####| Time: 0:00:28 412.67 kB/s  
python.app-1.2 100% |#####| Time: 0:00:01 166.70 kB/s  
pytz-2015.4-py 100% |#####| Time: 0:00:01 166.70 kB/s  
nvvvam1-3 11-nv 100% |#####| Time: 0:00:01 166.70 kB/s
```

To start the Notebook

/Users/jshanahan/anaconda/bin/ipython notebook&

- **/Users/jshanahan/anaconda/bin/ipython
notebook&**

Apache Spark Download

- **Install 1.4.1 (or latest version) by clicking here**
 - <https://dl.dropboxusercontent.com/u/27377155/spark-1.4.1-bin-hadoop2.6.tgz>
- **For other versions click here**
 - <http://spark.apache.org/downloads.html>
 - <http://spark.apache.org/docs/latest/programming-guide.html>

Download latest version of Spark



Download Libraries ▾ Documentation ▾ Examples Community ▾ FAQ

Get latest version Spark 1.6.2 as of June, 2016

[Download Spark](#)

The latest release of Spark is Spark 1.5.1, released on October 2, 2015 ([release notes](#)) ([git tag](#))

Step1

Choose a Spark release:

Step2

Choose a package type:

3. Choose a download type:

Step4

+ Download Spark: [spark-1.5.1-bin-hadoop2.6.tgz](#)

5. Verify this release using the [1.5.1 signatures and checksums](#).

Note: Scala 2.11 users should download the Spark source package and build [with Scala 2.11 support](#).

Untar Spark; start pyspark interpreter

```
tar -xzvf spark-1.4.1-bin-hadoop2.6.tgz  
cd spark-1.4.1-bin-hadoop2.6  
.bin/pyspark  
#if everything goes well, you can see the PySpark interactive shell  
.bin/pyspark
```

```
.....  
JAMES-SHANAHANs-Desktop-Pro:Software jshanahan$ ls spark-1.2.1-bin-hadoop2.4  
LICENSE README.md bin data examples python  
NOTICE RELEASE conf ec2 lib shin  
JAMES-SHANAHANs-Desktop-Pro:Software jshanahan$ spark-1.2.1-bin-hadoop2.4/bin/pyspark  
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
Spark assembly has been built with Hive, including Datanucleus jars on classpath  
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties  
15/02/11 16:57:02 INFO SecurityManager: Changing view acls to: jshanahan  
15/02/11 16:57:02 INFO SecurityManager: Changing modify acls to: jshanahan
```

Untar the Spark; start pyspark interpreter

- tar -xvf spark-1.2.1-bin-hadoop2.4.tgz
- #Start pyspark
- spark-1.2.1-bin-hadoop2.4/bin/pyspark

```
JAMES-SHANAHANS-Desktop-Pro:Software jshanahan$ ls spark-1.2.1-bin-hadoop2.4
LICENSE      README.md    bin        data      examples   python
NOTICE       RELEASE     conf      ec2       lib       sbin
JAMES-SHANAHANS-Desktop-Pro:Software jshanahan$ spark-1.2.1-bin-hadoop2.4/bin/pyspark
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/02/11 16:57:02 INFO SecurityManager: Changing view acls to: jshanahan
15/02/11 16:57:02 INFO SecurityManager: Changing modify acls to: jshanahan
15/02/11 16:57:02 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; user permissions: Set(jshanahan); users with modify permissions: Set(jshanahan)
15/02/11 16:57:02 INFO Slf4jLogger: Slf4jLogger started
15/02/11 16:57:02 INFO Remoting: Starting remoting
15/02/11 16:57:02 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@10.0.0.93:54079]
15/02/11 16:57:02 INFO Utils: Successfully started service 'sparkDriver' on port 54079.
15/02/11 16:57:02 INFO SparkEnv: Registering MapOutputTracker
15/02/11 16:57:02 INFO SparkEnv: Registering BlockManagerMaster
15/02/11 16:57:02 INFO DiskBlockManager: Created local directory at /var/folders/j4/95k348x940xcz40fk/T/spark-5081f827-8087-436d-8cc2-bb05a24410a3/spark-c4be330b-173d-4f3f-8659-2a919c2f7bab
15/02/11 16:57:02 INFO MemoryStore: MemoryStore started with capacity 273.0 MB
2015-02-11 16:57:03.065 java[44783:ea03] Unable to load realm info from SCDynamicStore
15/02/11 16:57:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using avro classes where applicable
15/02/11 16:57:03 INFO HttpFileServer: HTTP File server directory is /var/folders/j4/95k348x940xcz40fn/T/spark-d6caeef39-c3bc-48d3-8a9e-f4b24e3e4a21/spark-1c7165c4-a18d-4f23-86ae-5d92519ff345
15/02/11 16:57:03 INFO HttpServer: Starting HTTP Server
15/02/11 16:57:03 INFO Utils: Successfully started service 'HTTP file server' on port 54080.
15/02/11 16:57:03 INFO Utils: Successfully started service 'SparkUI' on port 4040.
15/02/11 16:57:03 INFO SparkUI: Started SparkUI at http://10.0.0.93:4040
15/02/11 16:57:03 INFO Executor: Starting executor ID <driver> on host localhost
15/02/11 16:57:03 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@10.0.0.93:54079
15/02/11 16:57:03 INFO NettyBlockTransferService: Server created on 54081
15/02/11 16:57:03 INFO BlockManagerMaster: Trying to register BlockManager
15/02/11 16:57:03 INFO BlockManagerMasterActor: Registering block manager localhost:54081 with 273.0 MB
15/02/11 16:57:03 INFO BlockManagerMaster: Registered BlockManager
Welcome to
   _/\_   _/\_   _/\_   _/\_
  / \ \ / \ \ / \ \ / \ \
 /   \ /   \ /   \ /   \ /   \
 /     \ /     \ /     \ /     \
 /       \ /       \ /       \ /       \
 /         \ /         \ /         \ /         \
 /           \ /           \ /           \ /           \
 /             \ /             \ /             \ /             \
 /               \ /               \ /               \ /               \
 /                 \ /                 \ /                 \ /                 \
 /                   \ /                   \ /                   \ /                   \
 /                     \ /                     \ /                     \ /                     \
 /                       \ /                       \ /                       \ /                       \
 /                         \ /                         \ /                         \ /                         \
 /                           \ /                           \ /                           \ /                           \
 /                             \ /                             \ /                             \ /                             \
 /                               \ /                               \ /                               \ /                               \
 /                                 \ /                                 \ /                                 \ /                                 \
 /                                   \ /                                   \ /                                   \ /                                   \
 /                                     \ /                                     \ /                                     \ /                                     \
 /                                       \ /                                       \ /                                       \ /                                       \
 /                                         \ /                                         \ /                                         \ /                                         \
 /                                           \ /                                           \ /                                           \ /                                           \
 /                                             \ /                                             \ /                                             \ /                                             \
 /                                               \ /                                               \ /                                               \ /                                               \
 /                                                 \ /                                                 \ /                                                 \ /                                                 \
 /                                                   \ /                                                   \ /                                                   \ /                                                   \
 /                                                     \ /                                                     \ /                                                     \ /                                                     \
 /                                                       \ /                                                       \ /                                                       \ /                                                       \
 /                                                         \ /                                                         \ /                                                         \ /                                                         \
 /                                                           \ /                                                           \ /                                                           \ /                                                           \
 /                                                             \ /                                                             \ /                                                             \ /                                                             \
 /                                                               \ /                                                               \ /                                                               \ /                                                               \
 /                                                                 \ /                                                                 \ /                                                                 \ /                                                                 \ /
```

Using Python version 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012 20:52:43)
SparkContext available as sc.
>>> █

```
.  
+-- bin  
|   +-- beeline  
|   |   beeline.cmd  
|   |   compute-classpath.cmd  
|   |   compute-classpath.sh  
|   |   load-spark-env.sh  
|   +-- pyspark #python  
|       pyspark2.cmd  
|       pyspark.cmd  
|       run-example  
|       run-example2.cmd  
|       run-example.cmd  
|       spark-class  
|       spark-class2.cmd  
|       spark-class.cmd  
|       spark-shell #SCALA  
|       spark-shell2.cmd  
|       spark-shell.cmd  
|       spark-sql  
|       spark-submit  
|       spark-submit2.cmd  
|       spark-submit.cmd  
|       utils.sh  
|       windows-utils.cmd  
+-- CHANGES.txt  
+-- conf  
|   +-- fairscheduler.xml.template  
|   +-- log4j.properties.template  
|   +-- metrics.properties.template  
|   +-- slaves.template  
|   +-- spark-defaults.conf.template  
|   +-- spark-env.sh.template  
+-- data  
    +-- mllib
```

```
+-- ec2  
|   +-- deploy.generic  
|   +-- README  
|   +-- spark-ec2  
|       spark_ec2.py  
+-- examples  
|   +-- src  
+-- lib  
|   +-- datanucleus-api-jdo-3.2.6.jar  
|   +-- datanucleus-core-3.2.10.jar  
|   +-- datanucleus-rdbms-3.2.9.jar  
|   +-- spark-1.3.1-yarn-shuffle.jar  
|   +-- spark-assembly-1.3.1-hadoop2.6.0.jar  
|   +-- spark-examples-1.3.1-hadoop2.6.0.jar  
+-- LICENSE  
+-- NOTICE  
+-- python  
|   +-- build  
|   +-- docs  
|   +-- lib  
|       +-- pyspark  
|           +-- run-tests  
|               test_support  
+-- README.md  
+-- RELEASE  
+-- sbin  
|   +-- slaves.sh  
|   +-- spark-config.sh  
|   +-- spark-daemon.sh  
|   +-- spark-daemons.sh  
|   +-- start-all.sh  
|   +-- start-history-server.sh  
|   +-- start-master.sh  
|   +-- start-slave.sh  
|   +-- start-slaves.sh  
|   +-- start-thriftserver.sh  
|   +-- stop-all.sh  
|   +-- stop-history-server.sh  
|   +-- stop-master.sh  
|   +-- stop-slaves.sh  
|   +-- stop-thriftserver.sh
```

```
holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1$ ./bin/pyspark
holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1$ ./bin/pyspark
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
14/11/19 14:38:03 WARN Utils: Your hostname, hmbp2 resolves to a loopback address: 127.0.1.1; using 172.17.42.1 instead (on interface docker0)
14/11/19 14:38:03 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Welcome to
    _____
   / \   \
  /  \  /
 /  _ \ / \
/  / \ \ \_ \
\  \_ \ \_ \
   \ \_ \_ \
    \ \_ \_ \
     \ \_ \_ \
      \ \_ \_ \
       \ \_ \_ \
        \ \_ \_ \
         \ \_ \_ \
          \ \_ \_ \
           \ \_ \_ \
            \ \_ \_ \
             \ \_ \_ \
              \ \_ \_ \
               \ \_ \_ \
                \ \_ \_ \
                 \ \_ \_ \
                  \ \_ \_ \
                   \ \_ \_ \
                    \ \_ \_ \
                     \ \_ \_ \
                      \ \_ \_ \
                       \ \_ \_ \
                        \ \_ \_ \
                         \ \_ \_ \
                          \ \_ \_ \
                           \ \_ \_ \
                            \ \_ \_ \
                             \ \_ \_ \
                              \ \_ \_ \
                               \ \_ \_ \
                                \ \_ \_ \
                                 \ \_ \_ \
                                  \ \_ \_ \
                                   \ \_ \_ \
                                    \ \_ \_ \
                                     \ \_ \_ \
                                      \ \_ \_ \
                                       \ \_ \_ \
                                        \ \_ \_ \
                                         \ \_ \_ \
                                          \ \_ \_ \
                                           \ \_ \_ \
                                            \ \_ \_ \
                                             \ \_ \_ \
                                              \ \_ \_ \
                                               \ \_ \_ \
                                                \ \_ \_ \
                                                 \ \_ \_ \
                                                  \ \_ \_ \
                                                   \ \_ \_ \
                                                    \ \_ \_ \
                                                     \ \_ \_ \
                                                      \ \_ \_ \
                                                       \ \_ \_ \
                                                        \ \_ \_ \
                                                         \ \_ \_ \
                                                          \ \_ \_ \
                                                           \ \_ \_ \
                                                            \ \_ \_ \
                                                             \ \_ \_ \
                                                              \ \_ \_ \
                                                               \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
                                                                 \ \_ \_ \
                                                                \ \_ \_ \
................................................................
```

Where did you install Spark and untar it?

/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6

cd /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6

bin/pyspark #python

Figure 2-2. The PySpark shell with less logging output

bin/sparkR

```
JAMES-SANAHANS-Desktop-Pro:KDD-Notebooks jshanahan$ cd /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6  
JAMES-SANAHANS-Desktop-Pro:spark-1.4.0-bin-hadoop2.6 jshanahan$ bin/sparkR
```

R
Co
Pl
R
Yo
Ty
I
R
Ty
'c
Ty
'h
Ty

Where did you install Spark and untar it?

**/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/
spark-1.5.0-bin-hadoop2.6**

#To run sparkR

**/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/
spark-1.5.0-bin-hadoop2.6/bin/sparkR**

```
Launching java with spark-submit command /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6/bin/spark-submit "sparkr-shell" /var/folders/j4/95k348x940xcz40fkdmgy_n40000gn/T//RtmpjQpRRC/backend_port14725284a4ad  
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).  
log4j:WARN Please initialize the log4j system properly.  
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.  
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties  
15/08/10 13:52:17 INFO SparkContext: Running Spark version 1.4.0  
15/08/10 13:52:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
15/08/10 13:52:19 INFO SecurityManager: Changing view acls to: jshanahan  
15/08/10 13:52:19 INFO SecurityManager: Changing modify acls to: jshanahan  
15/08/10 13:52:19 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(jshanahan); users with modify permissions: Set(jshanahan)
```

Commands in R: iris data

iris

package:datasets

R Documentation

[Edgar Anderson's Iris Data](#)

Description:

- **?iris**

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.

Usage:

- **?data**

iris
iris3

Format:

'iris' is a data frame with 150 cases (rows) and 5 variables (columns) named 'Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', and 'Species'.

'iris3' gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names 'Sepal L.', 'Sepal W.', 'Petal L.', and 'Petal W.', and the third the species.

> head(iris)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Data sets in package 'datasets' :

AirPassengers	Monthly Airline Passenger Numbers 1949–1960
BJSales	Sales Data with Leading Indicator
BJSales.lead (BJSales)	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
C02	Carbon Dioxide Uptake in Grass Plants
ChickWeight	Weight versus age of chicks on different diets
DNase	Elisa assay of DNase
EuStockMarkets	Daily Closing Prices of Major European Stock Indices, 1991–1998
Formaldehyde	Determination of Formaldehyde
HairEyeColor	Hair and Eye Color of Statistics Students
Harman23.cor	Harman Example 2.3
Harman74.cor	Harman Example 7.4
Indometh	Pharmacokinetics of Indomethacin
InsectSprays	Effectiveness of Insect Sprays
JohnsonJohnson	Quarterly Earnings per Johnson & Johnson Share
LakeHuron	Level of Lake Huron 1875–1972
LifeCycleSavings	Intercountry Life-Cycle Savings Data
Loblolly	Growth of Loblolly pine trees
Nile	Flow of the River Nile
Orange	Growth of Orange Trees
OrchardSprays	Potency of Orchard Sprays
PlantGrowth	Results from an Experiment on Plant Growth
Puromycin	Reaction Velocity of an Enzymatic Reaction
Seatbelts	Road Casualties in Great Britain 1969–84
Theoph	Pharmacokinetics of Theophylline
Titanic	Survival of passengers on the Titanic
ToothGrowth	The Effect of Vitamin C on Tooth Growth in Guinea Pigs
UCBAdmissions	Student Admissions at UC Berkeley
UKDriverDeaths	Road Casualties in Great Britain 1969–84
UKgas	UK Quarterly Gas Consumption
USAccDeaths	Accidental Deaths in the US 1973–1978
USArests	Violent Crime Rates by US State
USJudgeRatings	Lawyers' Ratings of State Judges in the US Superior Court
USPersonalExpenditure	Personal Expenditure Data
VADeaths	Death Rates in Virginia (1940)
WWWusage	Internet Usage per Minute
WorldPhones	The World's Telephones
ability.cov	Ability and Intelligence Tests
airmiles	Passenger Miles on Commercial US Airlines, 1937–1960

>data() #lists all datasets

airquality	New York Air Quality Measurements	morley	Michelson Speed of Light Data
anscombe	Anscombe's Quartet of 'Identical' Regressions	smtcars	Motor Trend Car Road Tests
attenu	The Joyner-Boore Attenuation Data	nhtemp	Average Yearly Temperatures in New Haven
attitude	The Chatterjee-Price Attitude Data	nottem	Average Monthly Temperatures at Nottingham, 1920-1939
austres	Quarterly Time Series of the Number of Australian Residents	npk	Classical N, P, K Factorial Experiment
beaver1 (beavers)	Body Temperature Series of Two Beavers	occupationalStatus	Occupational Status of Fathers and their Sons
beaver2 (beavers)	Body Temperature Series of Two Beavers	precip	Annual Precipitation in US Cities
cars	Speed and Stopping Distances of Cars	presidents	Quarterly Approval Ratings of US Presidents
chickwts	Chicken Weights by Feed Type	pressure	Vapor Pressure of Mercury as a Function of Temperature
co2	Mauna Loa Atmospheric CO2 Concentration	quakes	Locations of Earthquakes off Fiji
crimtab	Student's 3000 Criminals Data	randu	Random Numbers from Congruential Generator
discoveries	Yearly Numbers of Important Discoveries	rivers	RANDU
esoph	Smoking, Alcohol and Esophageal Cancer	rock	Lengths of Major North American Rivers
euro	Conversion Rates of Euro Currencies	sleep	Measurements on Petroleum Rock Samples
euro.cross (euro)	Conversion Rates of Euro Currencies	stack.loss	Student's Sleep Data
eurodist	Distances Between European Cities	(stackloss)	
faithful	Old Faithful Geyser Data	stack.x	Brownlee's Stack Loss Plant Data
fdeaths (UKLungDeaths)		(stackloss)	Brownlee's Stack Loss Plant Data
freeny	Monthly Deaths from Lung Diseases	state.abb	Brownlee's Stack Loss Plant Data
freeny.x (freeny)	Freeny's Revenue Data	(state)	US State Facts and Figures
freeny.y (freeny)	Freeny's Revenue Data	state.area	US State Facts and Figures
infert	Freeny's Revenue Data	state.center	US State Facts and Figures
iris	Infertility after Spontaneous Abortion	state.division	US State Facts and Figures
iris3	Edgar Anderson's Iris Data	(state)	US State Facts and Figures
islands	Edgar Anderson's Iris Data	state.region	US State Facts and Figures
ldeaths (UKLungDeaths)	Areas of the World's Major Landmasses	state.x77	US State Facts and Figures
lh		sunspot.month	Monthly Sunspot Data, from 1749 to "Present"
longley	Monthly Deaths from Lung Diseases	sunspot.year	Yearly Sunspot Data, 1700-1988
lynx	Luteinizing Hormone in Blood Samples	sunspots	Monthly Sunspot Numbers, 1749-1983
mdeaths (UKLungDeaths)	Longley's Economic Regression Data	swiss	Swiss Fertility and Socioeconomic Indicators (1888) Data
morley	Annual Canadian Lynx trappings	treering	Yearly Treering Data, -6000-1979
mtcars	Monthly Deaths from Lung Diseases	trees	Girth, Height and Volume for Black Cherry Trees
nhtemp	Michelson Speed of Light Data	uspop	Populations Recorded by the US Census
nottem	Motor Trend Car Road Tests	volcano	Topographic Information on Auckland's Maunga Whau Volcano
npk	Average Yearly Temperatures in New Zealand, 1920-1939	warpbreaks	The Number of Breaks in Yarn during Weaving
occupationalStatus	Average Monthly Temperatures at Nottingham, 1920-1939	women	Average Heights and Weights for American Women
precip	Classical N, P, K Factorial Experiment		
presidents	Occupational Status of Fathers and Sons		
pressure	Annual Precipitation in US Cities		Use 'data(package = .packages(all.available = TRUE))' to list the data sets in all *available* packages.
:	Quarterly Approval Ratings of US Presidents		
	Vapor Pressure of Mercury as a Function of Temperature		

(END)

Iris data check

- `df = createDataFrame(sqlContext, iris)`
- `head(df)`

```
> df
DataFrame[Sepal_Length:double, Sepal_Width:double, Petal_Length:double, Petal_Width:double, Species:string]
> head(df)
15/08/10 13:52:51 INFO SparkContext: Starting job: dfToCols at NativeMethodAccessorImpl.java:-2
15/08/10 13:52:51 INFO DAGScheduler: Got job 1 (dfToCols at NativeMethodAccessorImpl.java:-2) with 1 output partitions (allowLoc al=false)
15/08/10 13:52:51 INFO DAGScheduler: Final stage: ResultStage 1(dfToCols at NativeMethodAccessorImpl.java:-2)
15/08/10 13:52:51 INFO DAGScheduler: Parents of final stage: List()
15/08/10 13:52:51 INFO DAGScheduler: Missing parents: List()
15/08/10 13:52:51 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[4] at dfToCols at NativeMethodAccessorImpl.java: -2), which has no missing parents
15/08/10 13:52:51 INFO MemoryStore: ensureFreeSpace(8776) called with curMem=2134, maxMem=278019440
15/08/10 13:52:51 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 8.6 KB, free 265.1 MB)
15/08/10 13:52:51 INFO MemoryStore: ensureFreeSpace(3621) called with curMem=10910, maxMem=278019440
15/08/10 13:52:51 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 3.5 KB, free 265.1 MB)
15/08/10 13:52:51 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on localhost:54179 (size: 3.5 KB, free: 265.1 MB)
15/08/10 13:52:51 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:874
15/08/10 13:52:51 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[4] at dfToCols at NativeMet hodAccessorImpl.java:-2)
15/08/10 13:52:51 INFO TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
15/08/10 13:52:51 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, localhost, PROCESS_LOCAL, 15837 bytes)
15/08/10 13:52:51 INFO Executor: Running task 0.0 in stage 1.0 (TID 1)
15/08/10 13:52:52 INFO Executor: Finished task 0.0 in stage 1.0 (TID 1). 2502 bytes result sent to driver
15/08/10 13:52:52 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 637 ms on localhost (1/1)
15/08/10 13:52:52 INFO DAGScheduler: ResultStage 1 (dfToCols at NativeMethodAccessorImpl.java:-2) finished in 0.637 s
15/08/10 13:52:52 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
15/08/10 13:52:52 INFO DAGScheduler: Job 1 finished: dfToCols at NativeMethodAccessorImpl.java:-2, took 0.654923 s
  Sepal_Length Sepal_Width Petal_Length Petal_Width Species
1          5.1        3.5       1.4       0.2   setosa
2          4.9        3.0       1.4       0.2   setosa
3          4.7        3.2       1.3       0.2   setosa
4          4.6        3.1       1.5       0.2   setosa
```

Wordcount Notebook

- **Download the following notebook**
 - [https://www.dropbox.com/s/ul0l3q98w54dr8x/
WordCount.ipynb?dl=0](https://www.dropbox.com/s/ul0l3q98w54dr8x/WordCount.ipynb?dl=0)
- **Cd to the directory that contains the wordcount notebook**
 - cd /Users/jshanahan/Dropbox/NativeX-Internal/Publications/
WSDM-2016
- **Launch Notebook**
 - /Users/jshanahan/anaconda/bin/ipython notebook&

localhost:8888/notebooks/Notebooks/WordCount/WordCount.ipynb#

MIDS-MLS-2015 nbviewer.ipython.org Stanford Machine L Getting Started Statistical Analysis H eBay/Google 2013: E Inquiries InferPatents WindAlert - Coyote F SamCam www.3rdavekite.com Kiting james@yottapartners Import

jupyter WordCount (autosaved)

File Edit View Insert Cell Kernel Help Python 2

In [2]:

```
import os
import sys
#spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6'

if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.5.0

Using Python version 2.7.11 (default, Dec 6 2015 18:57:58)
SparkContext available as sc, HiveContext available as sqlContext.

In [3]:

```
%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Writing wordcount.txt

In [4]:

```
cat wordcount.txt
```

hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello

In []:

In [3]:

```
#Count words in README.md
logFileName = 'wordcount.txt'
text_file = sc.textFile(logFileName)
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
for v in counts.collect():
    print v
```

(u'ciao', 1)
(u'bonjour', 1)

The screenshot shows a Mac OS X desktop environment. A browser window is open at `localhost:8888/tree`. The title bar of the browser window says "IP[y]: Notebook". The browser's menu bar includes "File", "Edit", "View", "Search", "Help", and "SamCam". The toolbar below the menu bar has icons for "Apps", "Getting Started", "Statistical Analysis H", "eBay/Google 2013:", "Inquiries", "InferPatents", "WindAlert - Coyote F", and "SamCam". Below the toolbar, there are three tabs: "Notebooks" (selected), "Running", and "Clusters". A message in the center of the page says "To import a notebook, drag the file onto the listing below or [click here](#)". To the right of this message are "New Notebook" and "Import" buttons. The main content area lists five notebooks: "LogisticRegression.ipynb", "SPARK-Lecture1.ipynb", "SparkSQL.ipynb", "SummaryStatisticsExample.ipynb", and "WordCount.ipynb", each with a "Delete" button to its right. The browser window has a standard OS X title bar with red, yellow, and green buttons.

```
mkdir SparkTutorial
cd SparkTutorial
#start the python server and notebook in your browser
# from the shell/terminal command line
$ ipython notebook &
```

A terminal window titled "Notebooks — python — 80x24" is shown. The terminal output is as follows:

```
james-shanahan@JAMES-SHANAHANS-Desktop:~/Dropbox/Lectures-UC-Berkeley-ML-Class-2015>Notebooks$ ipython notebook
2015-02-11 17:51:52.879 [NotebookApp] Using existing profile directory /Users/jshanahan/.ipython/profile_default'
2015-02-11 17:51:52.879 [NotebookApp] Using MathJax from CDN: https://cdn.mathjax.org/mathjax/latest/MathJax.js
2015-02-11 17:51:52.897 [NotebookApp] Serving notebooks from local directory: /Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015>Notebooks
2015-02-11 17:51:52.897 [NotebookApp] 0 active kernels
2015-02-11 17:51:52.898 [NotebookApp] The IPython Notebook is running at: http://localhost:8888/
2015-02-11 17:51:52.898 [NotebookApp] Use Control-C to stop this server and shut
```

Large S

Modify path for SPARK_HOME

S localhost:8888/notebooks/KDD-Notebooks/WordCount/WordCount.ipynb
Bookmarks Stanford Machine Le Getting Started Statistical Analysis H eBay/Google 2013: E Inquiries InferPatents WindAlert - Coyote P SamCam

jupyter WordCount (autosaved) Python 2

In [1]:

```
import os
import sys
spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-ha
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.4.0

Using Python version 2.7.9 (default, Dec 15 2014 10:37:34)
SparkContext available as sc, HiveContext available as sqlContext.

In [2]:

```
%%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Writing wordcount.txt

To execute the cell
Press
Shift + Return

WordCount example test

- [https://www.dropbox.com/s/uI0I3q98w54dr8x/
WordCount.ipynb?dl=0](https://www.dropbox.com/s/uI0I3q98w54dr8x/WordCount.ipynb?dl=0)
- **Complete during the break**

KDD-Notebooks — bash — 120x23

```
15/08/09 21:22:33 INFO SparkEnv: Registering MapOutputTracker
15/08/09 21:22:33 INFO SparkEnv: Registering BlockManagerMaster
15/08/09 21:22:33 INFO DiskBlockManager: Created local directory at /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16baf4-b7e9-4c7e-a33d-15a6d8198406
15/08/09 21:22:33 INFO MemoryStore: MemoryStore started with capacity 265.1 MB
15/08/09 21:22:33 INFO HttpFileServer: HTTP File server directory is /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/httpd-a18d50aa-ea36-4339-9c15-604330e30548
15/08/09 21:22:33 INFO HttpServer: Starting HTTP Server
15/08/09 21:22:33 INFO Utils: Successfully started service 'HTTP file server' on port 52389.
15/08/09 21:22:33 INFO SparkEnv: Registering OutputCommitCoordinator
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
15/08/09 21:22:34 INFO Utils: Successfully started service 'SparkUI' on port 4042.
15/08/09 21:22:34 INFO SparkUI: Started SparkUI at http://192.168.1.51:4042
15/08/09 21:22:34 INFO Executor: Starting executor ID driver on host localhost
15/08/09 21:22:34 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 52390.
15/08/09 21:22:34 INFO NettyBlockTransferService: Server created on 52390
15/08/09 21:22:34 INFO BlockManagerMaster: Trying to register BlockManager
15/08/09 21:22:34 INFO BlockManagerMasterEndpoint: Registering block manager localhost:52390 with 265.1 MB RAM, BlockManagerId(driver, localhost, 52390)
15/08/09 21:22:34 INFO BlockManagerMaster: Registered BlockManager
```

localhost:8891/notebooks/WordCount/WordCount.ipynb

jupyter WordCount Last Checkpoint: 2 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Cell Toolbar: None

In [1]:

```
import os
import sys
spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.4.0-bin-hadoop2.6'
if not spark_home:
    raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.4.0

Using Python version 2.7.9 (default, Dec 15 2014 10:37:34)
SparkContext available as sc, HiveContext available as sqlContext.

```

KDD-Notebooks — bash — 120x23
15/08/09 21:22:33 INFO SparkEnv: Registering MapOutputTracker
15/08/09 21:22:33 INFO SparkEnv: Registering BlockManagerMaster
15/08/09 21:22:33 INFO DiskBlockManager: Created local directory at /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/blockmgr-d16baf4-b7e9-4c7e-a33d-15a6d8198406
15/08/09 21:22:33 INFO MemoryStore: MemoryStore started with capacity 265.1 MB
15/08/09 21:22:33 INFO HttpFileServer: HTTP File server directory is /private/var/folders/j4/95k348x940xcz40fkdmgy_n40000
gn/T/spark-ac6b33c1-7252-4e53-a335-f67957821ed7/httpd-a18d50aa-ea36-4339-9c15-604330e30548
15/08/09 21:22:33 INFO HttpServer: Starting HTTP Server
15/08/09 21:22:33 INFO Utils: Successfully started service 'HTTP file server' on port 52389.
15/08/09 21:22:33 INFO SparkEnv: Registering OutputCommitCoordinator
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
15/08/09 21:22:34 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
15/08/09 21:22:34 INFO Utils: Successfully started service 'SparkUI' on port 4042.
15/08/09 21:22:34 INFO SparkUI: Started SparkUI at http://192.168.1.51:4042
15/08/09 21:22:34 INFO Executor: Starting executor ID driver on host localhost
15/08/09 21:22:34 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 52390.
15/08/09 21:22:34 INFO NettyBlockTransferService: Server created on 52390
15/08/09 21:22:34 INFO BlockManagerMaster: Trying to register BlockManager
15/08/09 21:22:34 INFO BlockManagerMasterEndpoint: Registering block manager localhost:52390 with 265.1 MB RAM, BlockManagerId(driver, localhost, 52390)
15/08/09 21:22:34 INFO BlockManagerMaster: Registered block manager localhost:52390 with 265.1 MB RAM
```

The screenshot shows a Mac OS X desktop environment with several open windows:

- Terminal Window:** Titled "KDD-Notebooks — bash — 120x23", it displays a log of Spark system startup messages.
- Finder Window:** Titled "closure", showing a presentation slide titled "Presentation".
- Browser Window:** Titled "PySparkShell application UI", showing the Spark Jobs page. It includes sections for "Event Timeline", "Executors", and "Jobs", along with a timeline from Thu 6 August 2015 to Wed 12 August 2015.
- Address Bar:** Shows the URL "192.168.1.51:4042/jobs/".
- Toolbar:** Includes icons for In, Ti, K, P, O, Te, A, A, S, K, L, W, Li, W, H, S, S, S, C, pr, h, L, A, K, G, P, [S, w, m, D, D, D, D, H, T, D, ht, W, W, x].
- Menu Bar:** Shows "You" with a yellow warning icon.

Spark shell in Scala

```
Pacos-MacBook-Pro-3:spark ceteri$ ./bin/spark-shell
Welcome to
SPARK
version 1.0.0

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
2014-07-02 09:34:18.799 java[3899:f17] Unable to load realm info from SCDynamicStore
Spark context available as sc.

scala> ■
```

Large Scale Distributed Data Science using Spark © KDD2015 James G. Shanahan Contact:James.Shanahan @ gmail.com

TAB lists the available methods and attributes

The screenshot shows a Jupyter Notebook interface with a Python 2 kernel. In the top cell (In [1]), the following code is visible:

```
sys.path.insert(0, os.path.join(sp
sys.path.insert(0, os.path.join(sp
execfile(os.path.join(spark_home,

```

In the bottom cell (In [33]), the user has typed "sc." followed by a TAB key, triggering an autocomplete dropdown. The dropdown lists several methods and attributes of the sc variable, including:

- sc.environment
- sc.getLocalProperty
- sc.hadoopFile
- sc.hadoopRDD
- sc.master
- sc.newAPIHadoopFile
- sc.newAPIHadoopRDD
- sc.parallelize
- sc.pickleFile
- sc.pythonExec

A yellow box highlights the word "Autocomplete" next to the dropdown menu. The background of the cell shows some code related to setting up Spark.

Tutorial Outline

- **Part 1: Introduction [30 minutes]**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics [1.5 hours concept+exercises]**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark 3-4 hours**
 - Naïve Bayes +
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up [20]**
 - Latest version of Spark and beyond

Part 1

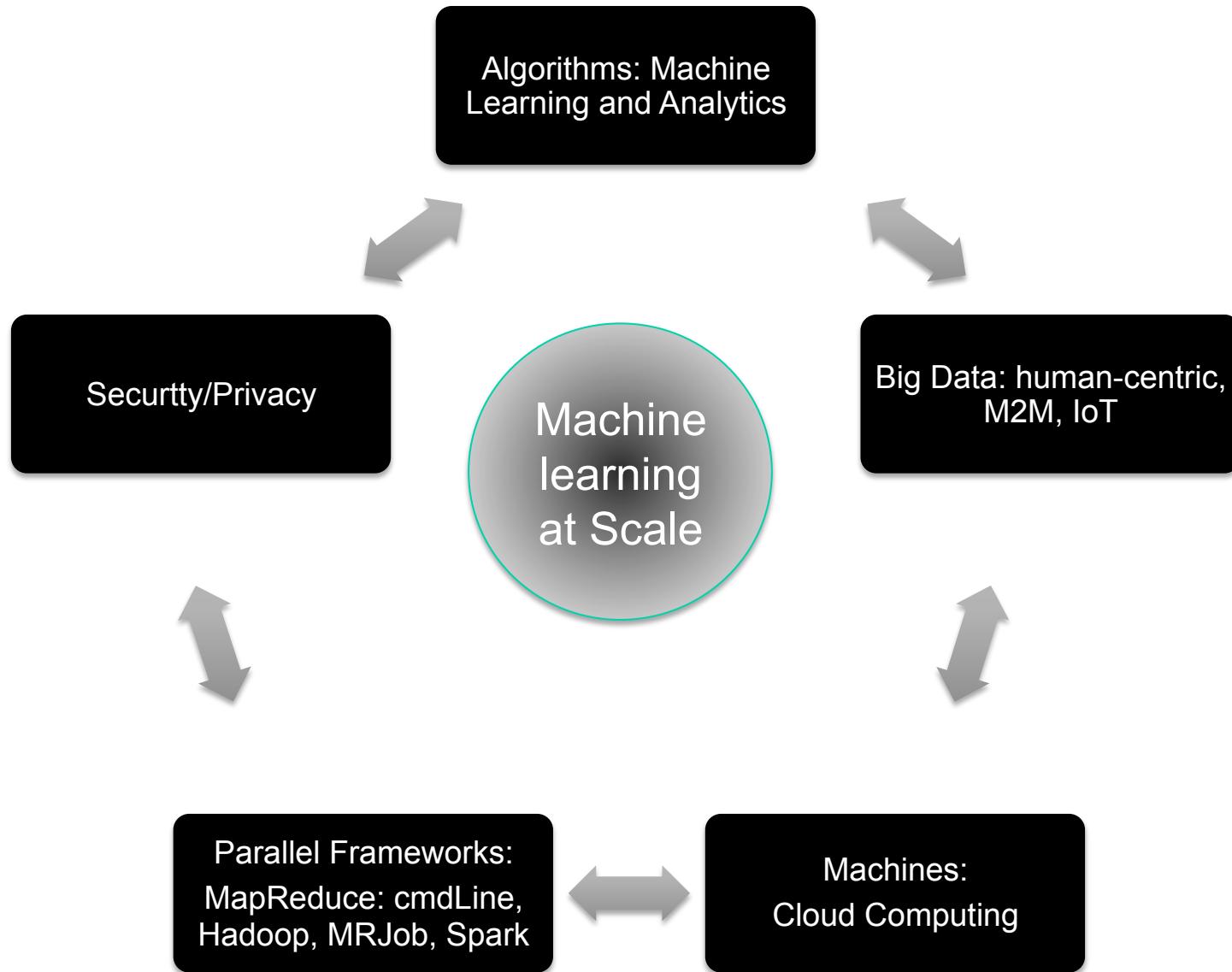
- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

- **Background and Motivation**

- Big Data Science
- Functional Programming
- Poorman's Map-Reduce (to dividing and conquering)

Machine learning at Scale



Big data Definition: use

- Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate.
 - PROCESSING:
 - Think of your laptop that gets overwhelmed with 3-4 gig of data (disk space is 1TB)
 - STORAGE:
 - Laptop : 1 TB
 - THROUGH-PUT
 - 1TB would take 3 hours to read it using your laptop
- Challenges
 - Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, security, and information privacy.

-
- In 2012, Gartner updated its definition as follows:
"Big data is high volume, high velocity, and/or
high variety information assets that require new
forms of processing to enable enhanced decision
making, insight discovery and process
optimization." [18]

Big Data: V³

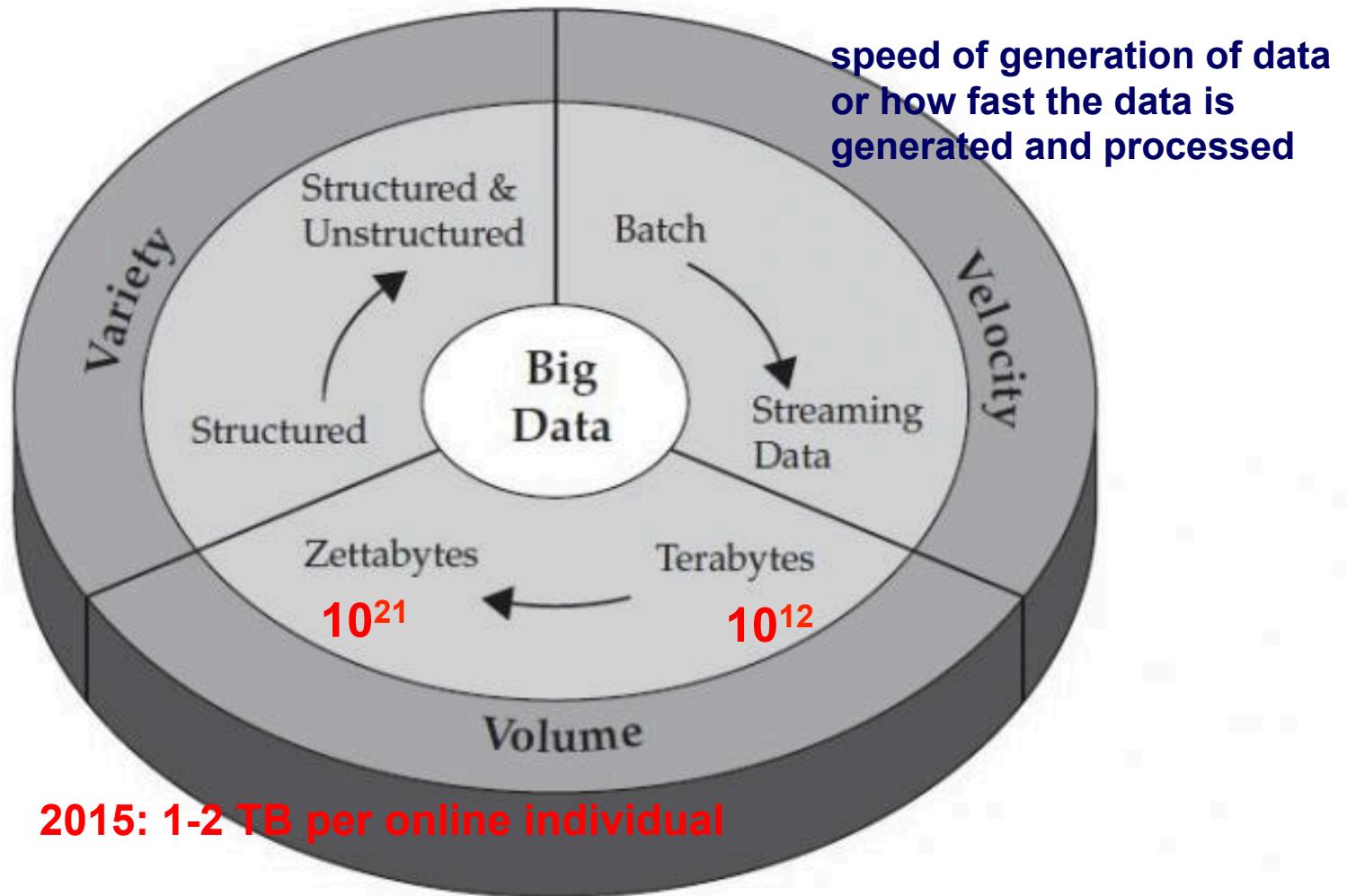
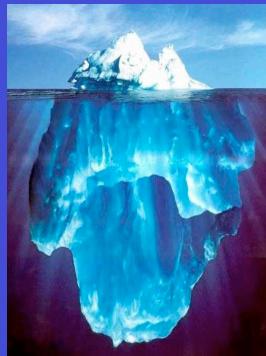


Figure 1-1 IBM characterizes Big Data by its volume, velocity, and variety—or simply, V³.

Sources Driving Big Data

It's All Happening On-line



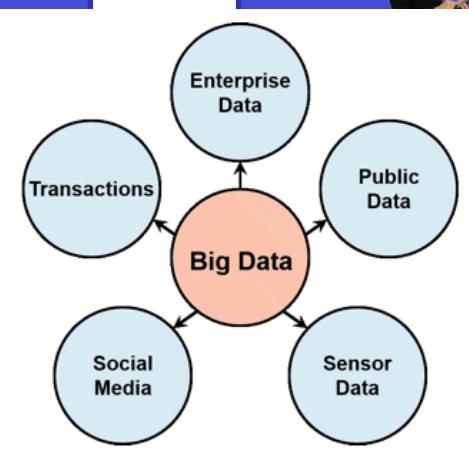
Every:
Click
Ad impression
Billing event
Fast Forward, pause,...
Friend Request
Transaction
Network message
Fault
...

User Generated (Web, Social & Mobile) Quantified Self

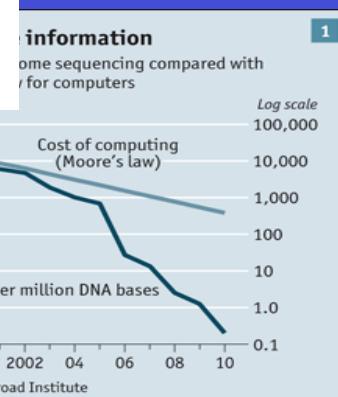


...

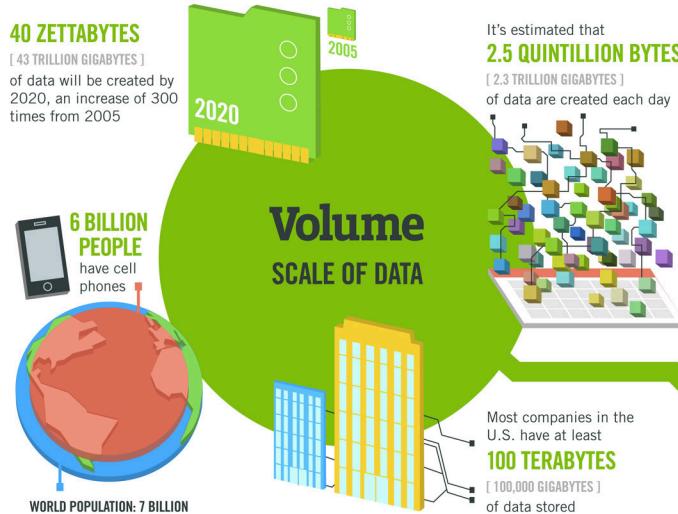
Internet of Things / M2M



Genomic Computing



By 2005 we had $120 \cdot 10^{18}$
 By 2007 we had $280 \cdot 10^{18}$
 By 2020 we will have $40 \cdot 10^{21}$



The New York Stock Exchange captures

1 TB OF TRADE INFORMATION during each trading session



Modern cars have close to **100 SENSORS** that monitor items such as fuel level and tire pressure

Velocity ANALYSIS OF STREAMING DATA

By 2016, it is projected there will be

18.9 BILLION NETWORK CONNECTIONS – almost 2.5 connections per person on earth



Big Data Infographic

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume**, **Velocity**, **Variety** and **Veracity**.

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015 **4.4 MILLION IT JOBS** will be created globally to support big data, with 1.9 million in the United States



As of 2011, the global size of data in healthcare was estimated to be

150 EXABYTES [161 BILLION GIGABYTES]



30 BILLION PIECES OF CONTENT

are shared on Facebook every month



Variety DIFFERENT FORMS OF DATA



By 2014, it's anticipated there will be **420 MILLION WEARABLE, WIRELESS HEALTH MONITORS**

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month



400 MILLION TWEETS are sent per day by about 200 million monthly active users

Poor data quality costs the US economy around

\$3.1 TRILLION A YEAR



1 IN 3 BUSINESS LEADERS don't trust the information they use to make decisions



27% OF RESPONDENTS

in one survey were unsure of how much of their data was inaccurate



Veracity UNCERTAINTY OF DATA

The quality of the data being captured can vary greatly

Sources: McKinsey Global Institute, Twitter, Cisco, Cartier, EMC, SAS, IBM, MERTEC, QAS

<http://www.ibmbigdatahub.com/info/infographic/>

http://www.ibmbigdatahub.com/sites/default/files/infographic_file/4-Vs-of-big-data.jpg



<http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>

Why all the excitement?

- **Government:**
 - Obama used 80 pieces of information on each person; 4 year history (versus Romney)
 - Nate Silver used Bayesian techniques to publish analyses and predictions related to the 2008 and 2012 United States presidential election
- **Sports:**
 - Oakland Athletics baseball team and its manager Billy Beane
- **Transportation (e.g., Autonomous Vehicles)**
- **HCI: Speech Recognition and Translation**
- **Healthcare**
 - AI Cure: Do you know if your patients are taking their meds?
- **Digital Advertising**
- **Search (web, local, mobile)**



3 Vs of Big Data

1-2 TB per person today 2014/2015

The data from these sources has a number of features that make it a challenge for a data warehouse:

Exponential Growth. An estimated 2.8ZB of data in 2012 is expected to grow to 40ZB by 2020. 85% of this data growth is expected to come from new types; with machine-generated data being projected to increase 15x by 2020. (Source IDC)

40TB per person by 2020

Varied Nature. The incoming data can have little or no structure, or structure that changes too frequently for reliable schema creation at time of ingest.

Value at High Volumes. The incoming data can have little or no value as individual, or small groups of records. But high volumes and longer historical perspectives can be inspected for patterns and used for advanced analytic applications.



<http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>

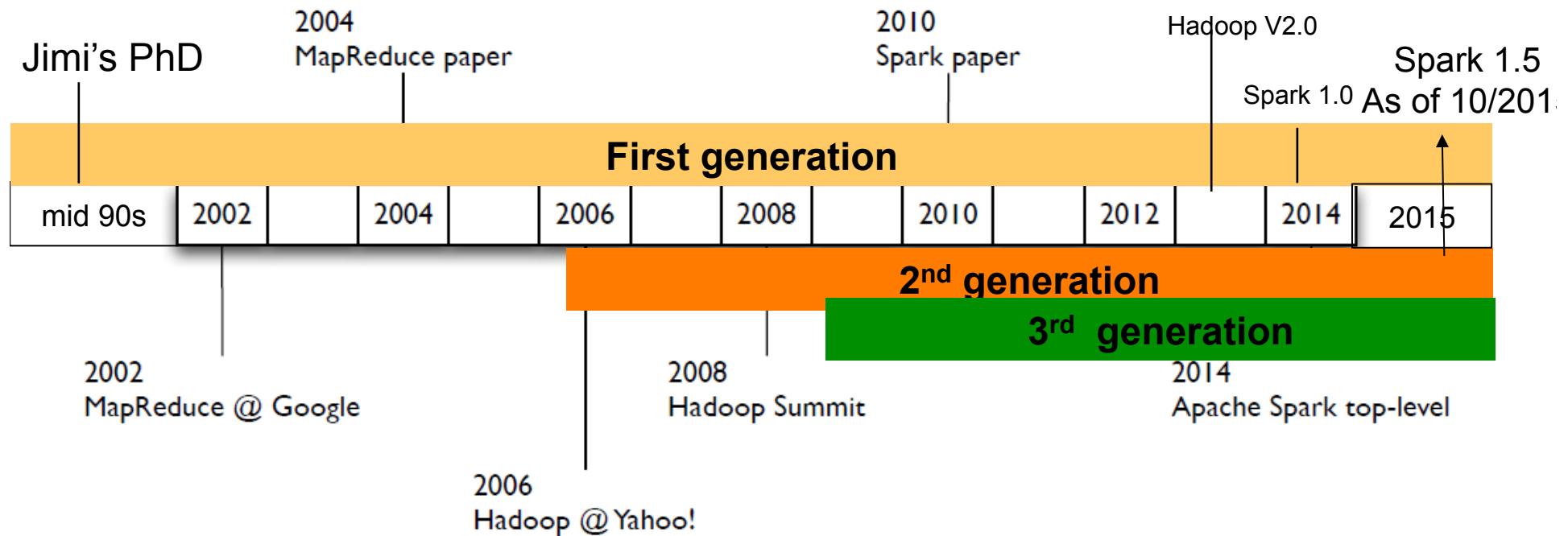
Personal; society; M2M; crowdsourcing

- **Society**
 - Graphs: Social, professional;
 - Quantified self: Eating; Sleeping; exercising
 - Voting
 - Education
 - Healthcare.... Economics, shopping, etc.
- **Internet of things**
 - Tracking Wildebeests in Serengeti, Tanzania (not just with GPS tags, but also with cameras at key strategic locations through out the Serengeti
 - Population changes in species; Scheduling safaris
 - 1 Billion smart meters by 2020;
 - 1 Petabyte of data per day? $10^9 = 10^{12} \text{ to } 10^{15}$
 - 1 Billion smart meters (One megabye of data per device per day; Poll meter 1000 times per day; 1000 bytes of data each time)
 - Smart cities

Three generations of machine learning

- **First generation: dataset that fits in memory**
 - Single node learning summary statistics and some batch modeling (at small scale); SQL, R
 - Down sampling the data
- **Second generation: General purpose clusters and frameworks**
 - Distributed frameworks that allows us to divide and conquer problems
 - Learning using general purpose frameworks such as hadoop big data analysis offline, realtime decision making, homegrown specialist systems (Hadoop for analysis and modeling;), Hadoop, R
 - In-house purpose built systems; specialist sport
- **Third generation: Purpose-built libraries and frameworks**
 - Built for iterative algorithms that are common place in ML
 - huge scale realtime analysis and decision making systems
 - Specialized frameworks for large scale manipulation the type of data you are working with.
 - For example, Machine learning libraries like MLLib in Spark, graph processing libraries like Apache Giraph or GraphX in Spark

Evolution of Map-Reduce frameworks for big data processing



Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Map-Reduce primitives

- **Map-Reduce/Hadoop was inspired by the concept of functional languages:**
 - “Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages....
 - Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.” Jeffrey Dean and Sanjay Ghemawat, Google
- **MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, Google, 2004**

Functional Programming as a guide for Spark

- **apply()**
- **map()**
- **filter()**
- **reduce()**
- **Lambda functions**
- **List comprehensions**

Functional programming background

- Treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- It is a declarative programming paradigm, which means programming is done with expressions.
- In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time.
- No side effects
 - (side effects, i.e. changes in state that do not depend on the function inputs)
 - Can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional Programming

In computer science, **functional programming** is a [programming paradigm](#), a style of building the structure and elements of computer programs, that treats [computation](#) as the evaluation of [mathematical functions](#) and avoids [changing-state](#) and [mutable data](#). It is a [declarative programming paradigm](#), which means programming is done with [expressions](#). In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result $f(x)$ each time. Eliminating [side effects](#), i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming.

Functional programming has its roots in [lambda calculus](#), a [formal system](#) developed in the 1930s to investigate [computability](#), the [Entscheidungsproblem](#), function definition, function application, and [recursion](#). Many functional programming languages can be viewed as elaborations on the lambda calculus. In the other well-known declarative [programming paradigm](#), [logic programming](#), [relations](#) are at the base of respective languages.^[1]

In contrast, [imperative programming](#) changes state with commands in the source language, the most simple example being assignment. Imperative programming does have functions, not in the mathematical sense, but in the sense of [subroutines](#). They can have [side effects](#) that may change the value of program state. Functions without return values therefore make sense. Because of this, they lack [referential transparency](#), i.e. the same language expression can result in different values at different times depending on the state of the executing program.^[1]

Functional programming languages, especially [purely functional](#) ones such as [Hope](#) and [Rex](#), have largely been emphasized in [academia](#) rather than in commercial software development. However, prominent functional programming languages such as [Common Lisp](#), [Scheme](#),^[2]^[3]^[4]^[5] [Clojure](#), [Racket](#),^[6] [Erlang](#),^[7]^[8]^[9] [OCaml](#),^[10]^[11] [Haskell](#),^[12]^[13] and [F#](#)^[14]^[15] have been used in industrial and commercial applications by a wide variety of organizations. Functional programming is also supported in [Large Scale Distributed Data Science using Spark](#) © KDD2015 James G. Shanahan Contact:james.shanahan@gmail.com

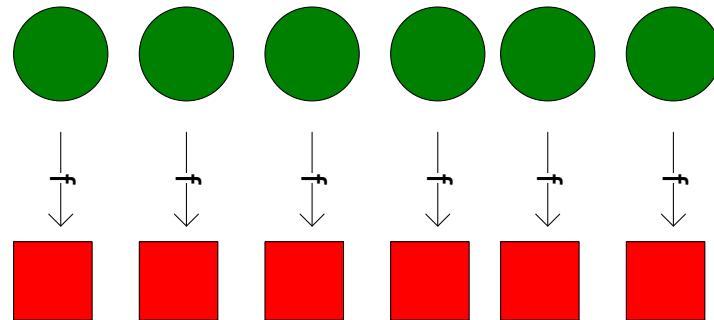
Higher-Order Functions

- A key feature of functional languages is the concept of higher order functions, or functions that can accept other functions as arguments (e.g., APL, Lisp and ML)
- A higher-order function is a function that takes another function as a parameter
- They are “higher-order” because it’s a function of a function
- Examples
 - Map (aka apply to all)
 - Reduce (aka fold)
 - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

Map

```
map(function, iterable, ...)
```

- Map applies **function** to each element of **iterable** and creates a list of the results
- You can optionally provide more iterables as parameters to map and it will place tuples in the result list
- Map returns an iterator which can be cast to list

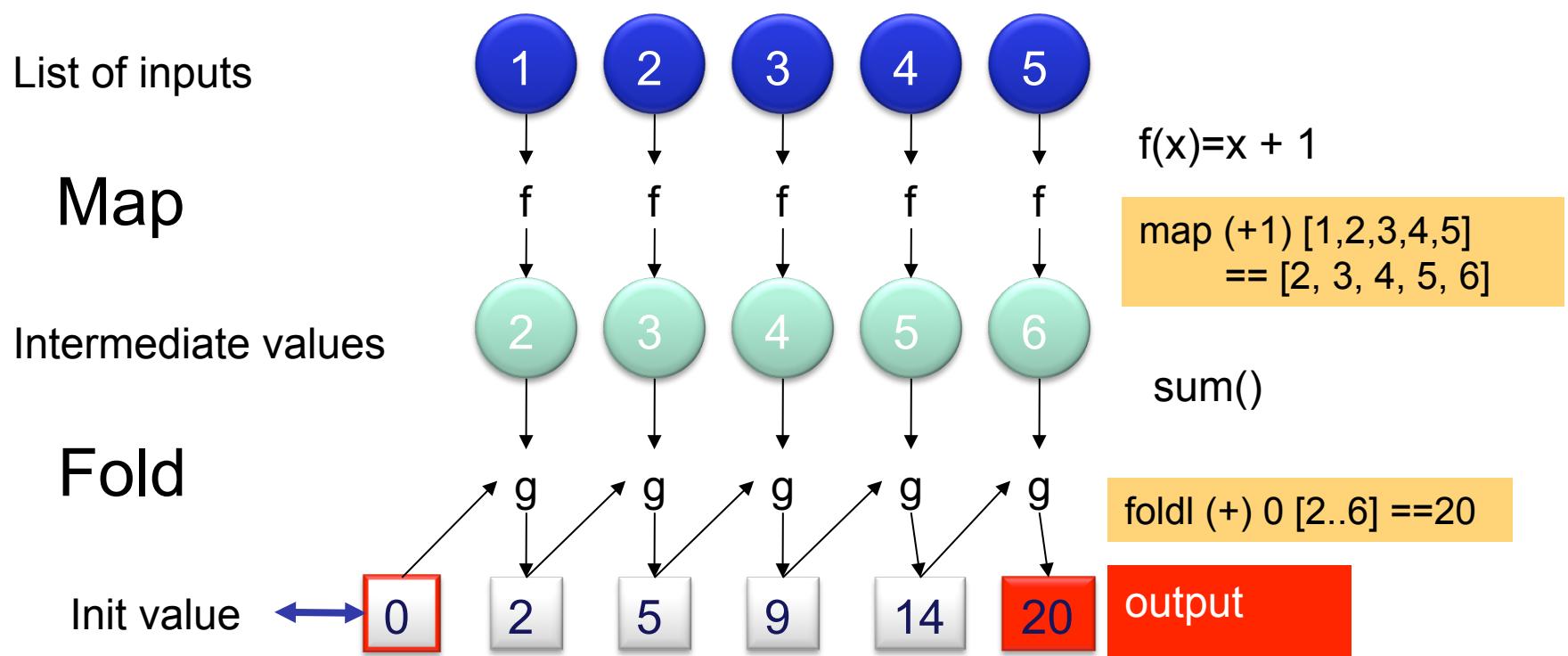


This is an abstract diagram, despite this you can see the potential for parallelism especially in the map phase

MapReduce is derived from FP

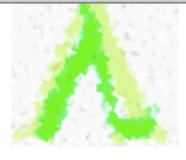
MapReduce ~ Map + Fold from functional programming!

Recursive defn but this unrolls into a loop



Lambda functions

- Shorthand version of def statement, useful for “Inlining” functions and other situations where it's convenient to keep the code of the function close to where it's needed
- Can only contain an expression in the function definition, not a block of statements (e.g., no if statements, etc)
- A lambda returns a function; the programmer can decide whether or not to assign this function to a name



Python 2 Tutorial

- History and Philosophy of Python
- Why Python?
- Interactive Mode
- Execute a Script
- Structuring with Indentation
- Data Types and Variables
- Operators
- input and raw_input via the keyboard
- Conditional Statements
- While Loops
- For Loops
- Formatted output
- Output with Print
- Sequential Data Types
- Dictionaries
- Sets and Frozen Sets
- Shallow and Deep Copy

Lambda, filter, reduce and map

Lambda Operator

Some like it, others hate it and many are afraid of the lambda operator. We are confident that you will like it, when you have finished with this chapter of our tutorial. If not, you can learn all about "[List Comprehensions](#)", Guido van Rossum's preferred way to do it, because he doesn't like Lambda, map, filter and reduce either.

becasu The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce(). The lambda feature was added to Python due to the demand from Lisp programmers.

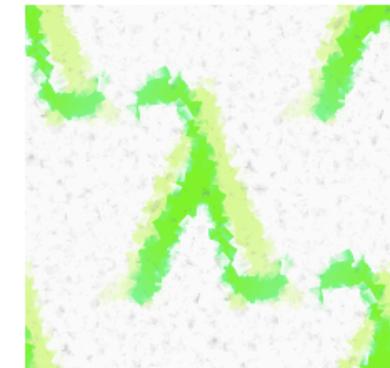
The general syntax of a lambda function is quite simple:

lambda argument_list: expression

The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments. You can assign the function to a variable to give it a name.

The following example of a lambda function returns the sum of its two arguments:

```
>>> f = lambda x, y : x + y
>>> f(1,1)
2
```



The map() Function

The advantage of the lambda operator can be seen when it is used in combination with the map() function. map() is a function with two arguments:

```
r = map(func, seq)
```

The first argument *func* is the name of a function and the second a sequence (e.g. a list) *seq*. *map()* applies the function *func* to all the elements of the sequence *seq*. It returns a new list with the elements changed by *func*

```
def fahrenheit(T):
    return ((float(9)/5)*T + 32)
def celsius(T):
    return (float(5)/9)*(T-32)
temp = (36.5, 37, 37.5, 39)

F = map(fahrenheit, temp)
C = map(celsius, F)
```

In the example above we haven't used lambda. By using lambda, we wouldn't have had to define and name the functions fahrenheit() and celsius(). You can see this in the following interactive session:

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
>>> Farenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)
>>> print Farenheit
[102.56, 97.70000000000003, 99.14000000000001, 100.03999999999991]
```

Lambda example

- Simple example:

```
>>> def sum(x,y): return x+y  
>>> ...  
>>> sum(1,2)  
3
```

```
>>> sum2 = lambda x, y: x+y  
>>> sum2(1,2)  
3
```

Closure (computer programming)

From Wikipedia, the free encyclopedia

Closure

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a [function](#)^[a] together with an environment:^[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.

Example The following program defines a function `startAt` that returns a function `incrementBy`. The nested function `incrementBy` adds its argument `y` to the value of `x`, even though `x` is not local to `incrementBy`. This is because `incrementBy` "captures" the `x` variable from the outer scope and associates it with the `y` value to the `x` value, and finds it once invoked:

```
function startAt(x)
  function incrementBy(y)
    return x + y
  return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(2)
```

Note that, as `startAt` returns a function, `closure1` and `closure2` are associated environments differ, and therefore evaluate to different values, thus evaluating the same code to different results.

A closure is a data structure storing a function[a] together with an environment:

- **a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.**
- A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.

Closure (computer programming)

From Wikipedia, the free encyclopedia

For other uses of this term, including in mathematics and computer science, see [Closure](#).

Not to be confused with [Clojure](#).

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a [function](#)^[a] together with an environment:^[1] a mapping associating each [free variable](#) of the function (variables that are used locally, but defined in an enclosing scope) with the [value](#) or [storage location](#) the name was bound to at the time the closure was created.^[b] A closure—unlike a plain function—allows the function to access those *captured variables* through the closure's reference to them, even when the function is invoked outside their scope.

Example The following program fragment defines a [higher-order function](#) `startAt` with a parameter `x` and a [nested function](#) `incrementBy`. The nested function `incrementBy` has access to `x`, because `incrementBy` is in the lexical scope of `x`, even though `x` is not local to `incrementBy`. The function `startAt` returns a closure containing the function `incrementBy`, which adds the `y` value to the `x` value, and a reference to the variable `x` from this invocation of `startAt`, so `incrementBy` will know where to find it once invoked:

```
function startAt(x)
  function incrementBy(y)
    return x + y
  return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(5)
```

A closure is a data structure storing a function^[a] together with an environment:^[1] a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.

Note that, as `startAt` returns a function, the variables `closure1` and `closure2` are of [function type](#). Invoking `closure1(3)` will return `4`, while invoking `closure2(3)` will return `8`. While `closure1` and `closure2` have the same function `incrementBy`, the associated environments differ, and invoking the closures will bind the name `x` to two distinct variables in the two invocations, with different values, thus evaluating the function to different results.

Lambda as a closure

Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

A closure is a function that encloses its surrounding state by referencing fields external to its body. The enclosed state remains across invocations of the closure.

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

closures

Map Example

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(map(lambda x : x % 5, nums)) Lambda/inline
4
5  print(nums)
6  #[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
7
8
9  def mod5(val) :
10     return x % 5
11
12 nums1 = list(map(mod5, nums))
print(nums1)
#[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
```

apply()

- In very general contexts, you may not know ahead of time how many arguments need to get passed to a function (perhaps the function itself is built dynamically)
- The apply() function calls a given function with a list of arguments packed in a tuple:
`def sum(x, y): return x+y
apply(sum, (3, 4))`
7
- Apply can handle functions defined with def or with lambda

Map Example: distance from origin

Goal: given a list of three dimensional points in the form of tuples, create a new list consisting of the distances of each point from the origin

Loop Method:

- $\text{distance}(x, y, z) = \sqrt{x^2 + y^2 + z^2}$
- loop through the list and add results to a new list

Map Problem: distance from origin

Solution

```
1 from math import sqrt  
2  
3 points = [(2, 1, 3), (5, 7, -3), (2, 4, 0), (9, 6, 8)]  
4  
5 def distance(point) :  
6     x, y, z = point  
7     return sqrt(x**2 + y**2 + z**2)  
8  
9 distances = list(map(distance, points))
```

Filter: higher order function

```
filter(function, iterable)
```

- The filter runs through each element of **iterable** (any iterable object such as a List or another collection)
- It applies **function** to each element of **iterable**
- If **function** returns True for that element then the element is put into a List
- This list is returned from filter in versions of python under 3
- In python 3, filter returns an iterator which must be cast to type list with list()

Filter Example: filter all non-zeros

Example

```
1  nums = [0, 4, 7, 2, 1, 0, 9, 3, 5, 6, 8, 0, 3]
2
3  nums = list(filter(lambda x : x != 0, nums))
4
5  print(nums)          #[4, 7, 2, 1, 9, 3, 5, 6, 8, 3]
6
```

Filter Problem

```
NaN = float("nan")
scores = [[NaN, 12, .5, 78, math.pi],
          [2, 13, .5, .7, math.pi / 2],
          [2, NaN, .5, 78, math.pi],
          [2, 14, .5, 39, 1 - math.pi]]
```

Goal: given a list of lists containing answers to an algebra exam, filter out those that did not submit a response for one of the questions, denoted by NaN

Filter Problem

Solution

```
1  NaN = float("nan")
2  scores = [[NaN, 12, .5, 78, pi],[2, 13, .5, .7, pi / 2],
3              [2,NaN, .5, 78, pi],[2, 14, .5, 39, 1 - pi]]
4  #solution 1 - intuitive
5  def has_NaN(answers) :
6      for num in answers :
7          if isnan(float(num)) :
8              return False
9      return True
0  valid = list(filter(has_NaN, scores))
1  print(valid)
2
3  #Solution 2 - sick python solution
4  valid2 = list(filter(lambda x : NaN not in x, scores))
5  print(valid2)
```

Lambda/inline

Reduce

```
reduce(function, iterable[, initializer])
```

- Reduce will apply **function** to each element in **iterable** along with the sum so far and create a cumulative sum of the results
- **function** must take two parameters
- If initializer is provided, initializer will stand as the first argument in the sum
- Unfortunately in python 3 reduce() requires an import statement
 - from functools import reduce

Reduce Example

Example

```
1  nums = [1, 2, 3, 4, 5, 6, 7, 8]
2
3  nums = list(reduce(lambda x, y : (x, y), nums))
4
5  Print(nums)          # (((((1, 2), 3), 4), 5), 6), 7), 8)
6
7
```

Reduce Problem

Goal: given a list of numbers I want to find the average of those numbers in a few lines using `reduce()`

For Loop Method:

- sum up every element of the list
- divide the sum by the length of the list

Reduce Problem

Solution

```
1 nums = [92, 27, 63, 43, 88, 8, 38, 91, 47, 74, 18, 16,  
2         29, 21, 60, 27, 62, 59, 86, 56]  
3  
4 sum = reduce(lambda x, y : x + y, nums) / len(nums)
```

MapReduce in Python

A framework for processing huge datasets on certain kinds of distributable problems

Map Step:

- master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes.
- worker node may chop its work into yet small pieces and redistribute again

MapReduce

Reduce Step:

- master node then takes the answers to all the sub-problems and combines them in a way to get the output

MapReduce

Problem: Given an email how do you tell if it is spam?

- Count occurrences of certain words. If they occur too frequently the email is spam.

MapReduce in Python: single core

map_reduce.py

```
1 email = ['the', 'this', 'annoy', 'the', 'the', 'annoy']
2
3 def inEmail (x):
4     if (x == "the"):
5         return 1;
6     else:
7         return 0;
8
9 map(inEmail, 1)                      #[1, 0, 0, 0, 1, 1, 0]
10
11 reduce((lambda x, xs: x + xs), map(inEmail, email)) #3
```

-
- Purely functional

Purely functional

Every function in Haskell is a function in the mathematical sense (i.e., "pure"). Even side-effecting IO operations are but a description of what to do, produced by pure code. There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers.

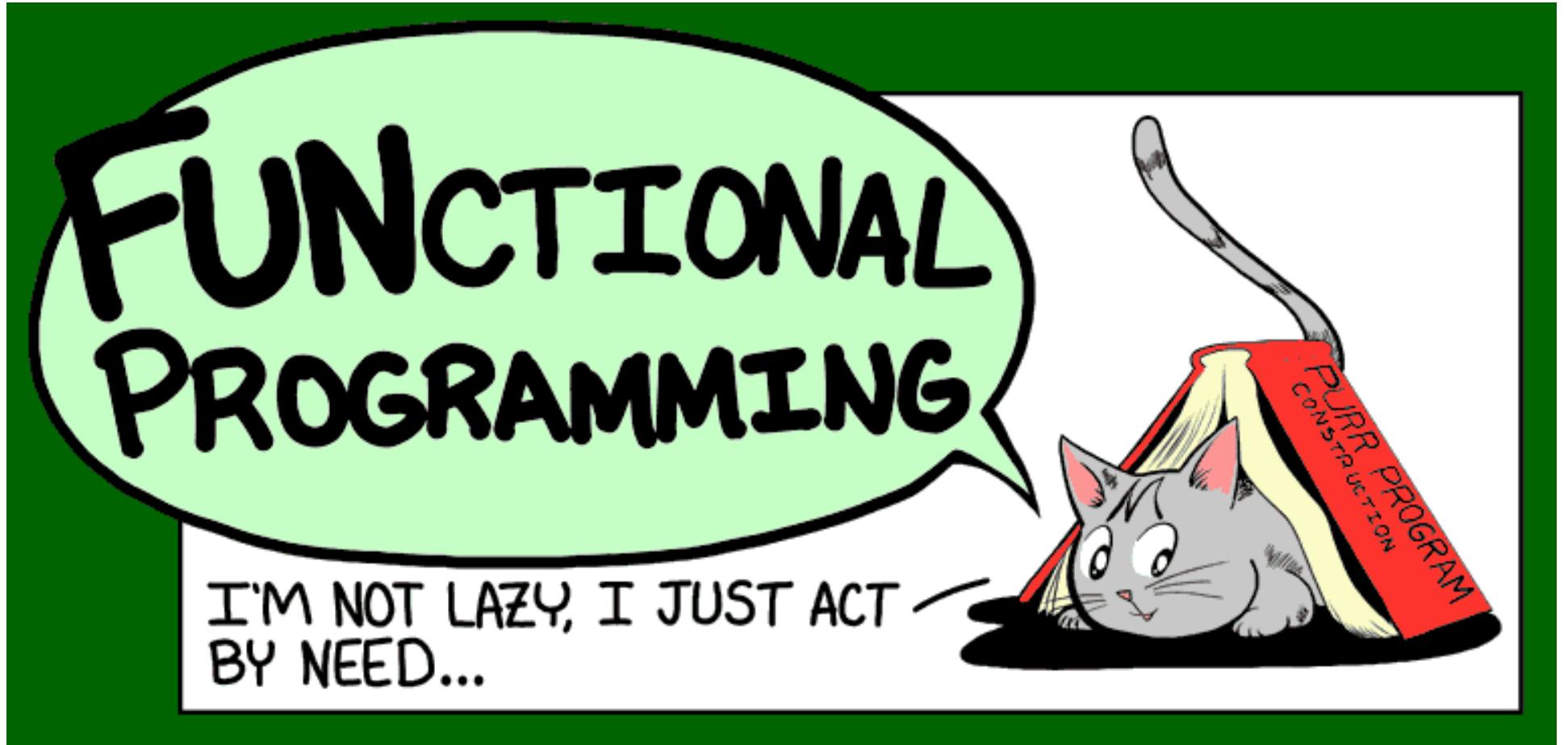
The following function takes an integer and returns an integer. By the type it cannot do any side-effects whatsoever, it cannot mutate any of its arguments.

```
square :: Int -> Int
square x = x * x
```

Functional Programming Review

- Functional operations do not modify data structures: They always create new ones
- Original data still exists in unmodified form
- Data flows are implicit in program design
- Order of operations does not matter
- Lists are primitive data types

Hadoop is not so lazy but Spark is!



-
- In this section we talked about functional programming and have seen the map and reduce higher order functions and we seen the potential to parallelize at least the map function
 - Over the next couple of sections we will see how the Spark framework has adopted FP constructs (albeit not purely observing the FP high standards)
 - Lazy evaluation
 - Data is immutable
 - Map, reduce, and 80 other functions

Part 1

- **Part 1: Introduction**

- Welcome Survey
- Install Spark
- Background and Motivation
 - Big Data Science
 - Functional Programming
 - Poorman's Map-Reduce (to dividing and conquering)

Assume 100 Billion webpages

- **How store them?**
 - $10K \text{ per page } 10^{11} = 10^{15} \text{ Bytes (1 Petabyte of raw data)}$
- **How can we scan them?**
 - $10^{15} \times 10^{-8} \times 10^{-5} = 10^2 \text{ days}$
- **How to do something useful with them?**
 - Document frequency for a term?
 - Extract outlinks of page and say just count the number of inlinks for a webpage

Why Parallelism?

- **Data size is increasing**
 - Single node architecture is reaching its limit
 - Scan 1000 TB on 1 node @ 100 MB/s = 120 days
 - Store that amount of data?
- **Growing demand to leverage data to do useful tasks**
- **Parallelism: Divide and conquer**
- **Standard/Commodity and affordable architecture emerging**
 - Cluster of commodity Linux nodes (CPU, memory and disk)
 - Gigabit ethernet interconnect

Design Goals for Parallelism

1. Scalability to large data volumes:

- Scan 1000 TB (1PB) on 1 node @ 50 MB/s = 120 days
- Scan on 10000-node cluster = 16 minutes!
- $10^{15} \times 10^{-8} \times 10^{-5} = 10^2$ days = 10^7 seconds
- 10^7 seconds X 10⁴ nodes = 10³ seconds = 16 minutes
- 100 Gig per node (10¹¹ bytes at 10⁸ / second)

2. Cost-efficiency:

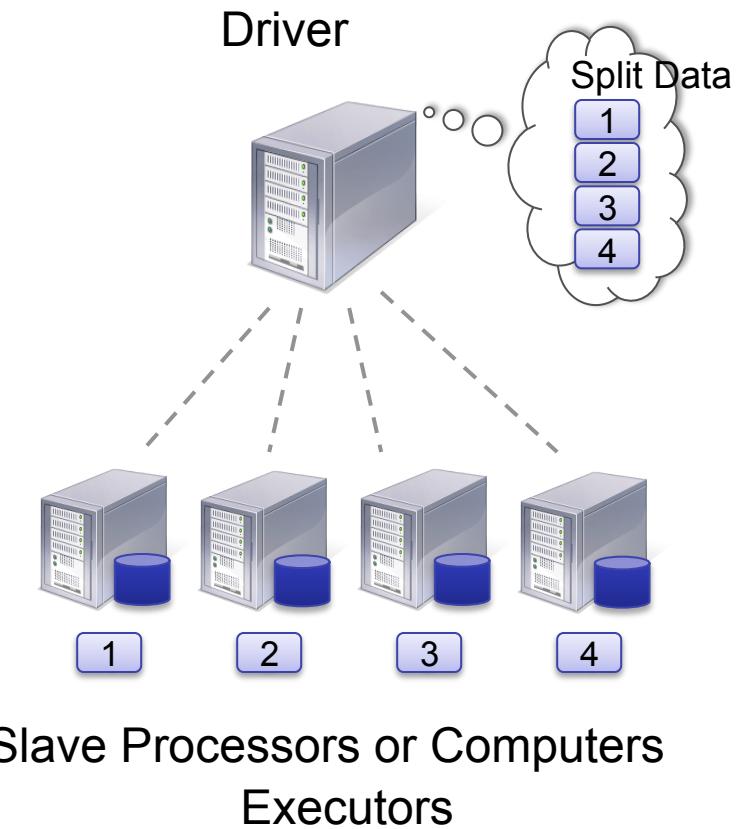
- Commodity nodes (cheap, but unreliable)
- Commodity network
- Automatic fault-tolerance (fewer admins)
- Easy to use (fewer programmers)

Command Line: Divide and Conquer

- Partitions the data
- The framework processes the objects within a partition in sequence, and can process multiple partitions in parallel

Partition and distribute data

Challenges?



-
- **Challenge**
 - State the challenge
 - Tools and rules
 - Timeout to solve

-
- We are going to hack this up using a unix-based divide and conquer strategy
 - AKA poorman's distributed computational framework

Ground Rules

- **Limit ourselves to using the following Unix commands only:**
 - split, grep, wc, merge, cat, for, echo
 - cut, paste and bc #to get a total
 - |, &, wait #parallel computing

grep command

- **Problem: Need to search very large file**
 - Ex: Your java applet records each user that logs in to your website
 - Assume the log file generated is large (everyone wants to see your website!)
 - You want to look at only the lines in log file that contain ‘mcorliss’
- **In unix, use ‘grep’ command**
 - ‘grep <string> <filename>’ returns all lines in file that contain string
 - If string contains ‘ ‘ (space) then need to enclose in quotes
 - <string> is case-sensitive: “mcorliss” != “Mcorliss”

```
prompt$ grep "mcorliss" logfile
Jan 10 10:06:54 log in by mcorliss from 66.94.234.13
Jan 12 11:36:22 log in by mcorliss from 65.92.231.10
...

```

- What if we want only lines with “mcorliss” on January 10?

Regular Expressions

- **Fortunately, grep supports more powerful matching**

- Using regular expressions
 - ‘grep <reg. exp.> filename’

- **For example, to solve previous problem:**

```
prompt$ grep "Jan 10.*mcorliss" logfile
```

```
Jan 10 10:06:54 log in by mcorliss from 66.94.234.13
```

- **Searching for songs whose title starts with “love”**

```
prompt$ grep "^love" musiclibrary.txt
```

- **Searching for a 7-digit phone number in a file:**

```
prompt$ grep "[0-9]\{3\}[-]\{1\}[0-9]\{4\}" file
```

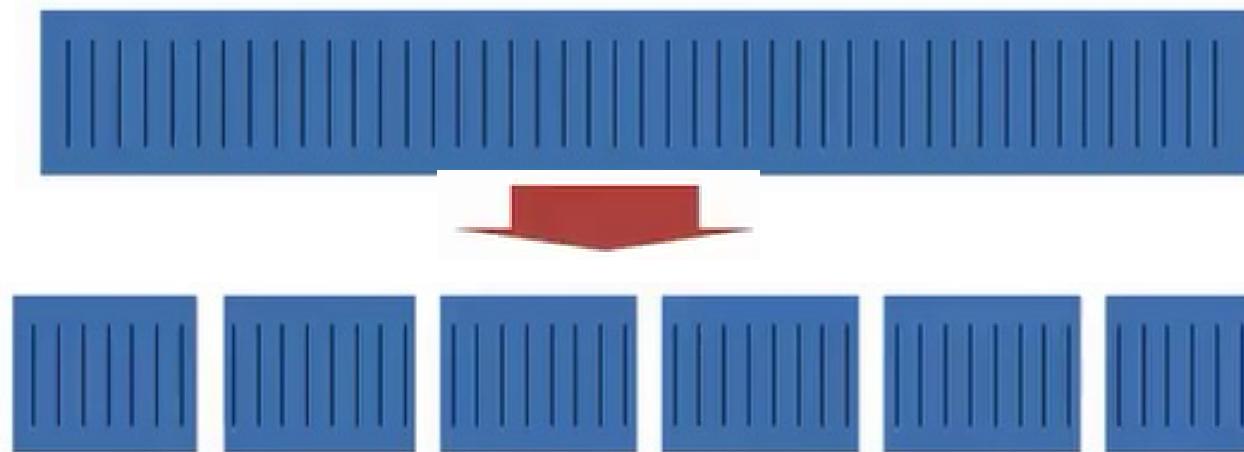
Pipe command “|”: Multiple Matches

- **Regular expressions are powerful**
 - But sometimes want to do multiple independent matches
 - For example, match on lines that contain ‘foo’, ‘goo’, and ‘zoo’
 - Trying to do this using regular expressions, would be very painful
- **Can use pipes denoted as | in Unix:**

```
prompt$ grep foo file | grep goo | grep zoo
```

split a large file into chunks/folds

- **split**
 - `cat logfile.txt | split -l 20 -d fold`
divide `logfile.txt` into files of 20 lines apiece, using “`fold`” as the prefix and with numeric suffixes
- **split -l 30 #30 lines per file**
- **split -b 24m #24 meg chunks**



Unix “split” and “cat” command

Split the file based up on the number of lines

Split the file into multiple pieces based up on the number of lines using -l option as shown below.

```
$ wc -l testfile  
2591 testfile  
$ split -l 1500 testfile importantlog  
$ wc -l *  
1500 importantlogaa  
1091 importantlogab  
2591 testfile
```

divide testfile into files of 1500 lines apiece, using “importantlog” as the prefix and with suffixes “aa”, “ab” in this case

To split *filename* to parts each 50 MB named *partaa*, *partab*, *partac*,....

Split
and
Cat

```
split -b50m filename part
```

To join the files back together again use the *cat* command

```
cat xaa xab xac > filename
```

or

```
cat xa[a-c] > filename
```

Split into chunks, wc

- **split**

- `cat file.txt | split -l 20 -d fold`
divide file.txt into files of 20 lines apiece, using “fold” as the prefix and with numeric suffixes

- **WC**

- WC is a counting utility
 - `wc -[l|c|w] file.txt`
counts number of lines, characters, or words in a file

```
>>>$ wc -l setup.py
```

271 setup.py

271 lines in the setup.py file

For loop in Unix + echo

- `for f in *.txt; do
 echo "file >> $f";
 #cat '$f' | wc -l ;
done`

```
setup.py  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ ls  
CHANGES.txt      LICENSE.txt      build          mlpv  
COPYRIGHT.txt    PKG-INFO       docs           setup.py  
INSTALL.txt     README.txt     gpl-3.0.txt  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ for f in *.txt; do      echo "$f" ; done  
CHANGES.txt  
COPYRIGHT.txt  
INSTALL.txt  
LICENSE.txt  
README.txt  
gpl-3.0.txt  
JAMES-SANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$
```

Get total using cut, paste and bc

```
--  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10  
1  
2 seq generates a sequence of numbers  
3  
4  
5  
6  
7  
8  
9  
10  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10 | paste -sd+ - | bc  
55  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ seq 10 >tmpp  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ cat tmpp
```

```
1  
2  
3 cat tmpp | cut -f 1 | paste -sd+ - | bc  
4  
5  
6 JAMES-SHANAHANS-Desktop-Pro-2:Notebooks jshanahan$ bc <<< "2+2"  
7 4  
8  
9  
10 1+2+3....+10|bc
```

```
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$ cat tmpp|cut -f 1|paste -sd+ - | bc  
55  
JAMES-SHANAHANS-Desktop-Pro:mlpy-3.5.0 jshanahan$
```

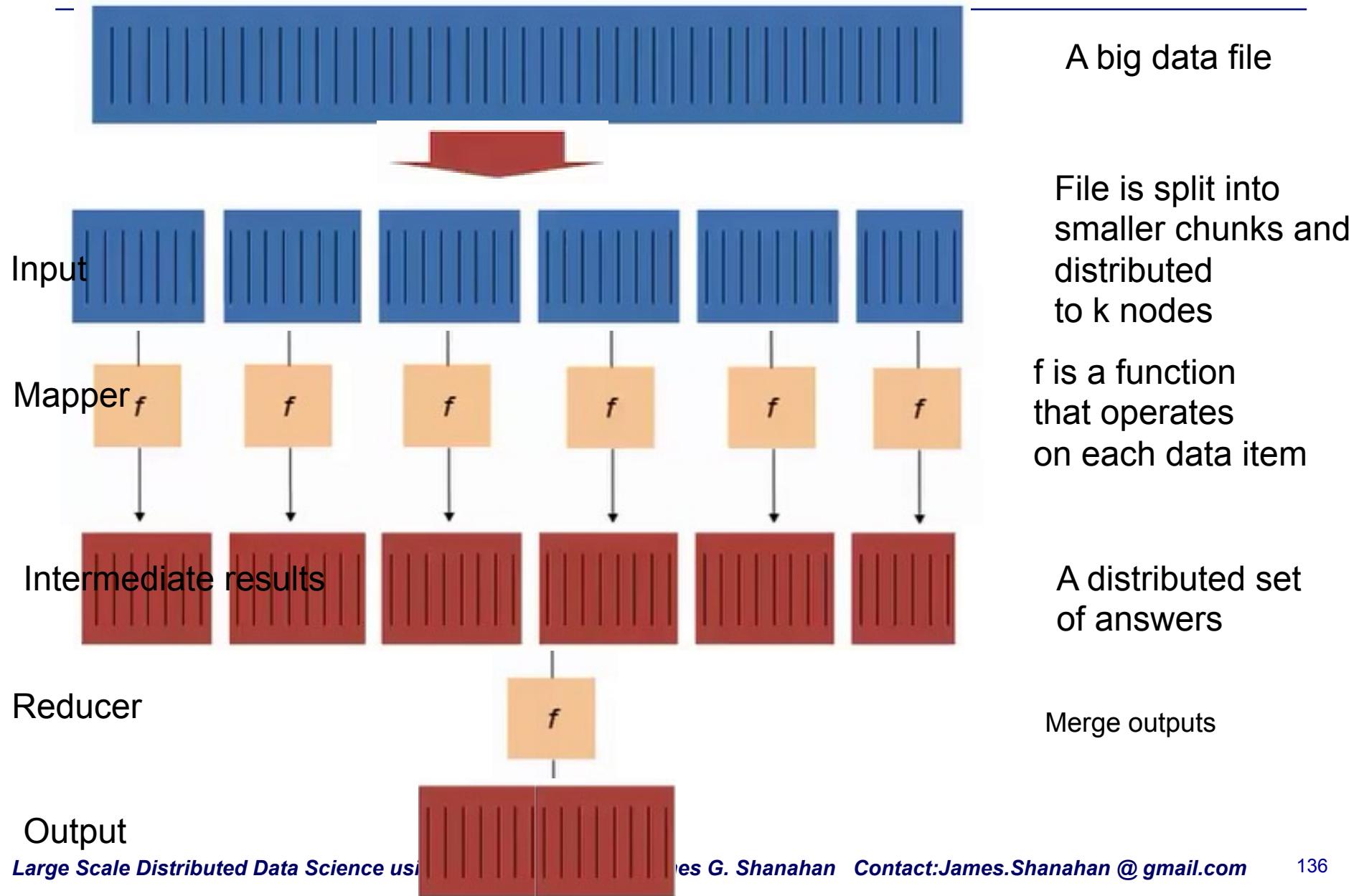
Parallel computing with “&” and wait

- The “**&**” causes parent process to spawn off parallel processes...
- And **wait** will cause the parent process to wait until child processes have finished

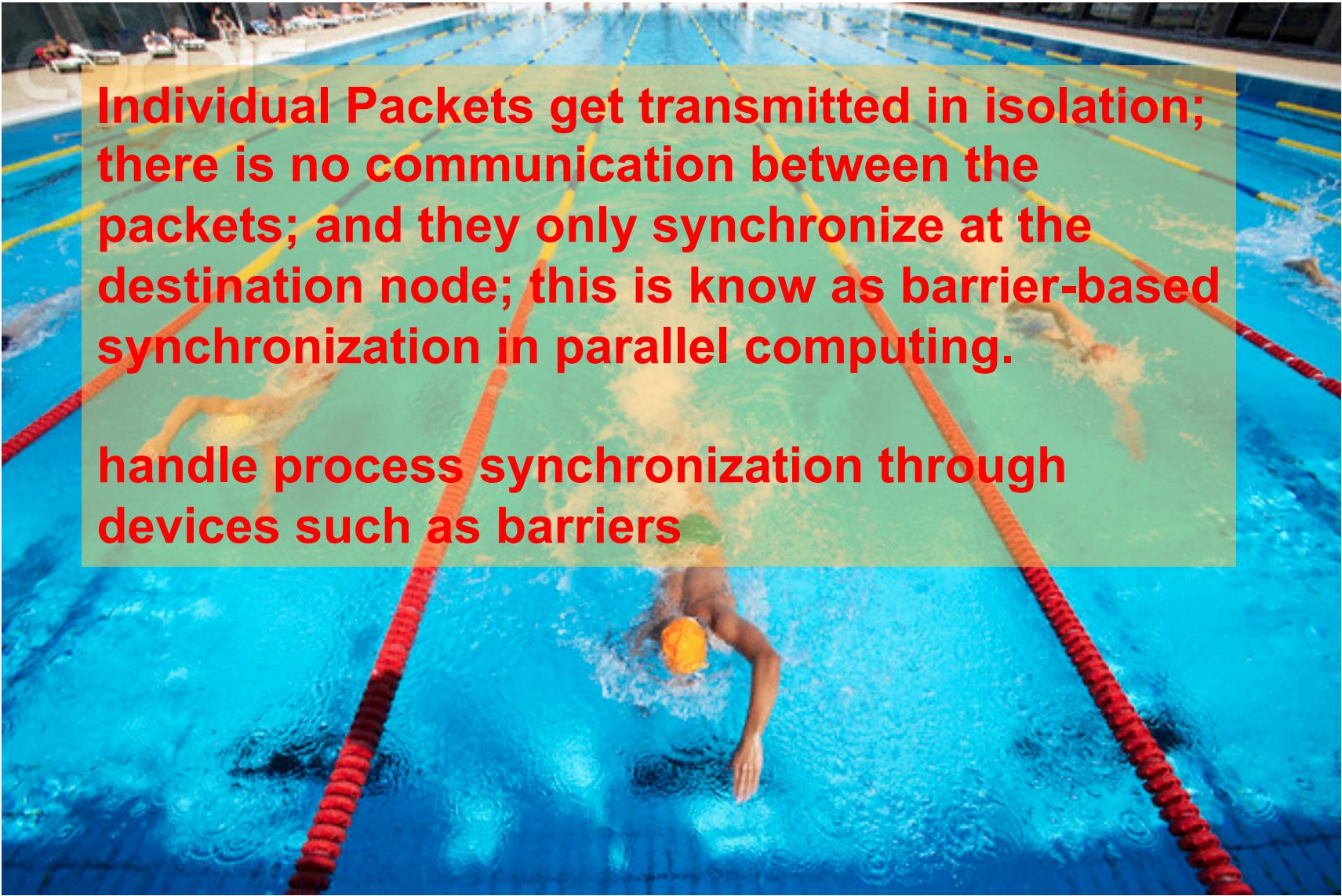
```
1 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ seq 10000000|wc & Child process;  
2 [1] 72233 takes time  
3 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ wait; echo "Finished waiting"  
4 10000000 10000000 113888814  
5 [1]+ Done seq 10000000 | wc  
6 Finished waiting Prints only after waiting  
7 JAMES-SHANAHANS-Desktop-Pro:~ jshanahan$ █
```

- Here **seq 1000000|wc &** causes the parent terminal process to spawn off a process to do this line count task.
- Meanwhile the next command “**wait**” causes the shell to wait until the child process (**seq 1000000|wc**) finishes. Once it finishes the shell can continue and run the echo command

Schematic of Parallel Processing



Individual Packets get transmitted in isolation

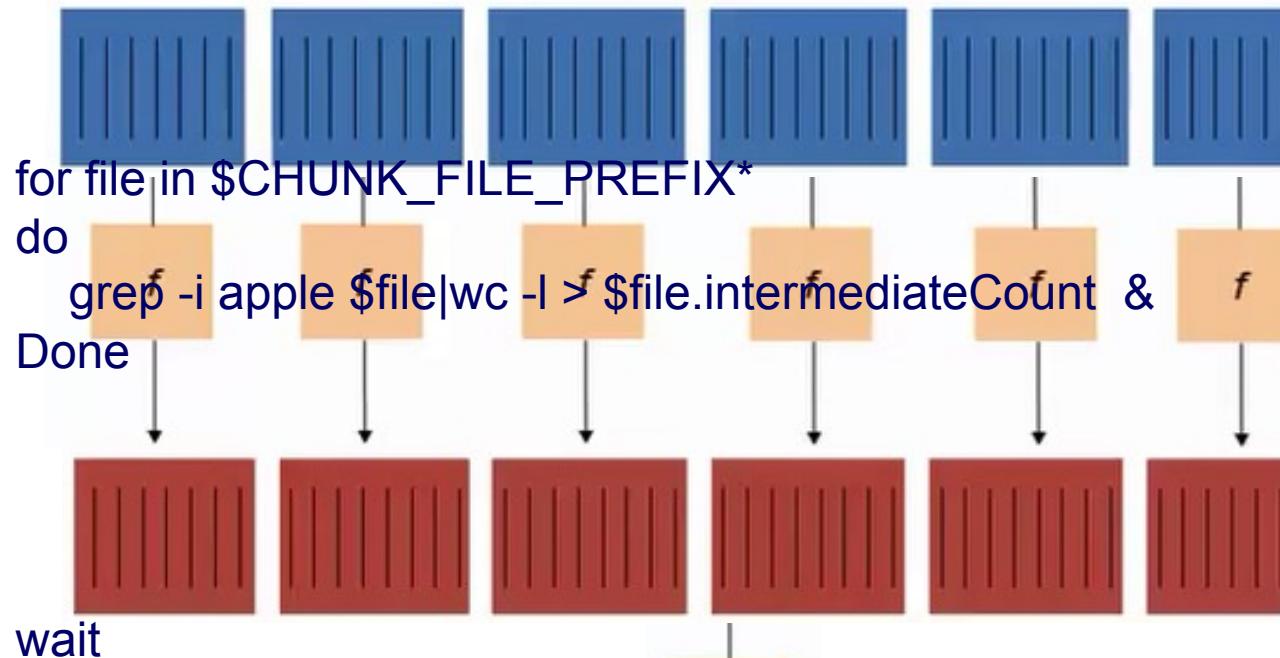


Individual Packets get transmitted in isolation; there is no communication between the packets; and they only synchronize at the destination node; this is known as barrier-based synchronization in parallel computing.

handle process synchronization through devices such as barriers

Schematic of Parallel Processing

1. #Splitting \$ORIGINAL_FILE into chunks ...
2. split -b 10000M \$ORIGINAL_FILE \$CHUNK_FILE_PREFIX



A big data file

File is split into smaller 100Gig chunks and distributed to k nodes

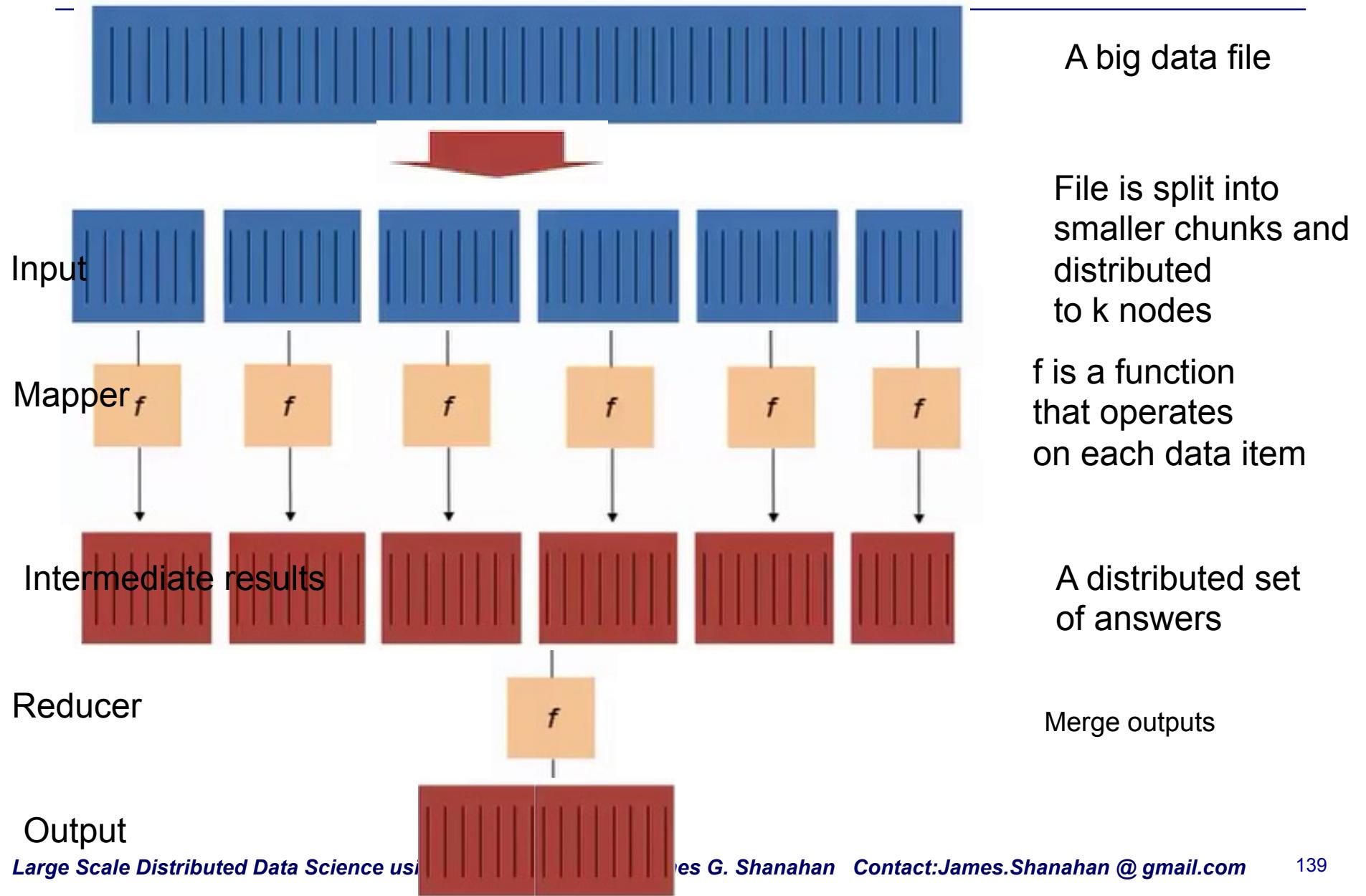
f is a function that operates on each data item

A distributed set of answers

Merge outputs

```
#Merging intermediate Count can take first column and total...
echo "found this many apples in the Facebook posts log file"
cat *.intermediateCount cut -f 1 | paste -sd+ - | bc
```

Schematic of Parallel Processing



1. ORIGINAL_FILE=\$1
2. CHUNK_FILE_PREFIX=\${ORIGINAL_FILE}.split.
3. SORTED_CHUNK_FILES=\${CHUNK_FILE_PREFIX}*.*.sorted

Solution

```

4. usage ()                                pGrepCount
5. {
6.   echo Parallel grep
7.   echo usage: pGrepCount file1 100
8.   echo greps file file1 for a “apple” and counts the number of lines
9.   echo Note: file1 will be split in chunks up to 10Gig Chunks
10.  echo and each chunk will be grepCounted in parallel
11. }
```

12. # *Splitting \$ORIGINAL_FILE into chunks ...*

13. split -b 10000M \$ORIGINAL_FILE \${CHUNK_FILE_PREFIX}

14. # *Distribute*

15. for file in \${CHUNK_FILE_PREFIX}*;

16. do

17. grep -i apple \$file | wc -l > \$file.intermediateCount &

Mapper

18. done

19. wait

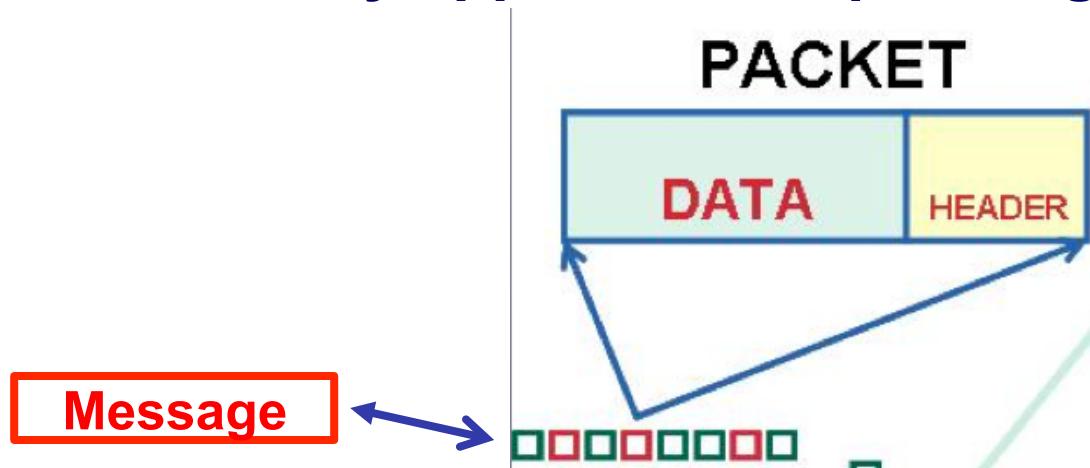
20. # *Merging intermediate Count can take first column and total...*

21. numOflnstances=\$(cat *.intermediateCount | cut -f 1 | paste -sd+ - | bc) Reducer

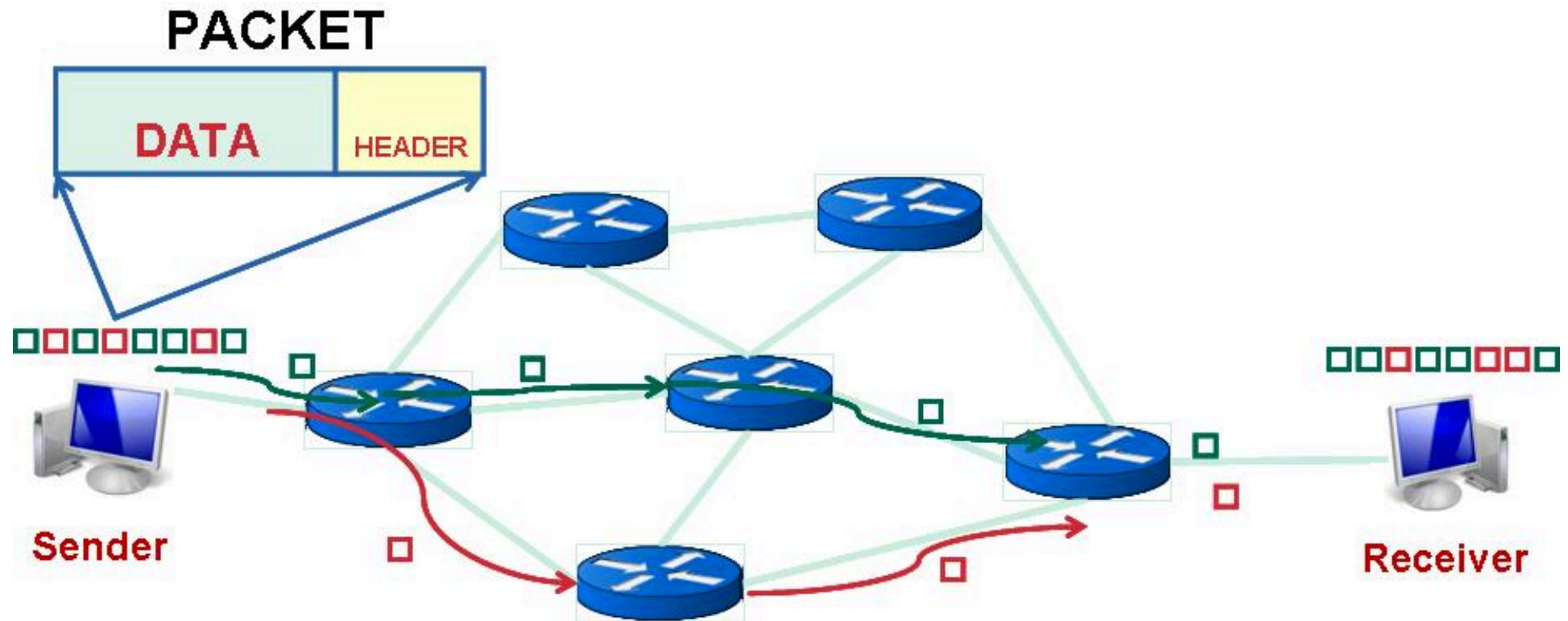
22. echo “found [\${numOflnstances}] of apples in the Facebook posts log file”

D&C is common: e.g., Packet switching

- Packet switching is a digital networking communications method that transmits messages in chunks or blocks, called *packets*
- Packets are transmitted via a medium that may be shared by multiple simultaneous communication sessions.
- Packet switching increases network efficiency, robustness and enables technological convergence of many applications operating on the same network.

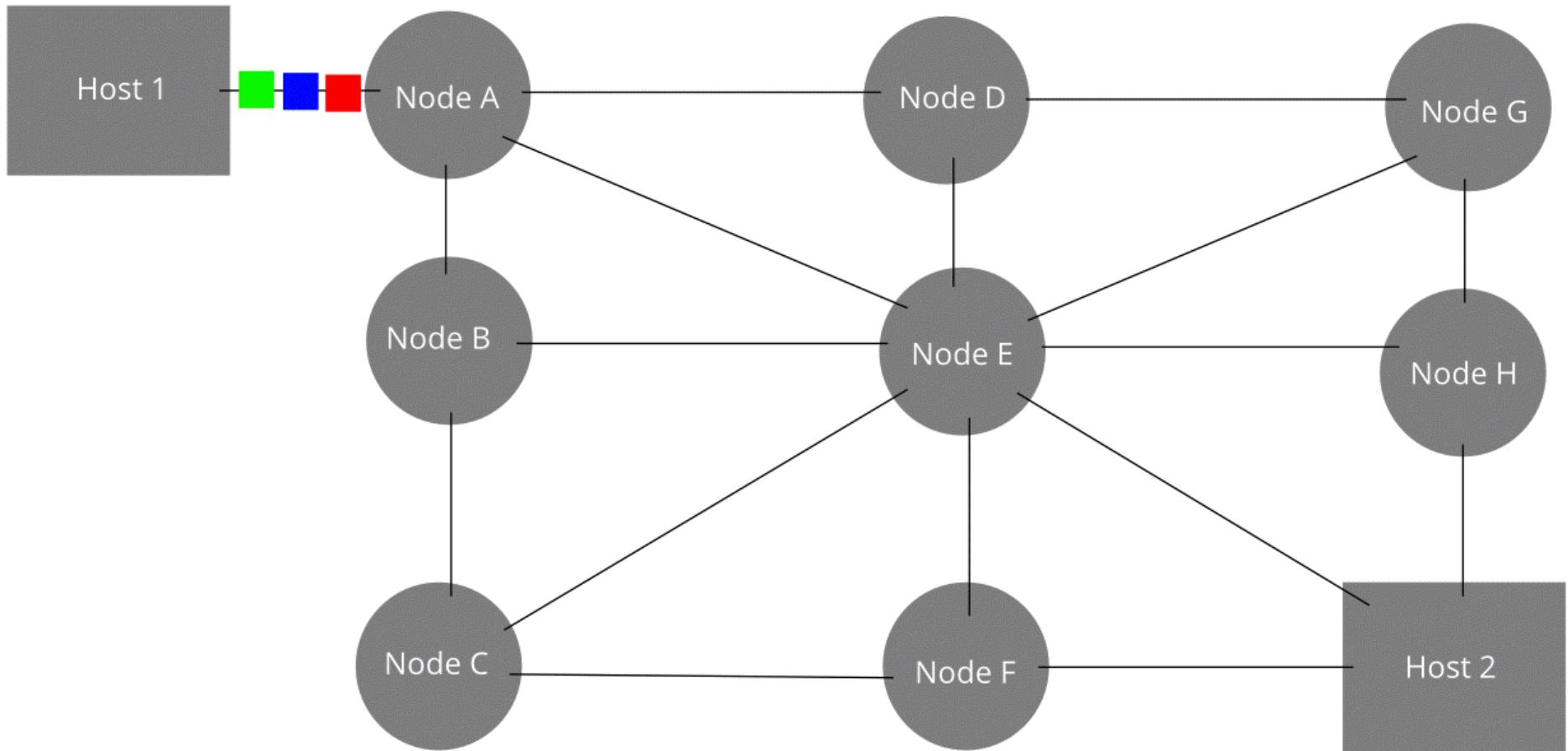


Packet = Header and Payload



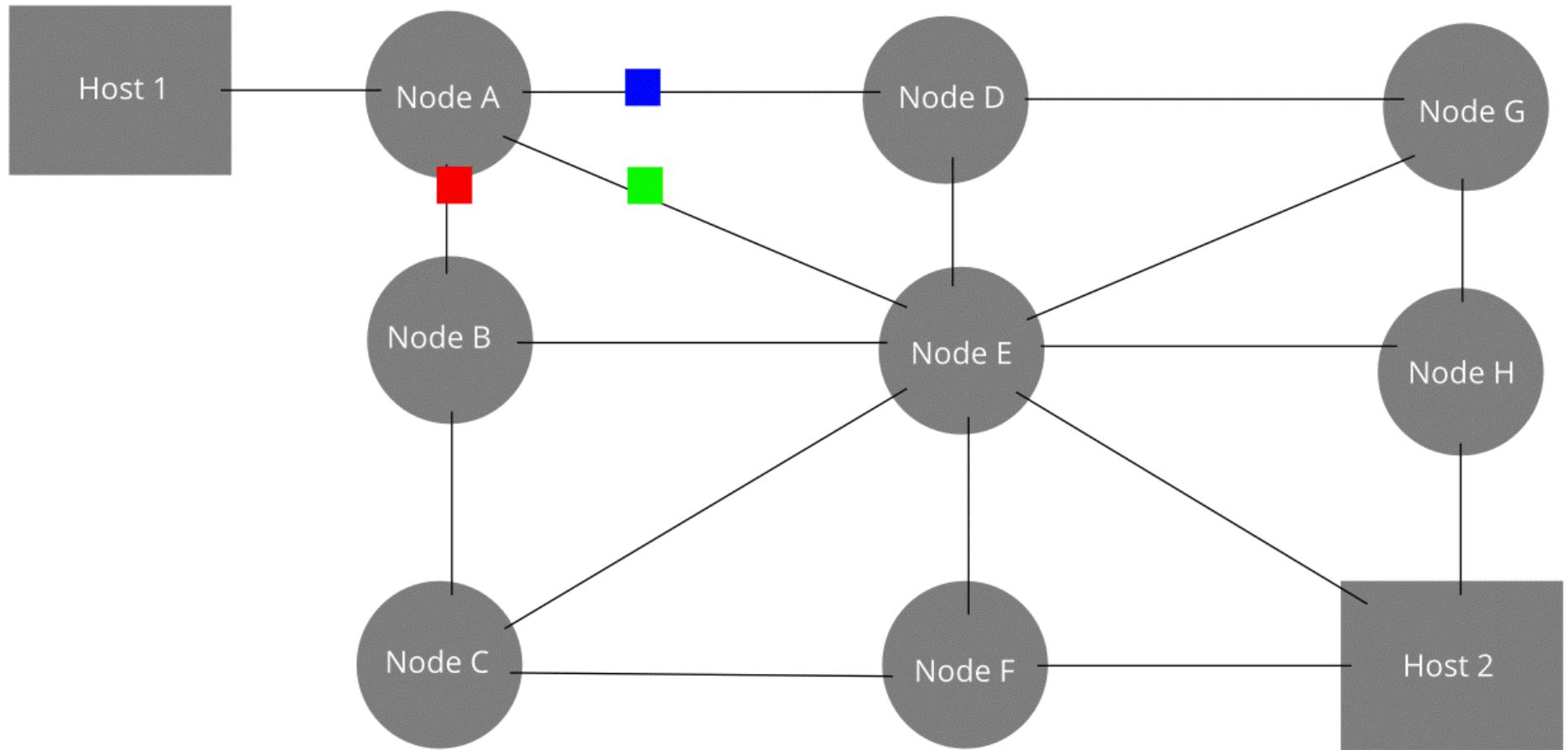
Message Passing: Divide and conquer/send

The original message is Green, Blue, Red.



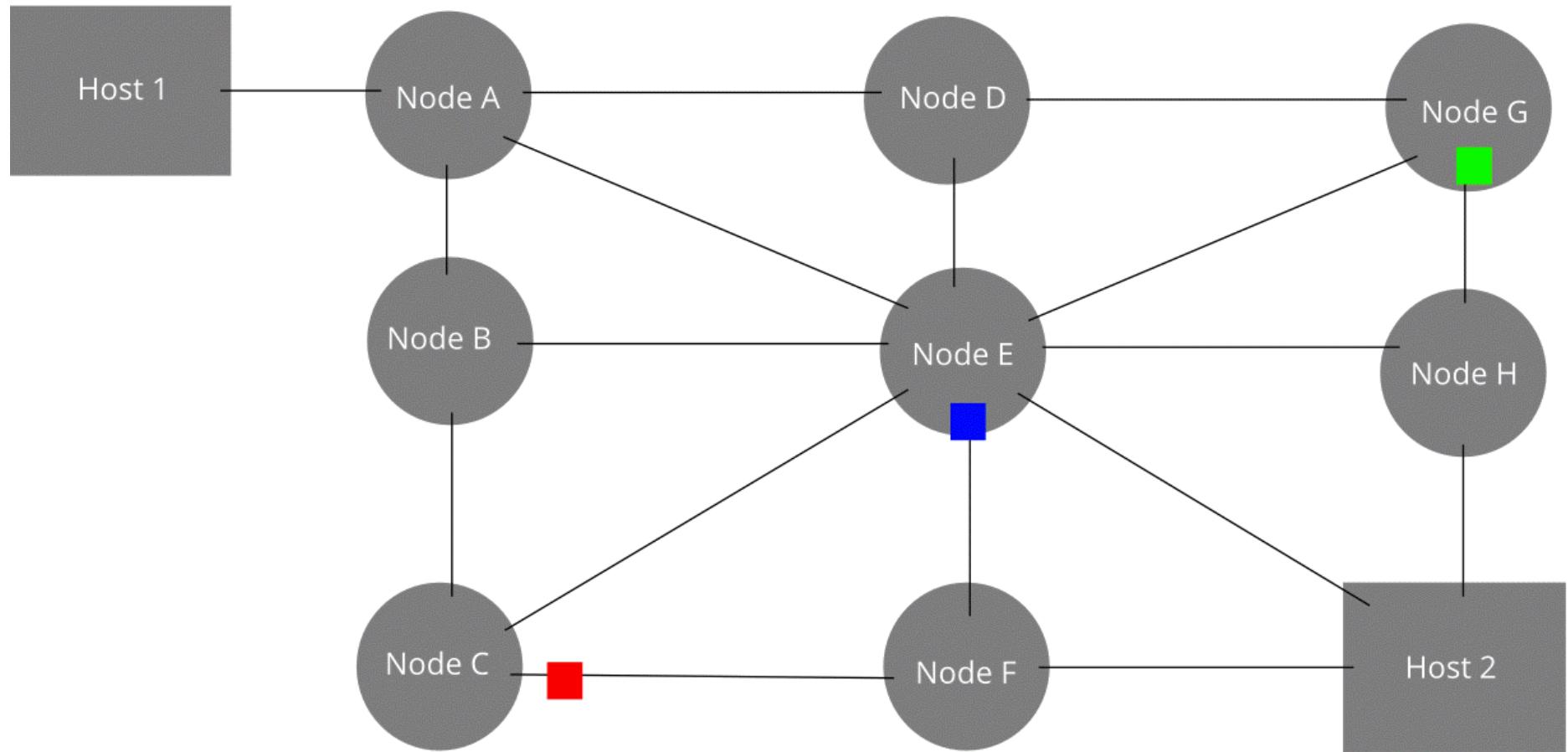
Route packets in parallel if possible

The data packets take different routes to their destinations.



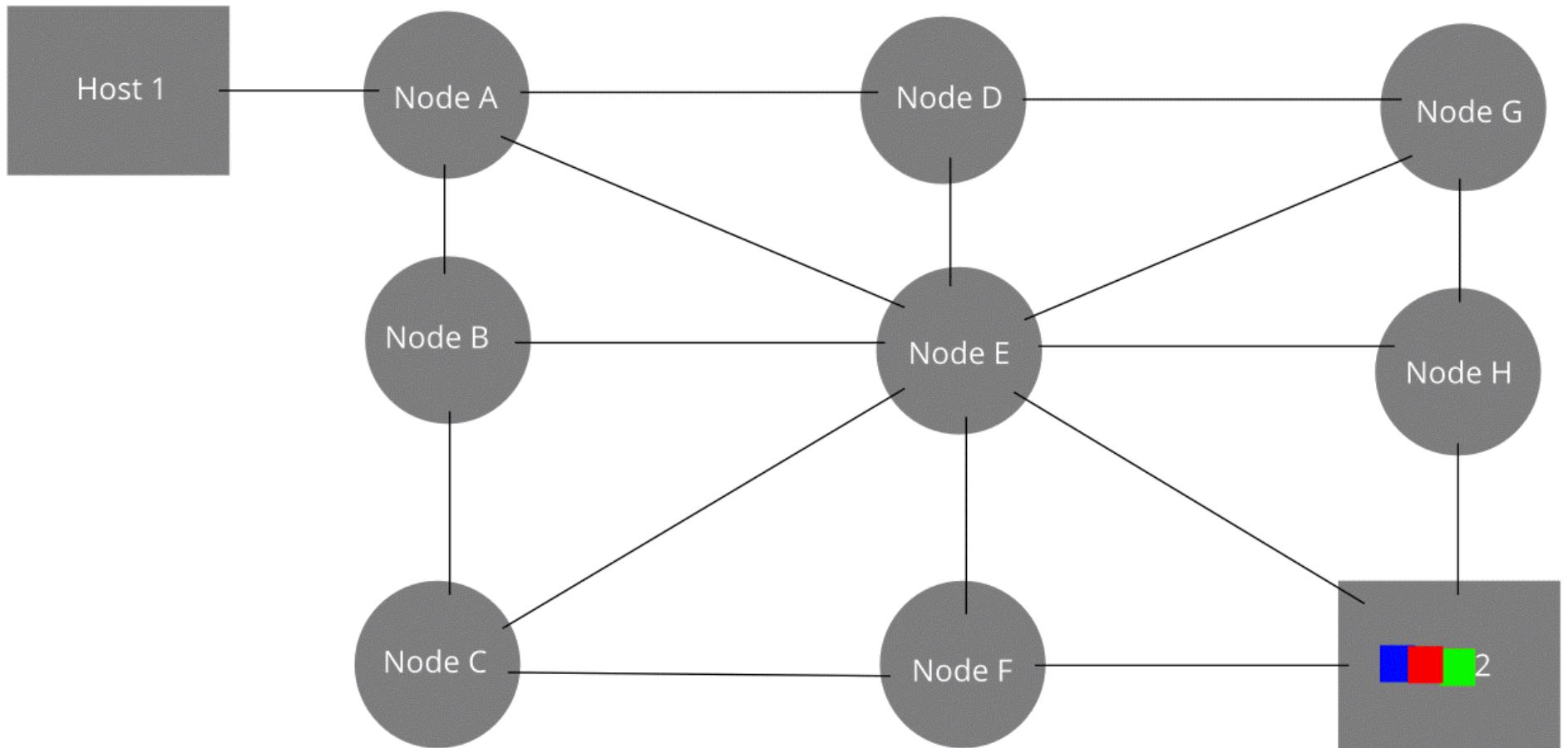
Routing packets in parallel if possible

The data packets take different routes to their destinations.



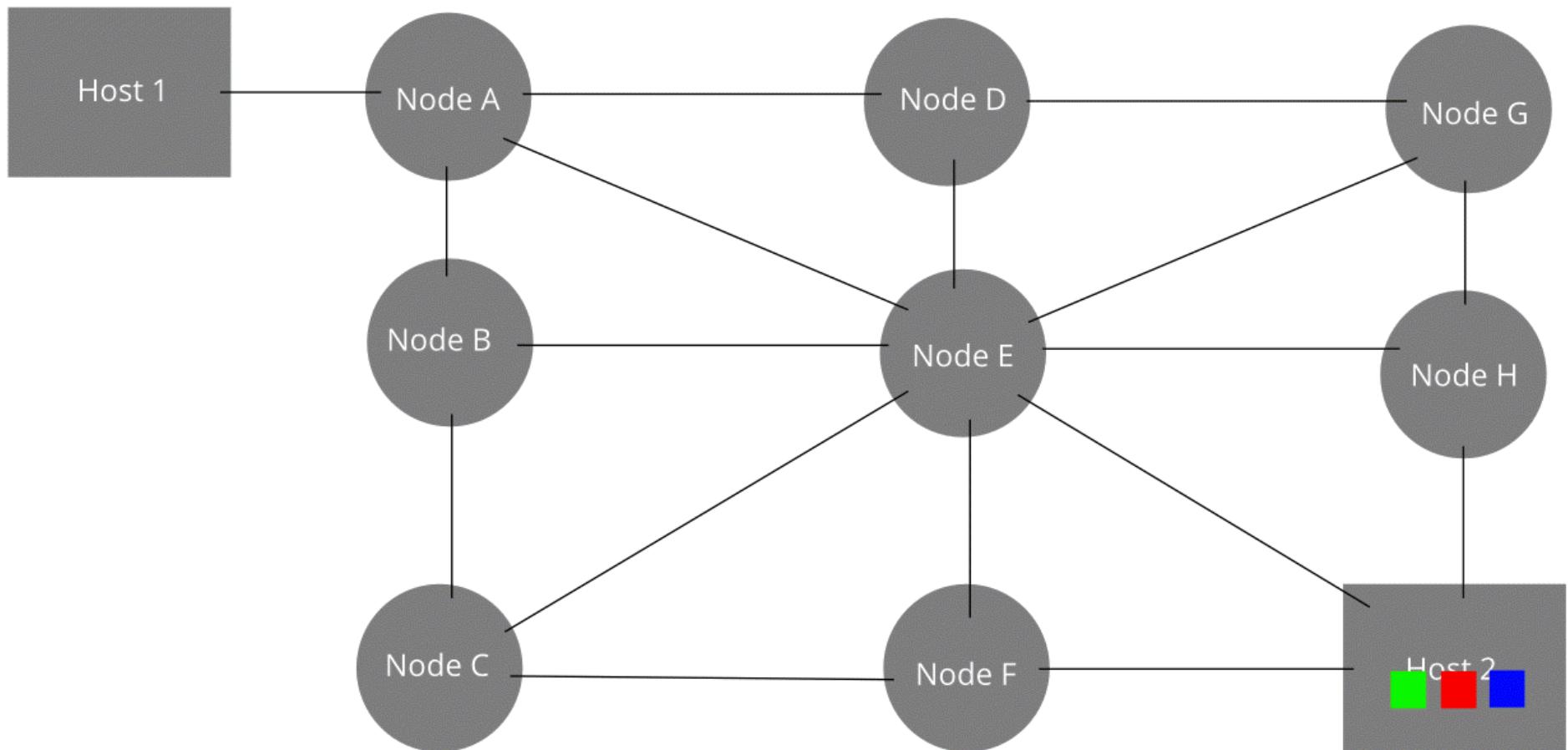
Barrier sync: wait until all packets arrive

The data packets take different routes to their destinations.

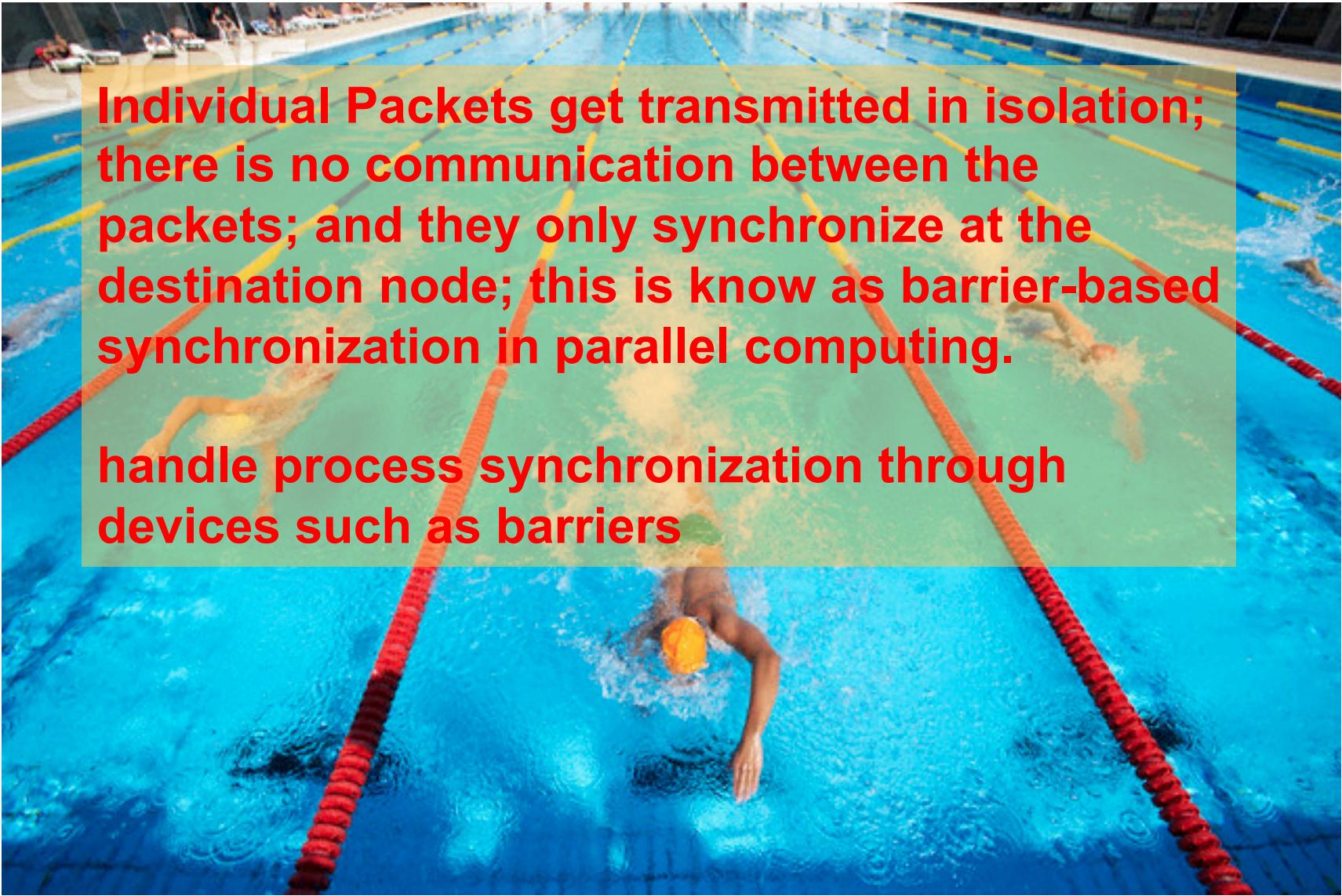


Barrier sync: reassemble payload

The received message is Green, Red, Blue



Individual Packets get transmitted in isolation



Individual Packets get transmitted in isolation; there is no communication between the packets; and they only synchronize at the destination node; this is known as barrier-based synchronization in parallel computing.

handle process synchronization through devices such as barriers

-
- **The key to success here was Divide and Conquer**
 - **Decompose a large task into smaller ones**
 - **We came up with a very nice framework for parallelizing tasks on the command line!**
 - But it is limited
 - Granularity of task is somewhat coarse
 - No fault tolerance
 - No control over shared filespace
 - **Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance**

Issues to be addressed

- ▶ How to break large problem into smaller problems? Decomposition for parallel processing
- ▶ How to assign tasks to workers distributed around the cluster?
- ▶ How do the workers get the data?
- ▶ How to synchronize among the workers?
- ▶ How to communicate with works?
- ▶ How to share partial results among workers?
- ▶ How to do all these in the presence of errors and hardware failures?

Divide and conquer does not come for free: there are obligations in terms of communication, synchronization, and fault tolerance

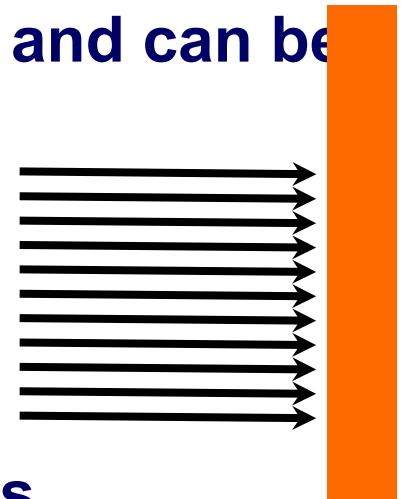
Data Parallelism – Embarrassingly Parallel Tasks

- Little or no effort is required to break up the problem into a number of parallel tasks, and there exists no dependency (or communication) between those parallel tasks.
- Examples:
 - map() function in Python:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> map(lambda e: e*e, x)
>>> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Synchronization thru a Barrier

- A barrier for a group of threads or processes in the source code means any thread/process must stop at this point and cannot proceed until all other threads/processes reach this barrier.
- Another popular way of syncing is the barrier method;
- Explicitly handle process synchronization through devices such as barriers
- it is pretty effective, and very coarse grained and can be great in certain types of problems.
- In parallel computing, a barrier is a type of synchronization method.
- Better than MPI and shared memory solutions



Summary: Communication/Synchronization

- Those problems that can be decomposed into independent subtasks, requiring no communication/synchronization between the subtasks except a join or barrier at the end are very parallelizable (embarrassingly parallel)
 - Mutex pattern
 - Join pattern
 - Barrier pattern

Categories of Parallel Computation Tasks

- Applications are often classified according to how often their subtasks need to synchronize or communicate with each other.
- **Fine-grained parallelism**
 - An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; (share memory programming)
- **Coarse-grained parallelism**
 - It exhibits coarse-grained parallelism if they do not communicate many times per second,
- **Embarrassingly parallel**
 - An application is embarrassingly parallel if it rarely or never has to communicate. Divide and conquer. Summing a list of numbers.
 - Such applications are considered the easiest to parallelize.
 - Can be realized on a shared nothing architecture (see this shortly)

Examples of embarrassingly parallel problems

- An application is embarrassingly parallel if it rarely or never has to communicate
- Applications
 - Summing a list of numbers
 - Matrix multiplication
 - Lots of machine learning algorithms
 - Tree growth step of the random forest machine learning technique.
 - Genetic algorithms and other evolutionary computation metaheuristics.
 - Serving static files on a webserver to multiple users at once.
 - Computer simulations comparing many independent scenarios, such as climate models.
 - Ensemble calculations of numerical weather prediction.
 - Discrete Fourier Transform where each harmonic is independently calculated.
 - Many many more....

Not everything is a MR

- MRs are ideal for “embarrassingly parallel” problems
 - Very little communication
 - Easily distributed
 - Linear computational flow
-
- Happy to report that most of the machine learning algorithms of practical importance are embarrassingly parallel

First generation MapReduce

- **0th Generation**
 - Command line
- **1st Generation**
 - MapReduce, Hadoop
 - Native versus Streaming
- **2nd Generation**
 - MrJob, Scalding
 - Write Map-Reduce pipelines
- **3rd Generation**
 - Spark
 - Memory backed, Rich API (80 calls)

Tutorial Outline

- **Part 1: Introduction**
 - Welcome Survey
 - Install Spark
 - Background and motivation
- **Part 2: Spark Intro and basics**
 - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
 - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
 - Spark 1.5 and beyond
 - Summary

Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistence
- Animated Example
- Pair RDDs
- Word count example

-
- **Traditional distributed data processing frameworks have limited APIs such as Map/Reduce**
 - **Spark is a much more expressive API (over 80 functions)**

Spark: A developer's perspective

- At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster.
- The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.
 - RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it.
 - Users may also ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations.
 - Finally, RDDs automatically recover from node failures.

Resilient distributed dataset (RDD)

- An RDD is simply a distributed collection of elements (Key-Value records).
- In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.

- Under the hood, Spark automatically distributes the data contained in RDDs across your cluster and parallelizes the operations you perform on them.

RDD: Collection of values: Map/Filter/Reduce

Step by Step

RDD: each row is a key, value pair

Focus on simple RDDs in this section, i.e., value only RDDs

Next section talk about RDDs of key/value pairs

Key	Value
d1	the quick brown fox
d2	the fox ate the mouse
d3	how now brown cow

```
graph LR; A[HDFS] --> B[HadoopRDD]; B --> C[FilteredRDD]; C --> D[MappedRDD]; D --> E[HDFS];
```

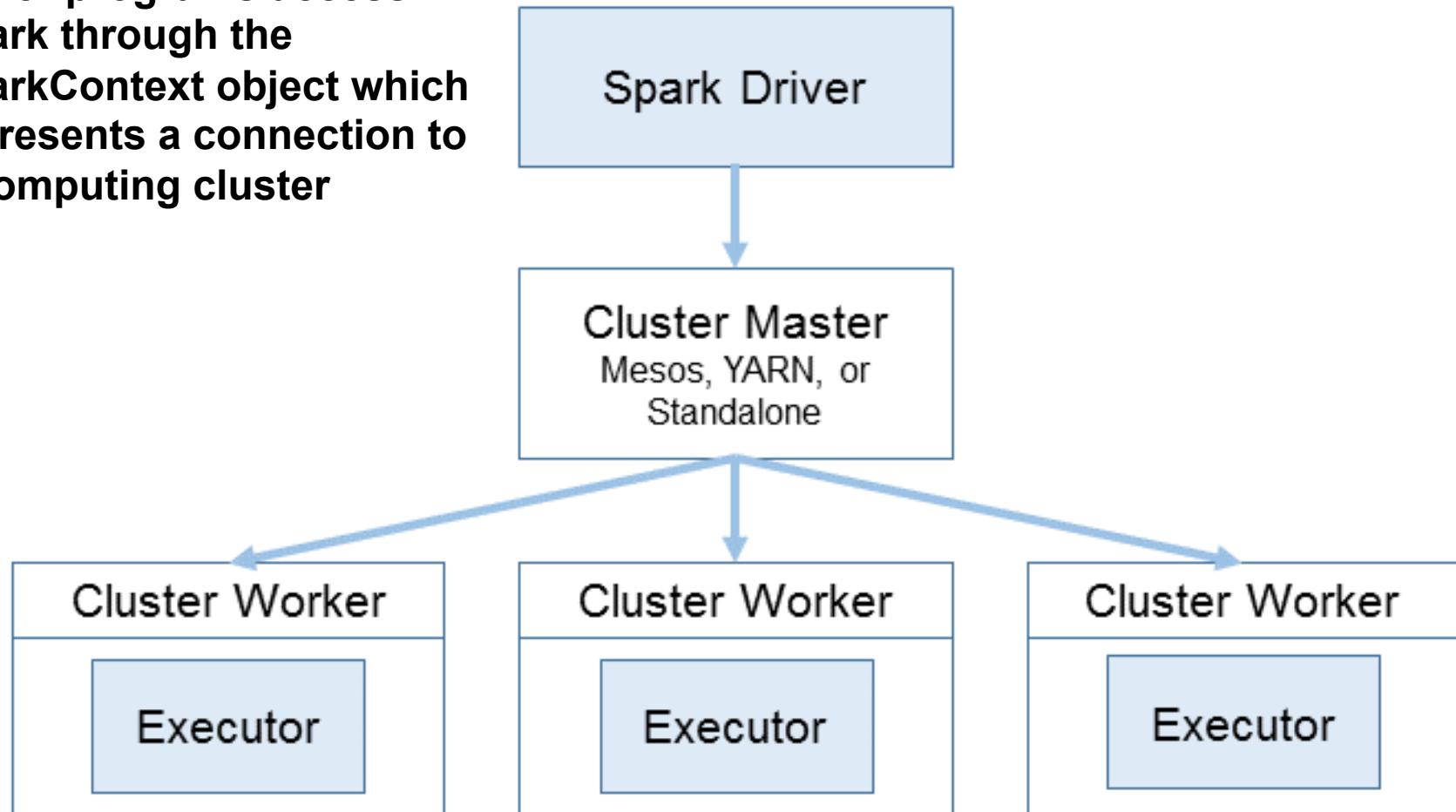
The RDD.saveAsTextFile() action triggers a job. Tasks are started on scheduled executors.

Spark Programming Interface

- **Spark has APIs available in**
 - Scala
 - Java
 - Python
 - R, SQL
- **A lot of Spark's API revolves around passing functions to its operators to run then on the cluster (ships codes to executors)**
- **Provides:**
 - Resilient distributed datasets (RDDs)
 - Partitioned collections with controllable caching, built in fault tolerance
 - Operations on RDDs
 - Transformations (define RDDs), actions (compute results)
 - Restricted shared variables (broadcast, accumulators)
- **Goal: make parallel programs look like local ones**

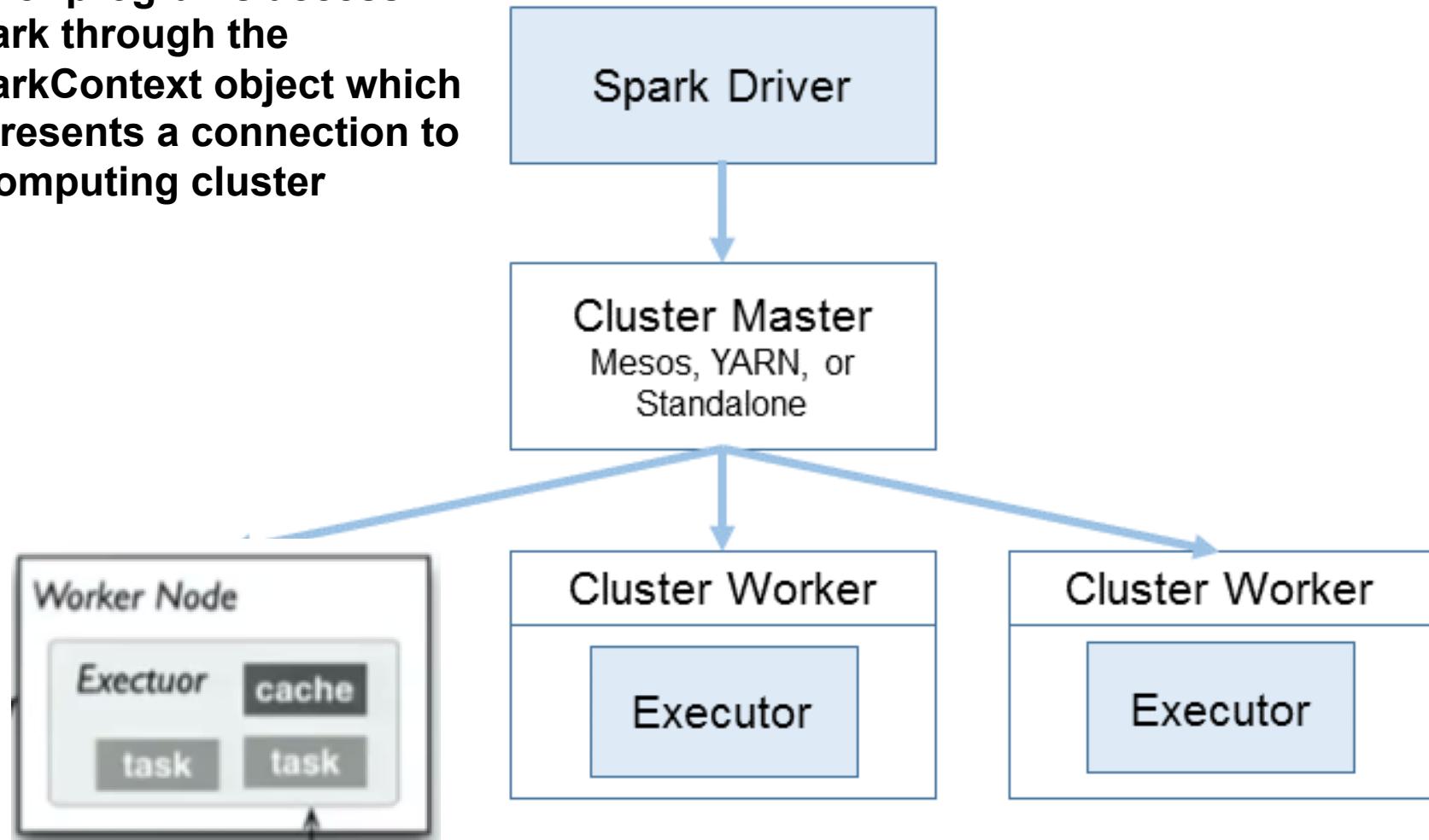
Spark Cluster

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster



Spark Cluster

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster



spark.apache.org/docs/latest/cluster-overview.html

Spark Essentials: Master

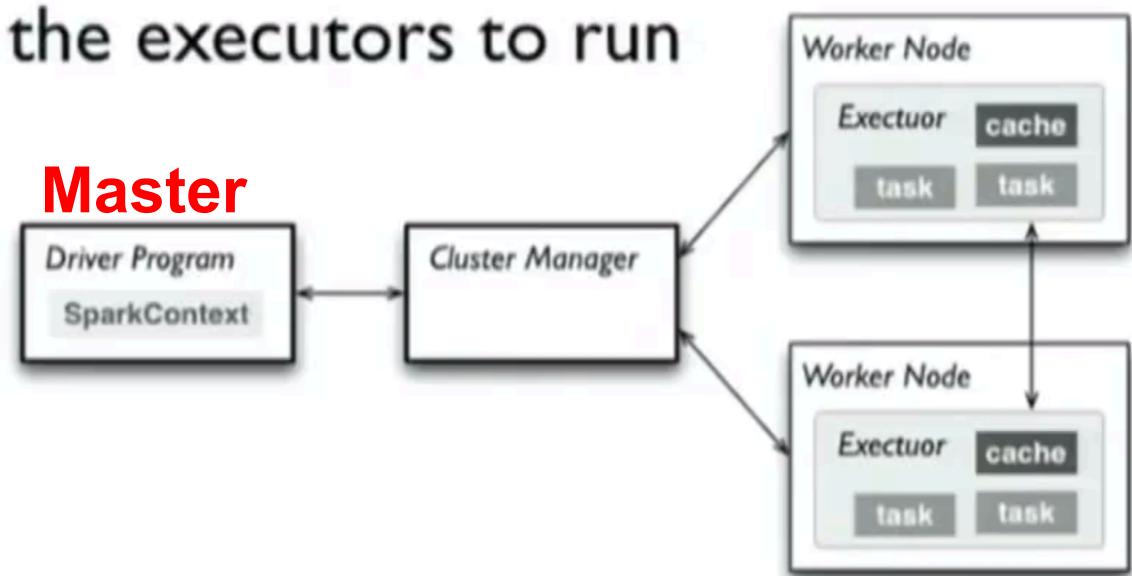
The master parameter for a SparkContext determines which cluster to use

<i>master</i>	<i>description</i>
local	run Spark locally with one worker thread (no parallelism)
local[K]	run Spark locally with K worker threads (ideally set to # cores)
spark://HOST:PORT	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
mesos://HOST:PORT	connect to a Mesos cluster; PORT depends on config (5050 by default)

Spark Essentials: Master

Driver programs access Spark through the `SparkContext` object which represents a connection to a computing cluster

1. connects to a *cluster manager* which allocate resources across applications
2. acquires executors on cluster nodes – worker processes to run computations and store data
3. sends *app code* to the executors **(serializes code and data)**
4. sends *tasks* for the executors to run



Spark Essentials: RDD

Resilient **D**istributed **D**atasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel

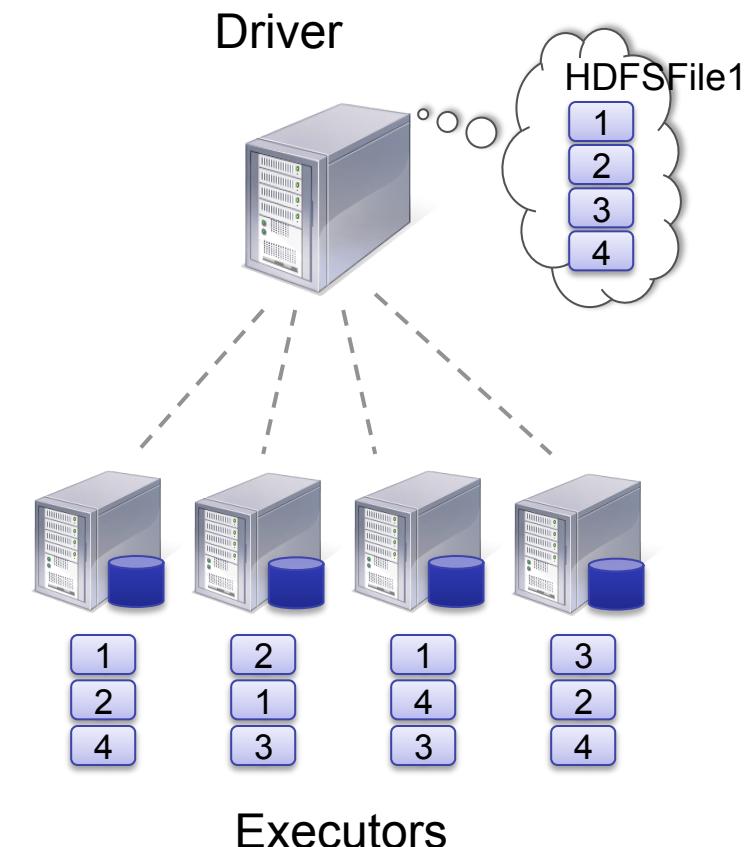
There are currently two types:

- *parallelized collections* – take an existing Scala collection and run functions on it in parallel
- *Hadoop datasets* – run functions on each record of a file in Hadoop distributed file system or any other storage system supported by Hadoop

RDD is a distributed collection of elements

- In Spark all work is expressed as either creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.
- An RDD is laid out across a cluster of machines as collection of Partitions, each including a subset of the data
- The framework processes the objects within a partition in sequence, and processes multiple partitions in parallel.
- Data (e.g., clicks, or record linkage) is stored in a text file, with one observation on each line.
 - JSON, zipped, AVRO, Parquet

Partition and distribute data



Mapper: Create RDD and ship to cluster; then apply mapper (X+1)

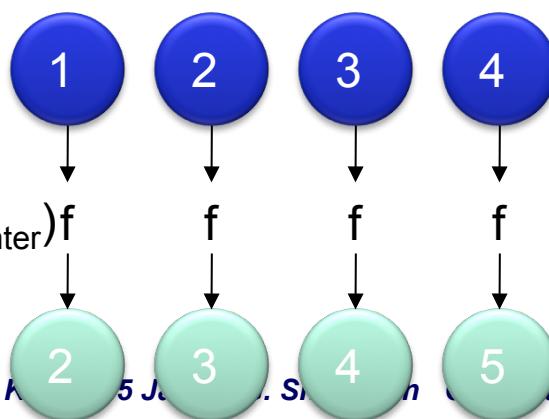
```
In [19]: #RDD mapper      Distribute      Map      Gather back to driver  
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()  
  
#returns [2, 3, 4, 4]  
  
Out[19]: [2, 3, 4, 5]
```

- Spark is oriented around *Key-Value* records

- plusOneData= Apply(Mapper=f(x)=x × 1, input=X)

- Map function:

$\text{Map}(\text{Key}_{\text{in}}, \text{Value}_{\text{in}}) \rightarrow \text{list}(\text{K}_{\text{inter}}, \text{V}_{\text{inter}})$ f $f(x)=x \times 1$



RDD with 4 elements

In [19]: #RDD mapper Distribute Map Gather back to driver
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()
#returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]

In [19]: #RDD mapper
sc.parallelize([1,2,3,4]).map(lambda x: x+1).collect()
#returns [2, 3, 4, 4]

Out[19]: [2, 3, 4, 5]

In [33]: def powerOfTwo(x):
 return x*x

def plusOne(x):
 return x+1

def myReduce(x, y):
 return x+y

print "Reduce by lambda: %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
print "Reduce by function: %d" % sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(myReduce)
#.reduce()

python anonymous function vs. top level function

- Arguments to Map/Reduce operators are closures and can be python anonymous functions (Lambdas) or any top level function

Find the line with most words

RDD actions and transformations can be used for more complex computations. Let's say we want to find the line with the most words:

Scala Python

```
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)  
15
```

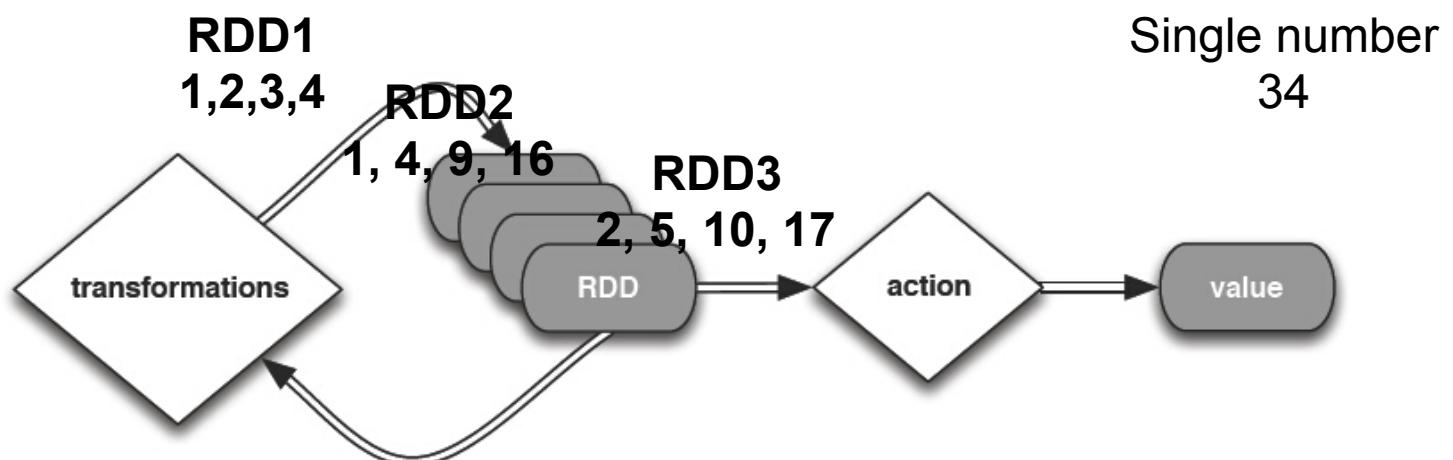
This first maps a line to an integer value, creating a new RDD. reduce is called on that RDD to find the largest line count. The arguments to map and reduce are Python [anonymous functions \(lambdas\)](#), but we can also pass any top-level Python function we want. For example, we'll define a max function to make this code easier to understand:

Spark Essentials: RDD

Spark can create RDDs from any file stored in HDFS or other storage systems supported by Hadoop, e.g., local file system, Amazon S3, Hypertable, HBase, etc.

Spark supports text files, SequenceFiles, and any other Hadoop InputFormat, and can also take a directory or a glob (e.g. /data/201404*)

```
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```



```
# Create an RDD from a local text file
textData = sc.textFile("textData.txt")
# View the contents of the RDD
for line in textData.collect():
    print line
# Lazily filter any lines that contain the word "orange"
orangeLines = textData.filter(lambda line: "orange" in line)
```

```
spark-1.4.0-bin-hadoop2.6 - java - 77x15
>>> for line in textData.collect():
...     print line
...
Hello, My name is Joe.
I live in Oregon.
My favorite color is orange. I
I drive a Chevrolet Silverado.
If I could eat anything, I would eat an orange.
>>> |
```

- ```
for line in orangeLines.collect():
 print line

caps = orangeLines.map(lambda line: line.upper())

for line in caps.collect():
 print line

Key/Value Exercise: WordCount

words = textData.flatMap(lambda line: line.split(" "))

result = words.map(lambda x: (x, 1)).reduceByKey(lambda x, v: x + v)
```

```
spark-1.4.0-bin-hadoop2.6 - java - 77x15
>>> for line in caps.collect():
... print line
...
MY FAVORITE COLOR IS ORANGE.
IF I COULD EAT ANYTHING, I WOULD EAT AN ORANGE.
>>> █
```

```

: from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
: from pyspark.mllib.linalg import Vectors
: from pyspark.mllib.regression import LabeledPoint

- def parseLine(line):
- parts = line.split(',')
- label = float(parts[0])
- features = Vectors.dense([float(x) for x in parts[1].split(' ')])
- return LabeledPoint(label, features)

data = sc.textFile('sample_naive_bayes_data.txt').map(parseLine)

Split data approximately into training (60%) and test (40%)
training, test = data.randomSplit([0.6, 0.4], seed = 0)

Train a naive Bayes model.
model = NaiveBayes.train(training, 1.0)

Make prediction and test accuracy.
predictionAndLabel = test.map(lambda p : (model.predict(p.features), p.label))
accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x == v).count() / test.count()

Save and load model
model.save(sc, "myModelPath")
sameModel = NaiveBayesModel.load(sc, "myModelPath")

```

```

: test.collect()

: [LabeledPoint(0.0, [1.0,0.0,0.0]),
: LabeledPoint(1.0, [0.0,2.0,0.0]),
: LabeledPoint(2.0, [0.0,0.0,2.0])]

: for record in test.collect():
: print record.label, record.features

```

**Large** 0.0 [1.0,0.0,0.0]

# Example Mappers and Reducers

---

```
def powerOfTwo(x):
 return x*x

def plusOne(x):
 return x+1

def myReduce(x, y):
 return x+y

print "Reduce by lambda: %d" %
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(lambda x, y: x + y)
```

```
print "Reduce by function: %d" %
sc.parallelize([1,2,3,4]).map(powerOfTwo).map(plusOne).reduce(myReduce)
#.reduce()
```

Reduce by lambda: 34  
Reduce by function: 34

# Create RDDs

```
./bin/pyspark
```

Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD). RDDs can be created from Hadoop InputFormats (such as HDFS files) or by transforming other RDDs. Let's make a new RDD from the text of the README file in the Spark source directory:

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

RDDs have *actions*, which return values, and *transformations*, which return pointers to new RDDs. Let's start with a few actions:

```
>>> textFile.count() # Number of items in this RDD
126

>>> textFile.first() # First item in this RDD
u'# Apache Spark'
```

Now let's use a transformation. We will use the *filter* transformation to return a new RDD with a subset of the items in the file.

```
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

We can chain together transformations and actions:

```
>>> textFile.filter(lambda line: "Spark" in line).count() # How many lines contain "Spark"?
15
```

# Self-Contained Applications in Python (4 threads)

As an example, we'll create a simple Spark application, SimpleApp.py:

```
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "YOUR_SPARK_HOME/README.md" # Should be some file on your system
sc = SparkContext("local", "Simple App")
logData = sc.textFile(logFile).cache()

numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()

print "Lines with a: %i, lines with b: %i" % (numAs, numBs)
```

This program just counts the number of lines containing ‘a’ and the number containing ‘b’ in a text file. Note that you’ll need to replace YOUR\_SPARK\_HOME with the location where Spark is installed. As with the Scala and Java examples, we use a SparkContext to create RDDs. We can pass Python functions to Spark, which are automatically serialized along with any variables that they reference. For applications that use custom classes or third-party libraries, we can also add code dependencies to spark-submit through its --py-files argument by packaging them into a .zip file (see spark-submit --help for details). SimpleApp is simple enough that we do not need to specify any code dependencies.

We can run this application using the bin/spark-submit script:

```
Use spark-submit to run your application
$ YOUR_SPARK_HOME/bin/spark-submit \
--master local[4] \
SimpleApp.py
...
Lines with a: 46, Lines with b: 23
```

# RDDs in More Detail

---

- An RDD is an immutable, partitioned, logical collection of records
  - Need not be materialized, but rather contains information to rebuild a dataset from stable storage or computer instructions (e.g., generate 1M random numbers)
  - **Materialize RDDs** through actions, e.g., count or reduce
- Partitioning can be based on a key in each record (using hash or range partitioning)
- Built using bulk transformations on other RDDs
- Can be cached for future reuse (or rebuilt if not cached and required again)

# Two types of RDD Operations

---

- **TRANSFORMATIONS:** Think Map (take an RDD as input and produce an RDD); LAZY
- **ACTIONS:** E.g., Reduce.
  - take an RDD as input and
  - return a result to the driver or write it to storage
  - AND kick off a computation
- A transformed RDD gets (re)computed when an action is run on it
- An RDD can be persisted into memory or disk

# Example RDD creation from an existing collection in your program

Scala:

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)
```

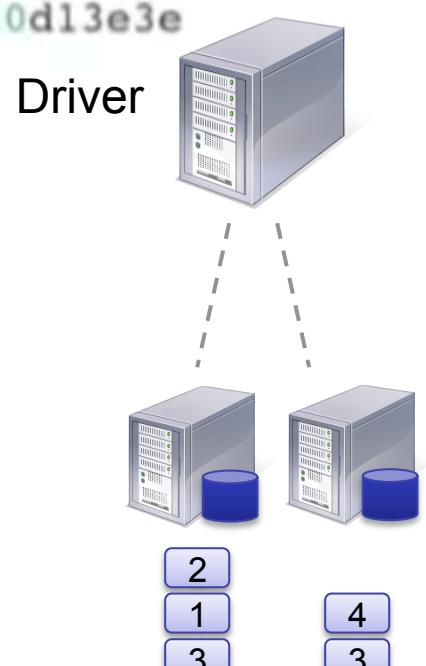
```
scala> val distData = sc.parallelize(data)
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

Python:

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]
```

```
>>> distData = sc.parallelize(data)
>>> distData
```

```
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```



# Example RDD creation from external data

---

Scala:

```
scala> val distFile = sc.textFile("README.md")
distFile: spark.RDD[String] = spark.HadoopRDD@1d4cee08
```

Python:

```
>>> distFile = sc.textFile("README.md")
14/04/19 23:42:40 INFO storage.MemoryStore: ensureFreeSpace(36827) called
with curMem=0, maxMem=318111744
14/04/19 23:42:40 INFO storage.MemoryStore: Block broadcast_0 stored as
values to memory (estimated size 36.0 KB, free 303.3 MB)
>>> distFile
MappedRDD[2] at textFile at NativeMethodAccessorImpl.java:-2
```

# Mapper: Apply tokenize to each input record

tokenize("coffee panda") = List("coffee", "panda")

RDD1  
{"coffee panda", "happy panda",  
"happiest panda party"}

rdd1.map(tokenize)

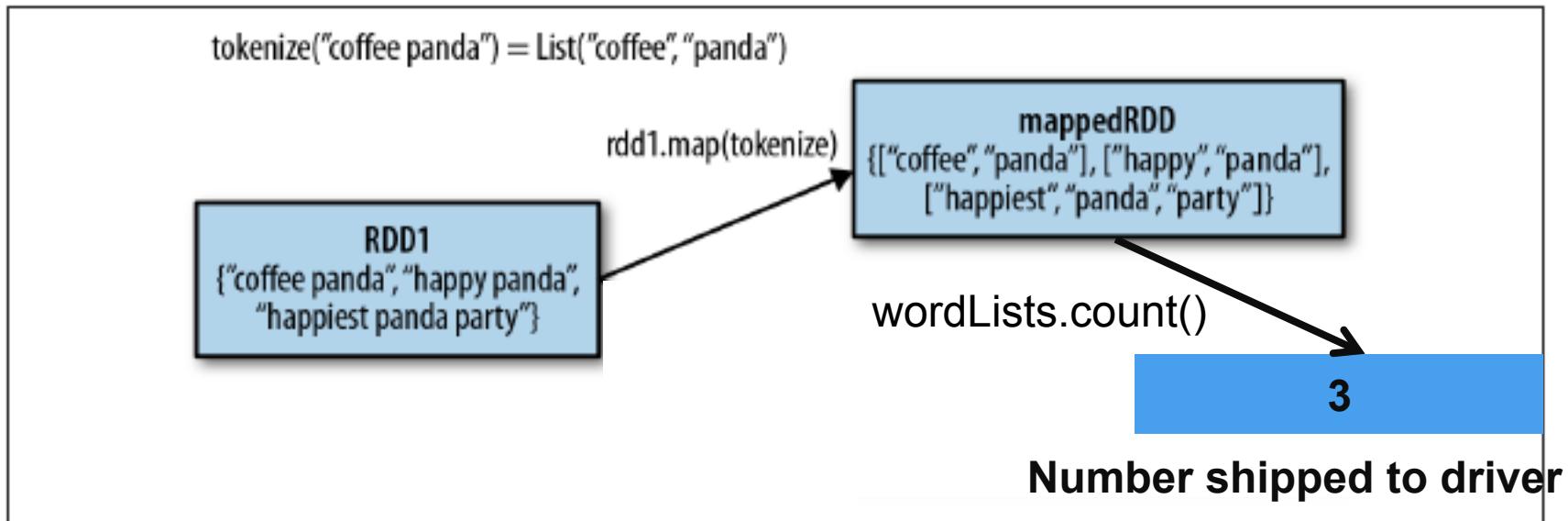
mappedRDD  
[["coffee", "panda"], ["happy", "panda"],  
["happiest", "panda", "party"]]

```
def tokenize(line):
 return(line.split(" "))
```

```
rdd1= sc.parallelize(["coffee panda", "happy panda", "happiest panda party"])
words = rdd.map(tokenize)
```

Framework takes care of passing the mapper's function to the executors (task nodes)

# Built in reducer for counting (action)



```
def tokenize(line):
 return(line.split(" "))
```

```
rdd1= sc.parallelize(["coffee panda", "happy panda", "happiest panda party"])
wordLists = rdd1.map(tokenize)
wordLists.count()
```

The two most common transformations you will likely be using are `map()` and `filter()` (see Figure 3-2). The `map()` transformation takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD. The `filter()` transformation takes in a function and returns an RDD that only has elements that pass the `filter()` function.

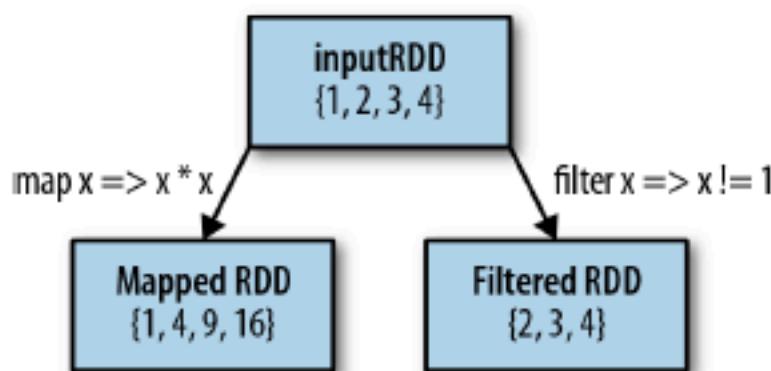


Figure 3-2. Mapped and filtered RDD from an input RDD

We can use `map()` to do any number of things, from fetching the website associated with each URL in our collection to just squaring the numbers. It is useful to note that `map()`'s return type does not have to be the same as its input type, so if we had an RDD `String` and our `map()` function were to parse the strings and return a `Double`, our input RDD type would be `RDD[String]` and the resulting RDD type would be `RDD[Double]`.

# Lazy Evaluation

---

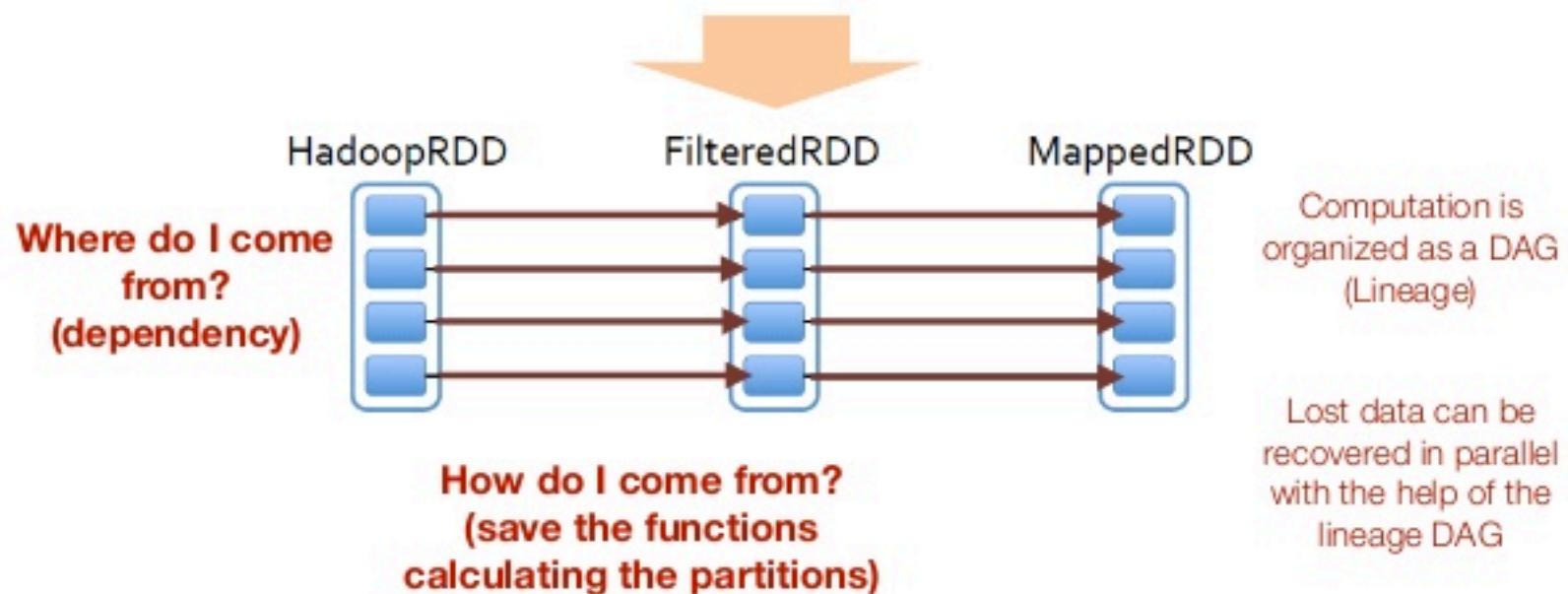
- **Transformations and actions are different because of the way Spark computes RDDs.**
- **Although you can define new RDDs any time, Spark computes them only in a lazy fashion—that is, the first time they are used in an action.**
  - `lines = sc.textFile("exampleFile")` #if not lazy would read whole file in
- **Versus**
  - `lines = sc.textFile("exampleFile").first()`

# Computation is organized as a DAG (Lineage/Recipe): resiliency

From data to computation

- Lineage

```
errorMessages= sc.parallelize.filter(lambda x: 'ERROR' in x) \
 .map(lambda line: line.split(" ")[1]) #second word
```



```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

# RDD: Lineage Graph

`union()` is a bit different than `filter()`, in that it operates on two RDDs instead of one. Transformations can actually operate on any number of input RDDs.



A better way to accomplish the same result as in [Example 3-14](#) would be to simply filter the `inputRDD` once, looking for either *error* or *warning*.

**RDD: Spark keeps track of the set of dependancies between different RDDs using Lineage Graph**

Finally, as you derive new RDDs from each other using transformations, Spark keeps track of the set of dependencies between different RDDs, called the *lineage graph*. It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost. [Figure 3-1](#) shows a lineage graph for [Example 3-14](#).

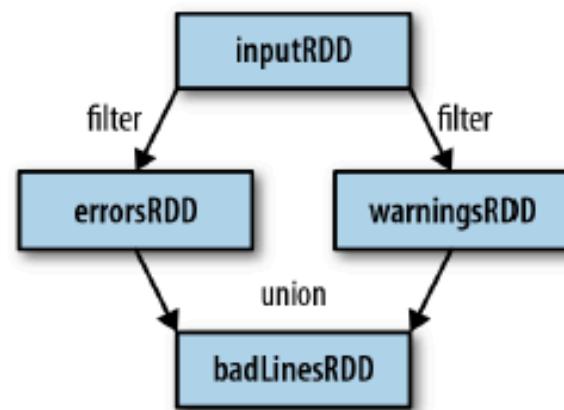


Figure 3-1. RDD lineage graph created during log analysis

# counts.toDebugString() #lineage

Slide 1/2

```
In [5]: lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
counts = lines.flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda a, b: a + b)

counts.collect()
```

```
Out[5]: [('1', 1),
 ('line', 5),
 ('Mining', 2),
 ('3', 1),
 ('2', 1),
 ('data', 1),
 ('5', 1),
 ('Data', 3),
 ('4', 1)]
```

```
In []: counts.to
counts.toDF
In [2]: counts.toDebugString and with NO stochasticity!
counts.toLocalIterator
counts.top + 1/m Σi(1 - yi(w'xi - b))+
A # gradient
```

# counts.toDebugString() #lineage/recipe

```
In [5]: lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
counts = lines.flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda a, b: a + b)

counts.collect()
```

Slide 2/2

```
Out[5]: [('1', 1),
 ('line', 5),
 ('Mining', 2),
 ('3', 1),
 ('2', 1),
 ('data', 1),
 ('5', 1),
 ('Data', 3),
 ('4', 1)]
```

```
In [7]: counts.toDebugString()
```

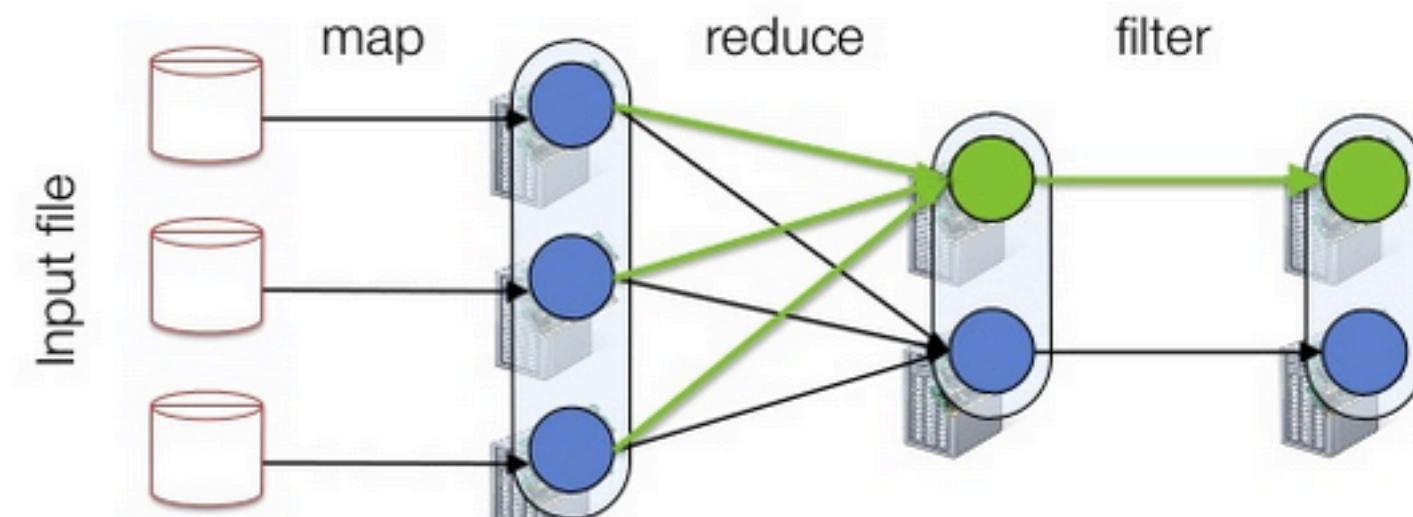
```
Out[7]: '(8) PythonRDD[7] at collect at <ipython-input-5-708c6b1f0496>:4 []\n| MapPartitionsRDD[6] at mapPartitions at PythonRDD.scala:346 []\n| ShuffledRDD[5] at partitionBy at NativeMethodAccessorImpl.java:-2 []\n+-(8) PairwiseRDD[4] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| PythonRDD[3] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:396 []'
```

```
PythonRDD[7] at collect at <ipython-input-5-708c6b1f0496>:4 []\n| MapPartitionsRDD[6] at mapPartitions at PythonRDD.scala:346 []\n| ShuffledRDD[5] at partitionBy at NativeMethodAccessorImpl.java:-2 []\n+-(8) PairwiseRDD[4] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| PythonRDD[3] at reduceByKey at <ipython-input-5-708c6b1f0496>:2 []\n| ParallelCollectionRDD[2] at parallelize at PythonRDD.scala:396 []'
```

# Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```
file.map(lambda rec: (rec.type, 1))
 .reduceByKey(lambda x, y: x + y)
 .filter(lambda (type, count): count > 10)
```

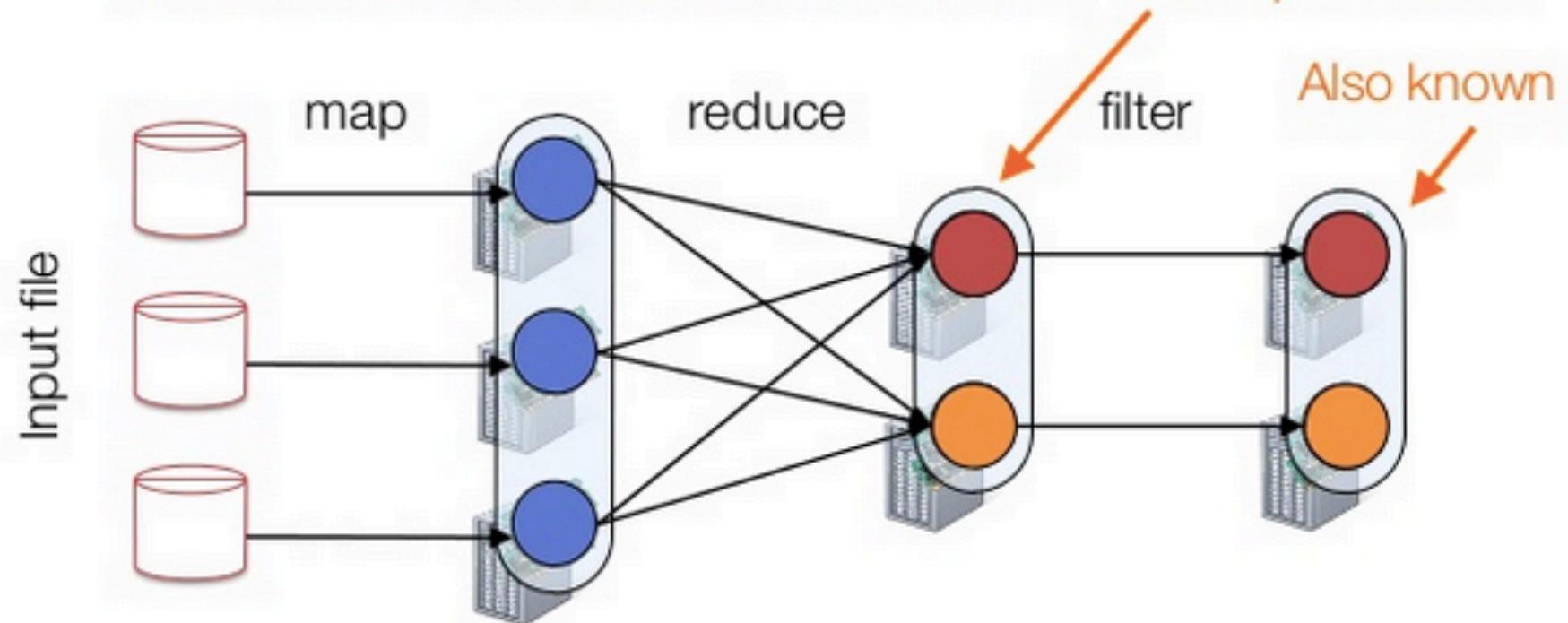


# RDDs know their partitioning function

- ..

```
file.map(lambda rec: (rec.type, 1))
 .reduceByKey(lambda x, y: x + y)
 .filter(lambda (type, count): count > 10)
```

Known to be  
hash-partitioned



# In summary

---

- That was a look at our first Spark program
- Just used very basic map and reduce (count) operations over the distributed key-value store (RDD)
- We explore more operations next and go a little deeper

# Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistence
- Animated Example
- Pair RDDs
- Word count example

# This section

---

- Just used very basic map and reduce (count) operations over the distributed key-value store (RDD)
- We explore more operations next and go a little deeper:
  - Work on base RDDs and how the Spark framework works
  - Write some code

# Spark Runtime

- 

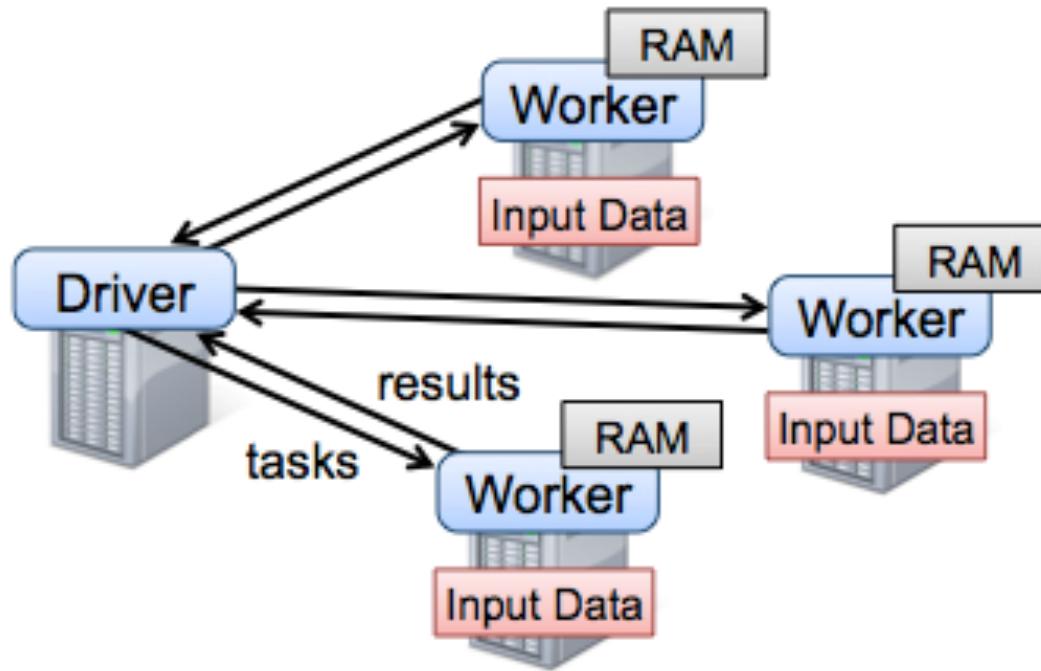


Figure 2: Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

# Divide and conquer with Closures

---

- **Decompose problems in non-overlapping subproblems**
- **Spark's API relies heavily on passing functions in the driver program to run on the cluster. There are three recommended ways to do this:**
  - Lambda expressions, for simple functions that can be written as an expression. (Lambdas do not support multi-statement functions or statements that do not return a value.)
  - Local defs inside the function calling into Spark, for longer code.
  - Top-level functions in a module.
- **Lazy evaluation! Optimize execution graphs**

# Driver vs Workers

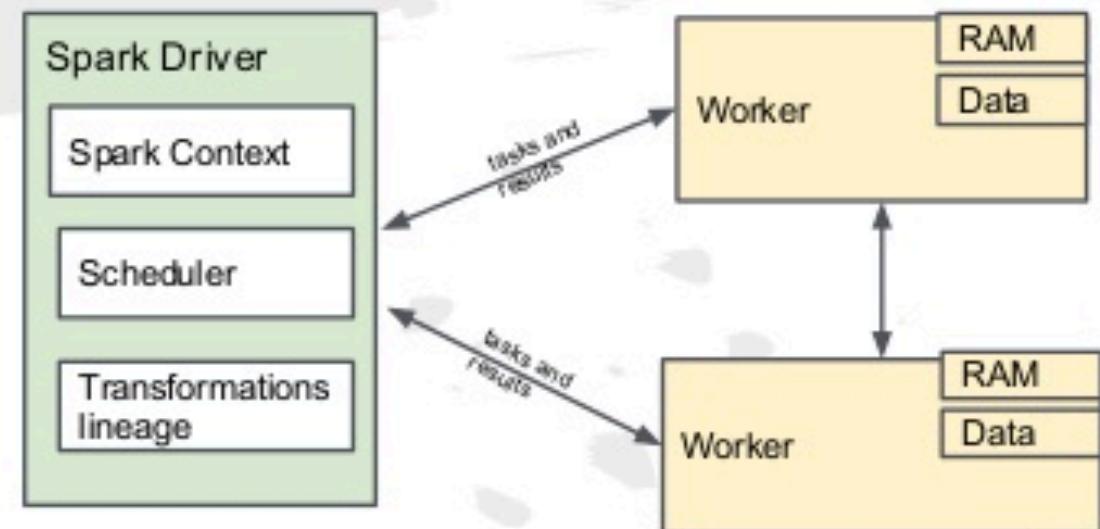
## At the core of Spark...

### Spark Driver

- Define & launch operations on RDDs
- Keeps track of RDD lineage allowing to recover lost partitions
- Use the RDD DAG in the Scheduler to define the stages and tasks to be executed

### Workers

- Read/Transform/Write RDD partitions
- Need to communicate with other workers to share their cache and shuffle data



# Closure (computer programming)

From Wikipedia, the free encyclopedia

# Closure

*For other uses of this term, including in mathematics and computer science, see [Closure](#).*

*Not to be confused with [Clojure](#).*

In programming languages, **closures** (also **lexical closures** or **function closures**) are a technique for implementing lexically scoped name binding in languages with [first-class functions](#). Operationally, a closure is a data structure storing a function<sup>[a]</sup> together with an environment:<sup>[1]</sup> a mapping associating each free variable (variables that are used locally, but defined in an enclosing scope) with the value or storage location the function to access those captured variables later.

**Example** The following program defines a function `startAt` that returns a function `incrementBy`. The nested function `incrementBy` adds its argument to the value of `x`, though `x` is not local to `incrementBy`. Instead, it finds `x` in the environment where `startAt` was defined, and thus has the value 1. When `incrementBy` is called, it adds its argument to 1, and returns the result. This means that `closure1` will return 4 when invoked with 3, and `closure2` will return 8 when invoked with 3. Both `closure1` and `closure2` have the same function `incrementBy`, but they have different environments, and thus evaluate the function differently.

**A closure is a data structure storing a function[a] together with an environment:**

- a mapping associating each free variable of the function (variables that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created.**

```
function startAt(x)
 function incrementBy(y)
 return x + y
 return incrementBy

variable closure1 = startAt(1)
variable closure2 = startAt(5)
```

Note that, as `startAt` returns a function, the variables `closure1` and `closure2` are of [function type](#). Invoking `closure1(3)` will return 4, while invoking `closure2(3)` will return 8. While `closure1` and `closure2` have the same function `incrementBy`, the associated environments differ, and invoking the closures will bind the name `x` to two distinct variables in the two invocations, with different values, thus evaluating the function to different results.

## Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```



Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

## Spark Essentials: Transformations

Scala:

```
val distFile = sc.textFile("README.md")
distFile.map(l => l.split(" ")).collect()
distFile.flatMap(l => l.split(" ")).collect()
```

closures

Python:

```
distFile = sc.textFile("README.md")
distFile.map(lambda x: x.split(' ')).collect()
distFile.flatMap(lambda x: x.split(' ')).collect()
```

*looking at the output, how would you  
compare results for map() vs. flatMap() ?*

# A Hello World Example: Summary stats of a large random dataset

---

- Example from scratch
- Generate a random array of numbers and put them into an RDD
- Transform them by doubling each one
- Filter all numbers  $> 1$
- RDD distributes the data (but not immediately, it waits, it is lazy!!)
- Cache the RDD in memory [still lazy]
- Count the number of numbers  $< 1$  (should be 0)
- Count the number of numbers  $> 1$
- Count the number of numbers  $> 1$  (should be the same as the previous result)

# SparkContext:

---

- A connection to a computing cluster
- Through SparkContext, driver program can access Spark.
- Automatically created in interactive shell
- You can also create your own SparkContext

```
from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("App")
sc = SparkContext(conf = conf)
```

File Edit View Insert Cell Kernel Help

In [1]:

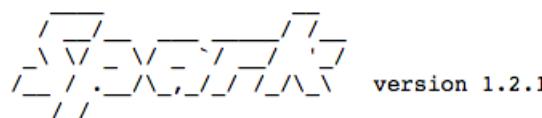
```
import os
import sys

#set up Spark and give us a spark context sc
spark_home = os.environ['SPARK_HOME']='/Users/jshanahan/Software/spark-1.2.1-bin-hadoop2.4/' #desktop

print "["+ spark_home+"]"
if not spark_home:
 raise ValueError('SPARK_HOME environment variable is not set')

sys.path.insert(0, os.path.join(spark_home, 'python'))
sys.path.insert(0, os.path.join(spark_home, 'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home, 'python/pyspark/shell.py'))
```

[/Users/jshanahan/Software/spark-1.2.1-bin-hadoop2.4/]  
Welcome to



version 1.2.1

Using Python version 2.7.8 (default, Aug 21 2014 15:21:46)  
SparkContext available as sc.

See next slide for detailed breakdown

In [29]:

```
1 import numpy as np
2 import pylab
3
4 dataRDD = sc.parallelize(np.random.random_sample(1000))
5 data2X= dataRDD.map(lambda x: x*2)
6 dataGreaterThan1 = data2X.filter(lambda x: x > 1.0)
7 cachedRDD = dataGreaterThan1.cache()
```

In [29]:

```
1 cachedRDD.filter(lambda x: x<1).count()
```

Out[29]: 0

In [31]:

```
1 cachedRDD.filter(lambda x: x>1).count()
```

Out[31]: 514

In [32]:

```
1 cachedRDD.filter(lambda x: x>1).count()
```

Out[32]: 514

Large Scale

---

```
In [28]: 1 import numpy as np
 2 import pylab
 3
 4 dataRDD = sc.parallelize(np.random.random_sample(1000))
 5 data2X= dataRDD.map(lambda x: x*x)
 6 dataGreaterThan1 = data2X.filter(lambda x: x > 1.0)
 7 cachedRDD = dataGreaterThan1.cache()
```

```
In [29]: 1 cachedRDD.filter(lambda x: x<1).count()
```

```
Out[29]: 0
```

```
In [31]: 1 cachedRDD.filter(lambda x: x>1).count()
```

```
Out[31]: 514
```

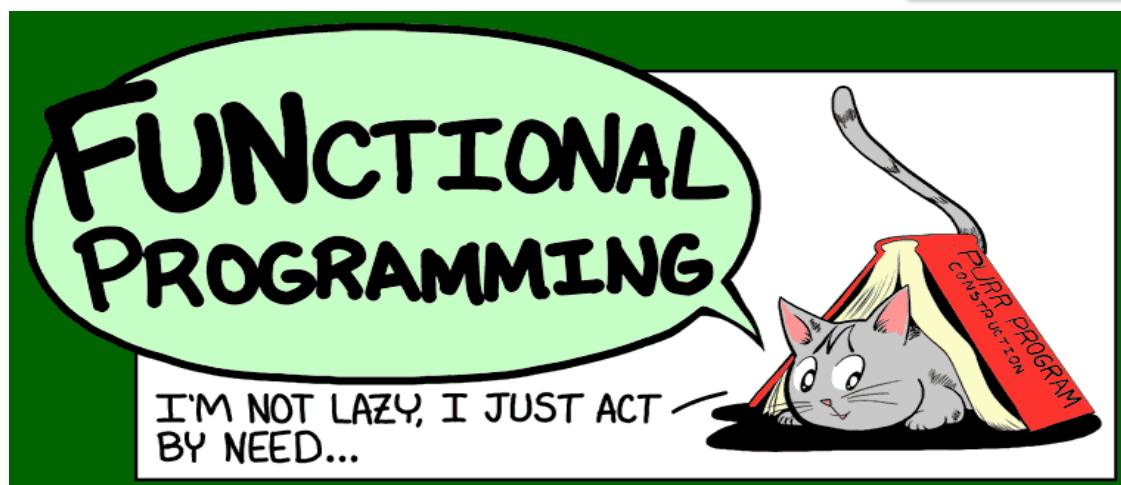
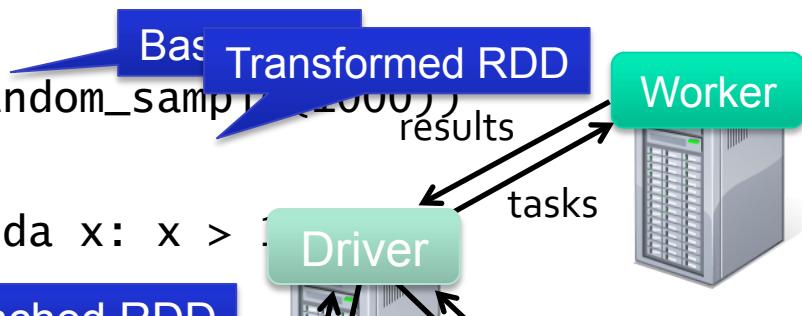
```
In [32]: 1 cachedRDD.filter(lambda x: x>1).count()
```

```
Out[32]: 514
```

# Example: Random Numbers

- SC: Spark Context (connection to cluster driver)
- Load a bunch random numbers into an RDD
- Spark is Lazy...

```
dataRDD = sc.parallelize(np.random.random_sample(1000))
data2x= dataRDD.map(lambda x: x*2)
dataGreaterThan1 = data2x.filter(lambda x: x > 1)
cachedRDD = dataGreaterThan1.cache()
```



# Two types RDD Operations

## Transformations (define a new RDD)

- map
- filter
- sample
- union
- groupByKey
- reduceByKey
- join
- cache
- ...

**Nothing is materialized**

## Parallel operations (Actions) (return a result to driver)

- reduce
- collect
- count
- save
- lookupKey
- ...

---

# Coffee break

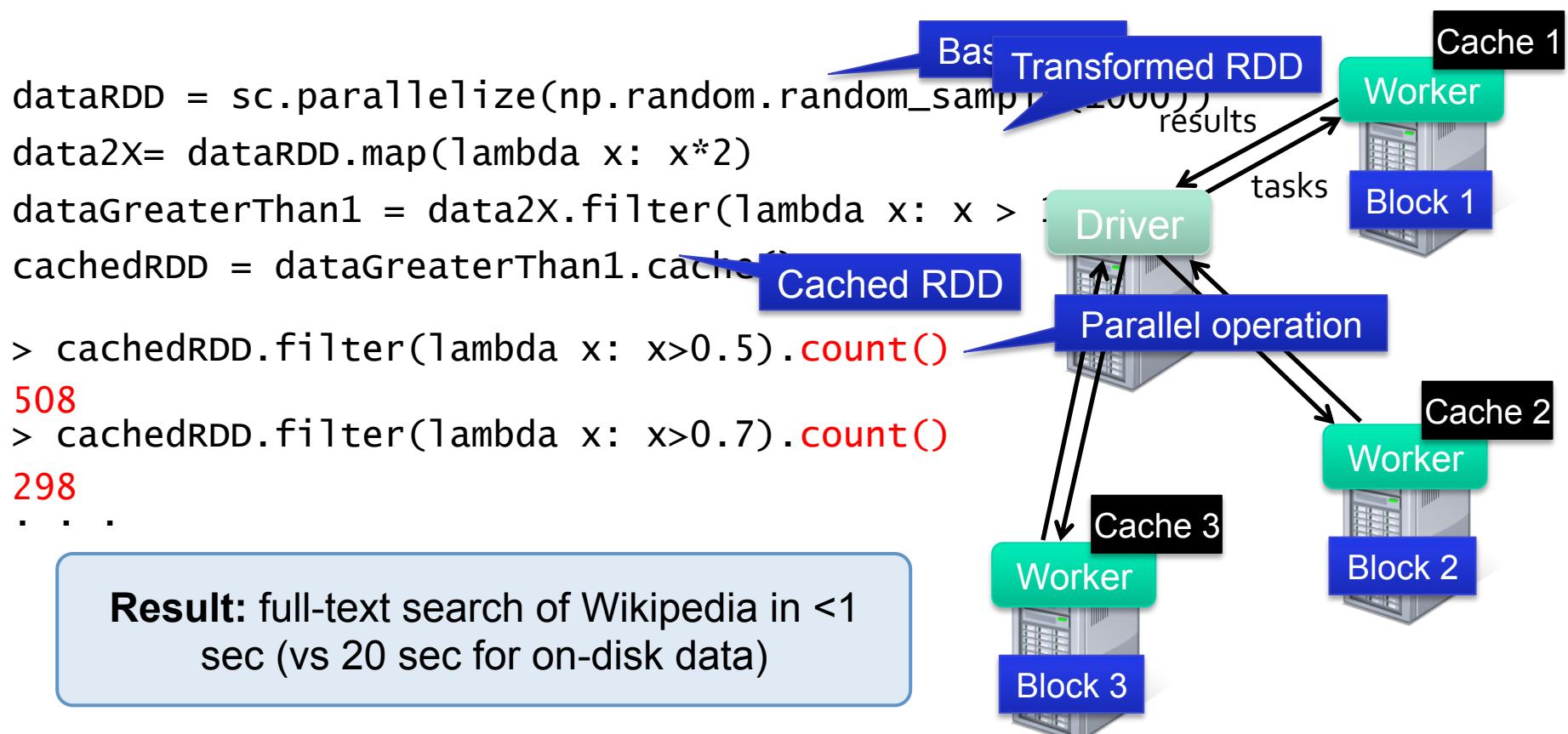
# Example: Random Numbers

- Load a bunch random numbers into an RDD



# Example: Random Numbers

- Load a bunch random numbers into an RDD
- Transform, filter, and the count (action)



# Recompute vs Cache

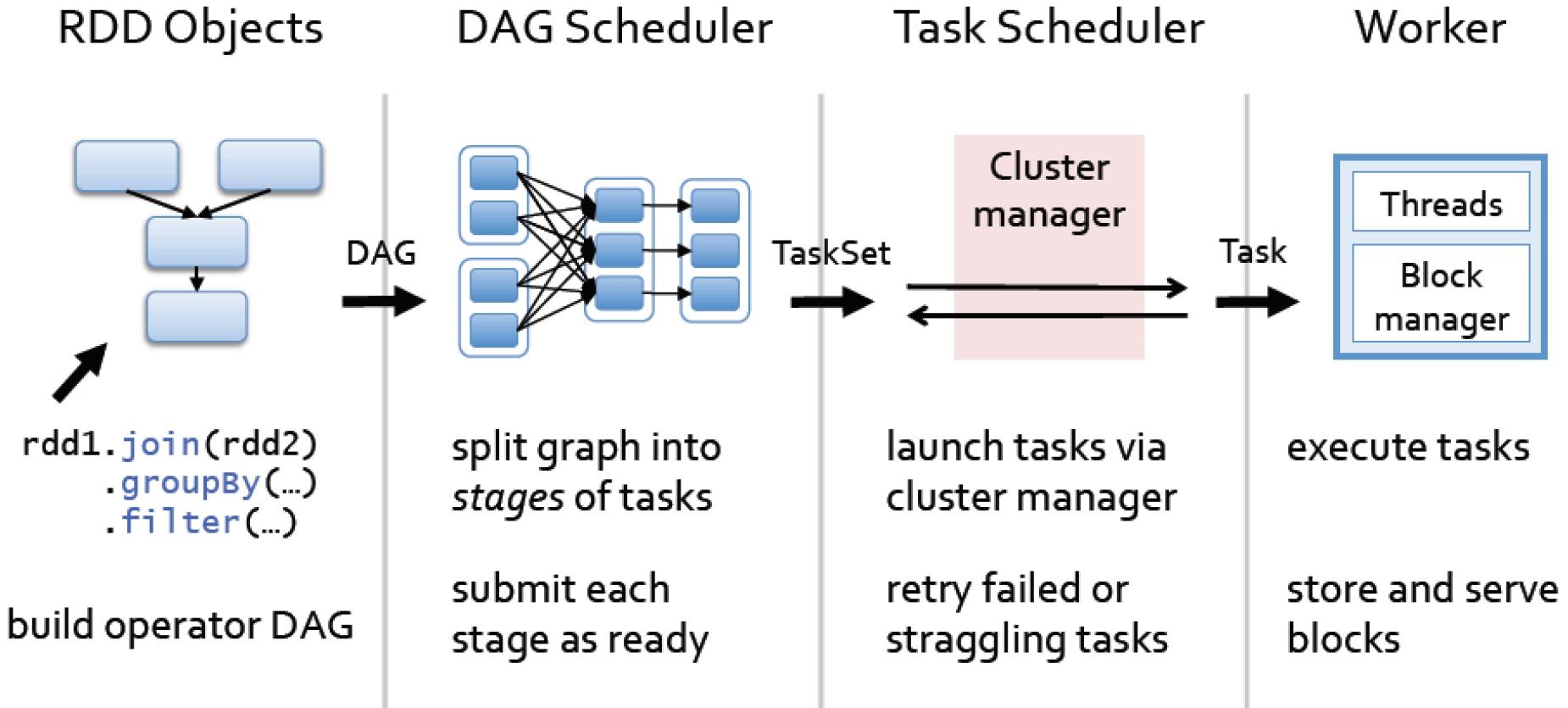
---

- Spark's RDDs are by default recomputed each time you run an action on them.
- If you would like to reuse an RDD in multiple actions, you can ask Spark to persist it using `RDD.persist()`.

# Shipping code to the cluster

Optimized steps into stages

## RDD → Stages → Tasks



# Resilient Distributed Datasets, or RDDs

---

- 
- 

The `SparkContext` has a long list of methods, but the ones that we're going to use most often allow us to create *Resilient Distributed Datasets*, or *RDDs*. An RDD is Spark's fundamental abstraction for representing a collection of objects that can be distributed across multiple machines in a cluster. There are two ways to create an RDD in Spark:

- Using the `SparkContext` to create an RDD from an external data source, like a file in HDFS, a database table via JDBC, or from a local collection of objects that we create in the Spark shell.
- Performing a transformation on one or more existing RDDs, like filtering records, aggregating records by a common key, or joining multiple RDDs together.

RDDs are a convenient way to describe the computations that we want to perform on our data as a sequence of small, independent steps.

# RDD: parallelize, textFile

## Resilient Distributed Datasets

An RDD is laid out across the cluster of machines as a collection of *partitions*, each including a subset of the data. Partitions define the unit of parallelism in Spark. The framework processes the objects within a partition in sequence, and processes multiple partitions in parallel. One of the simplest ways to create an RDD is to use the `parallelize` method on `SparkContext` with a local collection of objects:

```
val rdd = sc.parallelize(Array(1, 2, 2, 4), 4) Number of partitions
...
rdd: org.apache.spark.rdd.RDD[Int] = ...
```

The first argument is the collection of objects to parallelize. The second is the number of partitions. When the time comes to compute the objects within a partition, Spark fetches a subset of the collection from the driver process.

To create an RDD from a text file or directory of text files residing in a distributed file system like HDFS, we can pass the name of the file or directory to the `textFile` method:

```
val rdd2 = sc.textFile("hdfs://some/path.txt") Create and RDD from a
file or from a directory
...
rdd2: org.apache.spark.rdd.RDD[String] = ...
```

---

- Transformations

## Spark Essentials: Transformations

| <i>transformation</i>                                | <i>description</i>                                                                                                                                                          |
|------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>map(func)</code>                               | return a new distributed dataset formed by passing each element of the source through a function <code>func</code>                                                          |
| <code>filter(func)</code>                            | return a new dataset formed by selecting those elements of the source on which <code>func</code> returns true                                                               |
| <code>flatMap(func)</code>                           | similar to <code>map</code> , but each input item can be mapped to 0 or more output items (so <code>func</code> should return a <code>Seq</code> rather than a single item) |
| <code>sample(withReplacement, fraction, seed)</code> | sample a fraction <code>fraction</code> of the data, with or without replacement, using a given random number generator <code>seed</code>                                   |
| <code>union(otherDataset)</code>                     | return a new dataset that contains the union of the elements in the source dataset and the argument                                                                         |
| <code>distinct([numTasks]))</code>                   | return a new dataset that contains the distinct elements of the source dataset                                                                                              |

## Spark Essentials: Transformations

| transformation                            | description                                                                                                                                                                                            |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>groupByKey([numTasks])</b>             | when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs                                                                                                                       |
| <b>reduceByKey(func, [numTasks])</b>      | when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function                                               |
| <b>sortByKey([ascending], [numTasks])</b> | when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument |
| <b>join(otherDataset, [numTasks])</b>     | when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key                                                                      |
| <b>cogroup(otherDataset, [numTasks])</b>  | when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith                                                                             |
| <b>cartesian(otherDataset)</b>            | when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)                                                                                                    |

# Transformations vs Actions

## Transformation: RDD → RDD

Once created, RDDs offer two types of operations: *transformations* and *actions*. *Transformations* construct a new RDD from a previous one. For example, one common transformation is filtering data that matches a predicate. In our text file example, we can use this to create a new RDD holding just the strings that contain the word *Python*, as shown in Example 3-2.

*Example 3-2. Calling the filter() transformation*

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

## Actions: RDD → result is given to the driver or saved to external storage

*Actions*, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS). One example of an action we called earlier is `first()`, which returns the first element in an RDD and is demonstrated in Example 3-3.

*Example 3-3. Calling the first() action*

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

## Spark Essentials: Actions

| action                                             | description                                                                                                                                                                                                     |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>reduce(func)</b>                                | aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel |
| <b>collect()</b>                                   | return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data                                |
| <b>count()</b>                                     | return the number of elements in the dataset                                                                                                                                                                    |
| <b>first()</b>                                     | return the first element of the dataset – similar to <i>take(1)</i>                                                                                                                                             |
| <b>take(n)</b>                                     | return an array with the first <i>n</i> elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements                                                      |
| <b>takeSample(withReplacement, fraction, seed)</b> | return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, using the given random number generator seed                                                           |

## Spark Essentials: Actions

| action                          | description                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>saveAsTextFile(path)</b>     | write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file                                                                                                                                                                   |
| <b>saveAsSequenceFile(path)</b> | write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's <code>Writable</code> interface or are implicitly convertible to <code>Writable</code> (Spark includes conversions for basic types like <code>Int</code> , <code>Double</code> , <code>String</code> , etc). |
| <b>countByKey()</b>             | only available on RDDs of type <code>(K, V)</code> . Returns a 'Map' of <code>(K, Int)</code> pairs with the count of each key                                                                                                                                                                                                                                                                                                            |
| <b>foreach(func)</b>            | run a function <code>func</code> on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems                                                                                                                                                                                                                                                     |

# Actions

---

## Scala:

```
val f = sc.textFile("README.md")
val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
words.reduceByKey(_ + _).collect.foreach(println)
```

## Python:

```
from operator import add
f = sc.textFile("README.md")
words = f.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1))
words.reduceByKey(add).collect()
```

# Persistency

| <i>transformation</i>                                  | <i>description</i>                                                                                                                                                                                              |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEMORY_ONLY</b>                                     | 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. |
| <b>MEMORY_AND_DISK</b>                                 | 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.                                |
| <b>MEMORY_ONLY_SER</b>                                 | 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. |
| <b>MEMORY_AND_DISK_SER</b>                             | 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.                                                              |
| <b>DISK_ONLY</b>                                       | Store the RDD partitions only on disk.                                                                                                                                                                          |
| <b>MEMORY_ONLY_2,</b><br><b>MEMORY_AND_DISK_2, etc</b> | Same as the levels above, but replicate each partition on two cluster nodes.                                                                                                                                    |

---

| <i>transformation</i> | <i>description</i>                                                                                                                                                                                                 |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MEMORY_ONLY</b>    | Store RDD as deserialized Java objects in the JVM.<br>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. |

# RDD Operations (over 80 ops)

## Transformations (define a new RDD)

map  
filter  
sample  
union  
*PAIR RDDs (below)*  
*groupByKey*  
*reduceByKey*  
*join*  
*cache*

...

## Parallel operations (Actions) (return a result to driver)

reduce  
collect  
count  
save  
*lookupKey*  
...

Nothing is  
materialized

# Other RDD Operations

|                                              |                                                                               |                                                       |
|----------------------------------------------|-------------------------------------------------------------------------------|-------------------------------------------------------|
| <b>Transformations</b><br>(define a new RDD) | map<br>filter<br>sample<br><i>groupByKey</i><br><i>reduceByKey</i><br>cogroup | flatMap<br>union<br>join<br>cross<br>mapValues<br>... |
| <b>Actions</b><br>(output a result)          | collect<br>reduce<br>take<br>fold                                             | count<br>saveAsTextFile<br>saveAsHadoopFile<br>...    |

# Map versus flatmap

---

*Example 3-26. Python squaring the values in an RDD*

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
 print "%i " % (num)
```

Sometimes we want to produce multiple output elements for each input element. The operation to do this is called `flatMap()`. As with `map()`, the function we provide to `flatMap()` is called individually for each element in our input RDD. Instead of returning a single element, we return an iterator with our return values. Rather than producing an RDD of iterators, we get back an RDD that consists of the elements from all of the iterators. A simple usage of `flatMap()` is splitting up an input string into words, as shown in Examples 3-29 through 3-31.

**Input produces multiple elements; want to get all into the stream**

*Example 3-29. `flatMap()` in Python, splitting lines into words*

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

# flatMap() versus map()

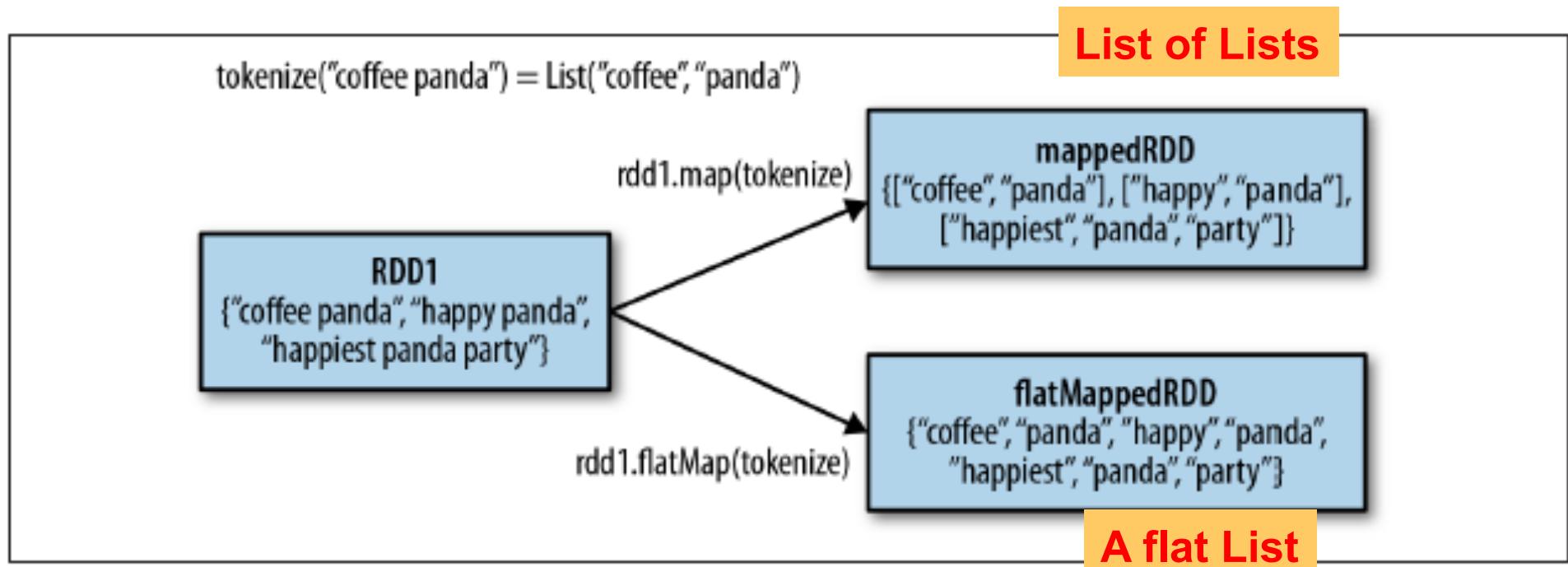


Figure 3-3. Difference between `flatMap()` and `map()` on an RDD

# Set operations

RDD1  
{coffee, coffee, panda,  
monkey, tea}

RDD2  
{coffee, money, kitty}

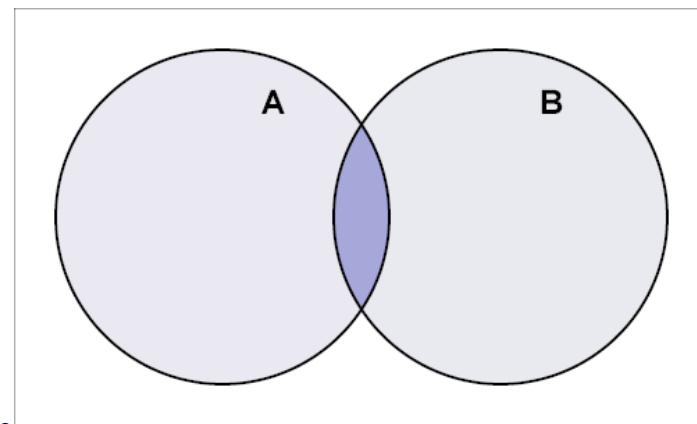
RDD1.distinct()  
{coffee, panda,  
monkey, tea}

RDD1.union(RDD2)  
{coffee, coffee, coffee,  
panda, monkey,  
monkey, tea, kitty}

RDD1.intersection(RDD2)  
{coffee, monkey}

RDD1.subtract(RDD2)  
{panda, tea}

- Distinct
- Union
- Intersection
- Subtract



# Set operations

RDD1  
{coffee, coffee, panda,  
monkey, tea}

RDD2  
{coffee, money, kitty}

RDD1.distinct()  
{coffee, panda,  
monkey, tea}

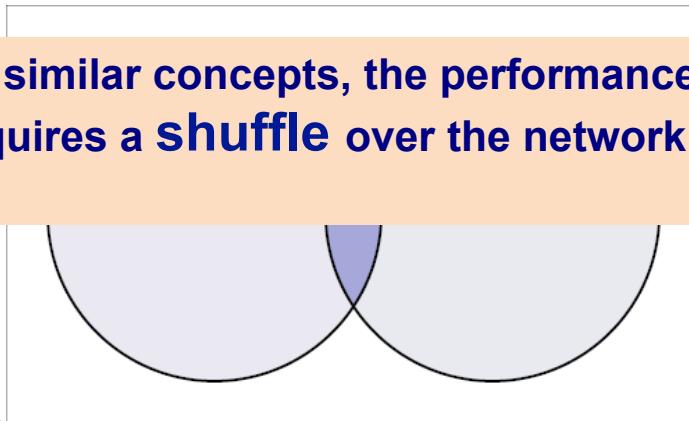
RDD1.union(RDD2)  
{coffee, coffee, coffee,  
panda, monkey,  
monkey, tea, kitty}

RDD1.intersection(RDD2)  
{coffee, monkey}

RDD1.subtract(RDD2)  
{panda, tea}

While intersection() and union() are two similar concepts, the performance of intersection() is much worse since it requires a **Shuffle** over the network to identify common elements.

- **Intersection**
- **Subtract**



# Cartesian Product, intersection, etc.

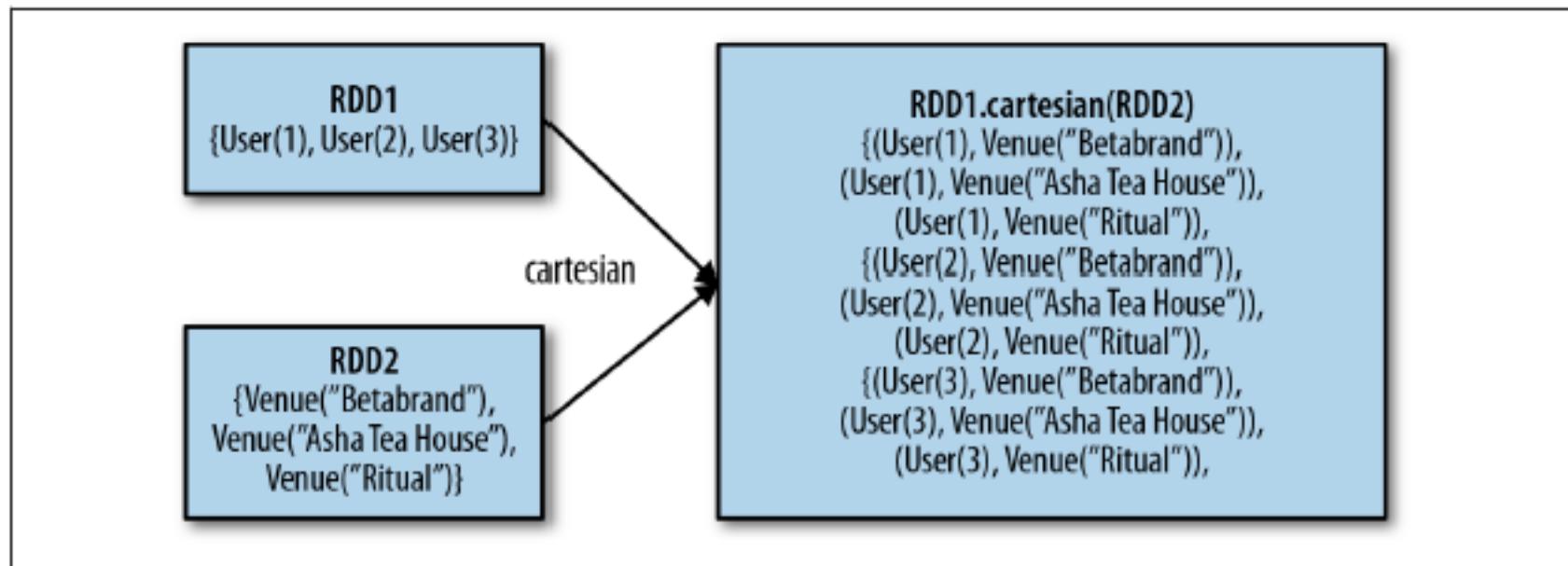


Figure 3-5. Cartesian product between two RDDs

**Be warned, however, that the Cartesian product is very expensive for large RDDs.**

*Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}*

| Function name                                          | Purpose                                                                                                                               | Example                                   | Result                |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|-----------------------|
| <code>map()</code>                                     | Apply a function to each element in the RDD and return an RDD of the result.                                                          | <code>rdd.map(x =&gt; x + 1)</code>       | {2, 3, 4, 4}          |
| <code>flatMap()</code>                                 | Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words. | <code>rdd.flatMap(x =&gt; x.to(3))</code> | {1, 2, 3, 2, 3, 3, 3} |
| <code>filter()</code>                                  | Return an RDD consisting of only elements that pass the condition passed to <code>filter()</code> .                                   | <code>rdd.filter(x =&gt; x != 1)</code>   | {2, 3, 3}             |
| <b>Distinct</b>                                        | Remove duplicates.                                                                                                                    | <code>rdd.distinct()</code>               | {1, 2, 3}             |
| <code>sample(withReplacement, fraction, [seed])</code> | Sample an RDD, with or without replacement                                                                                            | <code>rdd.sample(false, 0.5)</code>       | Nondeterministic      |
| <b>Sample</b>                                          |                                                                                                                                       |                                           |                       |

---

*Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}*

| Function name  | Purpose                                                      | Example                 | Result                      |
|----------------|--------------------------------------------------------------|-------------------------|-----------------------------|
| union()        | Produce an RDD containing elements from both RDDs.           | rdd.union(other)        | {1, 2, 3, 3, 4, 5}          |
| intersection() | RDD containing only elements found in both RDDs.             | rdd.intersection(other) | {3}                         |
| subtract()     | Remove the contents of one RDD (e.g., remove training data). | rdd.subtract(other)     | {1, 2}                      |
| cartesian()    | Cartesian product with the other RDD.                        | rdd.cartesian(other)    | {(1, 3), (1, 4), ... (3,5)} |

---

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

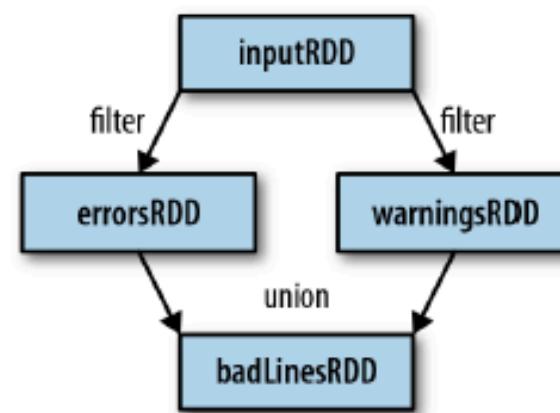
## union()

`union()` is a bit different than `filter()`, in that it operates on two RDDs instead of one. Transformations can actually operate on any number of input RDDs.



A better way to accomplish the same result as in [Example 3-14](#) would be to simply filter the `inputRDD` once, looking for either *error* or *warning*.

Finally, as you derive new RDDs from each other using transformations, Spark keeps track of the set of dependencies between different RDDs, called the *lineage graph*. It uses this information to compute each RDD on demand and to recover lost data if part of a persistent RDD is lost. [Figure 3-1](#) shows a lineage graph for [Example 3-14](#).



**Large Scale** [Figure 3-1](#). RDD lineage graph created during log analysis

# Actions: Collect can do a lot of damage

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

Example 3-15. Python error count using actions

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
 print line
```

Example 3-16. Scala error count using actions

```
println("Input had " + badLinesRDD.count() + " concerning lines")
println("Here are 10 examples:")
badLinesRDD.take(10).foreach(println)
```

Example 3-17. Java error count using actions

```
System.out.println("Input had " + badLinesRDD.cou
System.out.println("Here are 10 examples:")
for (String line: badLinesRDD.take(10)) {
 System.out.println(line);
}
```

In most cases RDDs can't just be collect()ed to the driver because they are **too large**.

# Persisting results on

---

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
 print line
```

In most cases RDDs can **NOT** just be collect()ed to the driver because they are too large. In these cases, it's common to write data out to a distributed storage system such as HDFS or Amazon S3.

- You can save the contents of an RDD using the `saveAsTextFile()` action, `saveAsSequenceFile()`, or any of a number of actions for various built-in formats. We will cover the different options for exporting data in Chapter 5.

`cache()`

It is important to note that each time we call a new action, the entire RDD must be computed “from scratch.” To avoid this inefficiency, users can persist intermediate results, as we will cover in “Persistence (Caching)” on page 44.

# count() action

---

- The act of creating a RDD does not cause any distributed computation to take place on the cluster.
- Rather, RDDs define logical datasets that are intermediate steps in a computation.
- Distributed computation occurs upon invoking an action on an RDD. For example, the count action returns the number of objects in an RDD.

```
rdd.count()
14/09/10 17:36:09 INFO SparkContext: Starting job: count at <console>:15
...
14/09/10 17:36:09 INFO SparkContext: Job finished: count at <console>:15,
took 0.18273803 s
res0: Long = 4
```

# Get data from Cluster to client (first, collect, take)

---

- RDDs have a number of methods that allow us to read data from the cluster into the Scala REPL on our client machine. Perhaps the simplest of these is `first`, which returns the first element of the RDD into the client:

```
rawblocks.first
...
res: String = "id_1","id_2","cmp_fname_c1","cmp_fname_c2",...

val head = rawblocks.take(10)
...
head: Array[String] = Array("id_1","id_2","cmp_fname_c1",...

head.length
...
res: Int = 10
```

# collect() → array in local memory

---

- The collect action returns an Array with all the objects from the RDD.
- This Array resides in local memory, not on the cluster.
- Be careful: don't collect too much data; this will cause your driver to crash

```
rdd.collect()
14/09/29 00:58:09 INFO SparkContext: Starting job: collect at <console>:17
...
14/09/29 00:58:09 INFO SparkContext: Job finished: collect at <console>:17,
took 0.531876715 s
res2: Array[(Int, Int)] = Array((4,1), (1,1), (2,2))
```

# saveAsTextFile()

- Actions need not only return results to the local process. The `saveAsTextFile` action saves the contents of an RDD to persistent storage like HDFS.

```
rdd.saveAsTextFile("hdfs://user/ds/mynumbers")
14/09/29 00:38:47 INFO SparkContext: Starting job: saveAsTextFile at <console>:15
...
14/09/29 00:38:49 INFO SparkContext: Job finished: saveAsTextFile at <console>:15,
took 1.818305149 s
```

- The action creates a directory and writes out each partition as a file within it. From the command line outside of the Spark shell:

```
hadoop fs -ls /user/ds/mynumbers
```

|            |   |    |            |   |                  |                       |
|------------|---|----|------------|---|------------------|-----------------------|
| -rw-r--r-- | 3 | ds | supergroup | 0 | 2014-09-29 00:38 | myfile.txt/_SUCCESS   |
| -rw-r--r-- | 3 | ds | supergroup | 4 | 2014-09-29 00:38 | myfile.txt/part-00000 |
| -rw-r--r-- | 3 | ds | supergroup | 4 | 2014-09-29 00:38 | myfile.txt/part-00001 |

# Foreach(): Print-friendly

- To make it easier to read the contents of an array, we can use the `foreach` method in conjunction with `println` (`print` in Python) to print each value in the array out on its own line:

---

- **Summary**

- **Spark**

- Transformations
- Actions
- On Base RDDs
- Starting to get excited about spark?

# Part 2: Spark Intro and Basics

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

# Example: filter Error lines from logfile

HDFS  
file

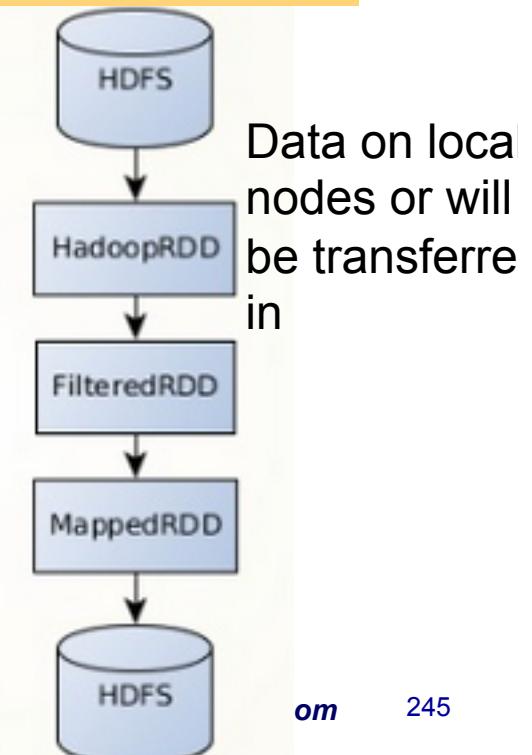
INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
 .filter(_.startsWith("ERROR"))
 .map(_.split(" ")(1))
 .saveAsTextFile("hdfs://<output>")
```



# Example: filter Error lines from logfile

HDFS  
file

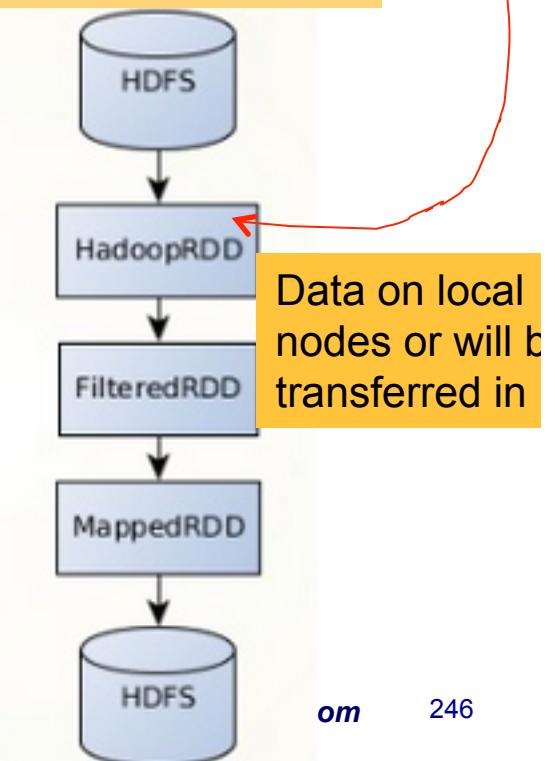
INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
 .filter(_.startsWith("ERROR"))
 .map(_.split(" ")(1))
 .saveAsTextFile("hdfs://<output>")
```



# Example: filter Error lines from logfile

HDFS  
file

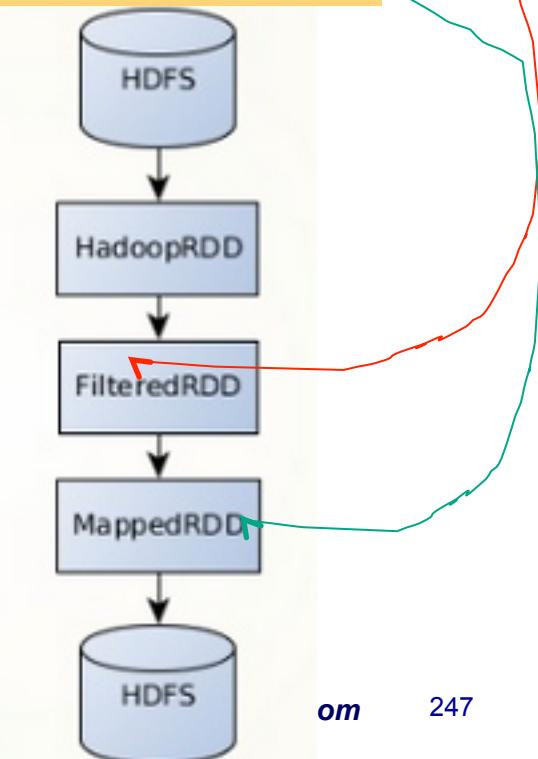
INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

Scala

```
sc.textFile("hdfs://<input>")
 .filter(_.startsWith("ERROR"))
 .map(_.split(" ")(1))
 .saveAsTextFile("hdfs://<output>")
```



# saveAsTextFile Action: Step by Step

HDFS  
file

INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```



The RDD.saveAsTextFile() action triggers a job. Tasks are started on scheduled executors.

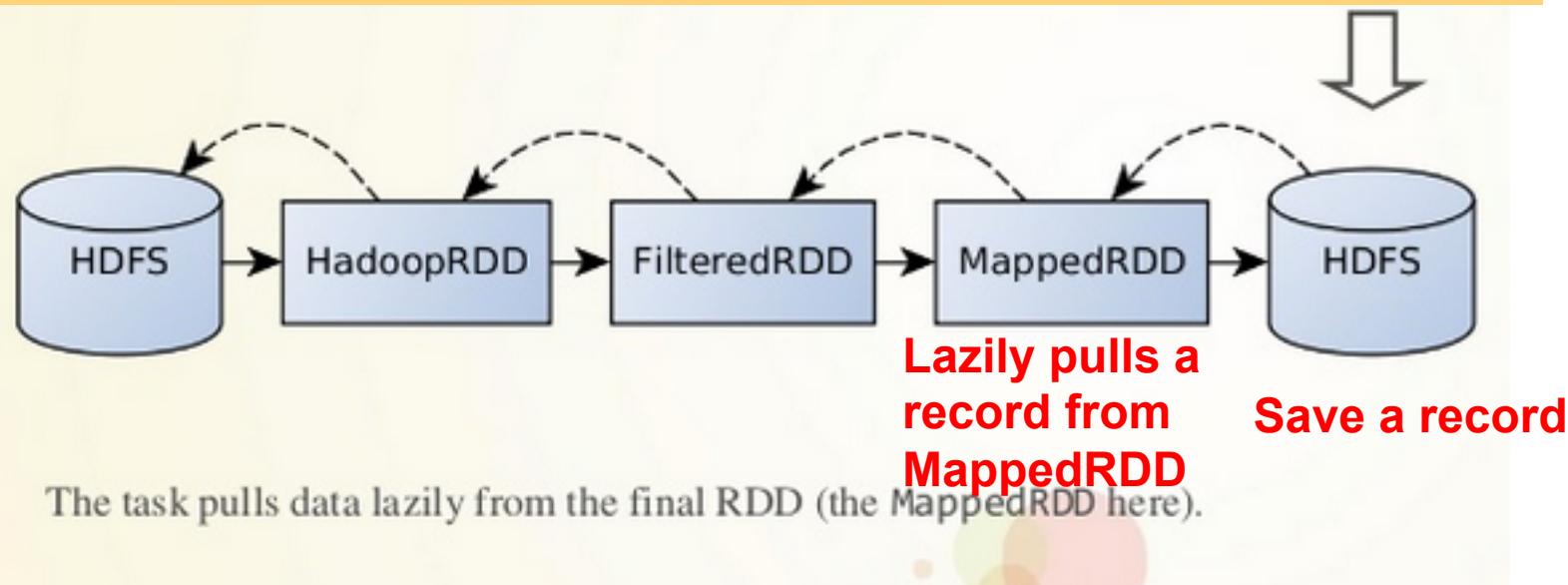
# saveAsTextFile has a domino effect

HDFS  
file

INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

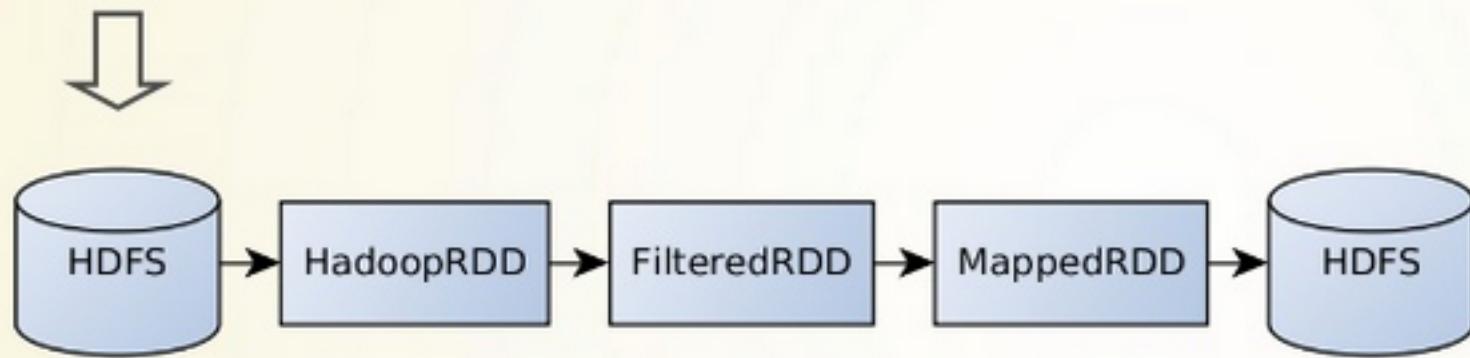
pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```



# Chain effect right back to the source

Step by Step



A record (text line) is pulled out from HDFS...

# Materializes a record that is passed forward in the pipeline

HDFS  
file

INFO not much going on here

ERROR 30330 task aborted with no status

• ERROR 44404 task aborted with no status

INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

INFO ...



... into a HadoopRDD

# Record is passed thru the filter

HDFS  
file

INFO not much going on here  
ERROR 30330 task aborted with no status  
• ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

INFO ...



Then filtered with the FilteredRDD

# Record is passed thru the filter but Rejected

HDFS  
file

INFO not much going on here  
ERROR 30330 task aborted with no status  
ERROR 44404 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```



Oops, not a match

# Action: Take 2

HDFS  
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status
- INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

ERROR ...



Another record is pulled out...

# Pass second record thru filter

HDFS  
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status
- INFO doodle

pySpark

```
sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

ERROR ...



Filtered again...

# Passes file and write to HDFS

HDFS  
file

INFO not much going on here

ERROR 30330 task aborted with no status

• ERROR 44404 task aborted with no status

INFO doodle

ERROR 30330 task aborted with no status

pySpark

```
sc.textFile("hdfs://<some input path or any local file") \
 .filter(lambda x: 'ERROR' in x) # Error anywhere in the line
 .map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```



Transformed by the MappedRDD (the error message is extracted)

# A Synthesized Iterator

```
new Iterator[String] {
 private var head: String = _
 private var headDefined: Boolean = false

 def hasNext: Boolean = headDefined || {
 do {
 try head = readOneLineFromHDFS(...) // (1) read from HDFS
 catch {
 case _: EOFException => return false
 }
 } while (!head.startsWith("ERROR")) // (2) filter closure
 true
 }

 def next: String = if (hasNext) {
 headDefined = false
 head.split(" ")(1) // (3) map closure
 } else {
 throw new NoSuchElementException("...")
 }
}
```

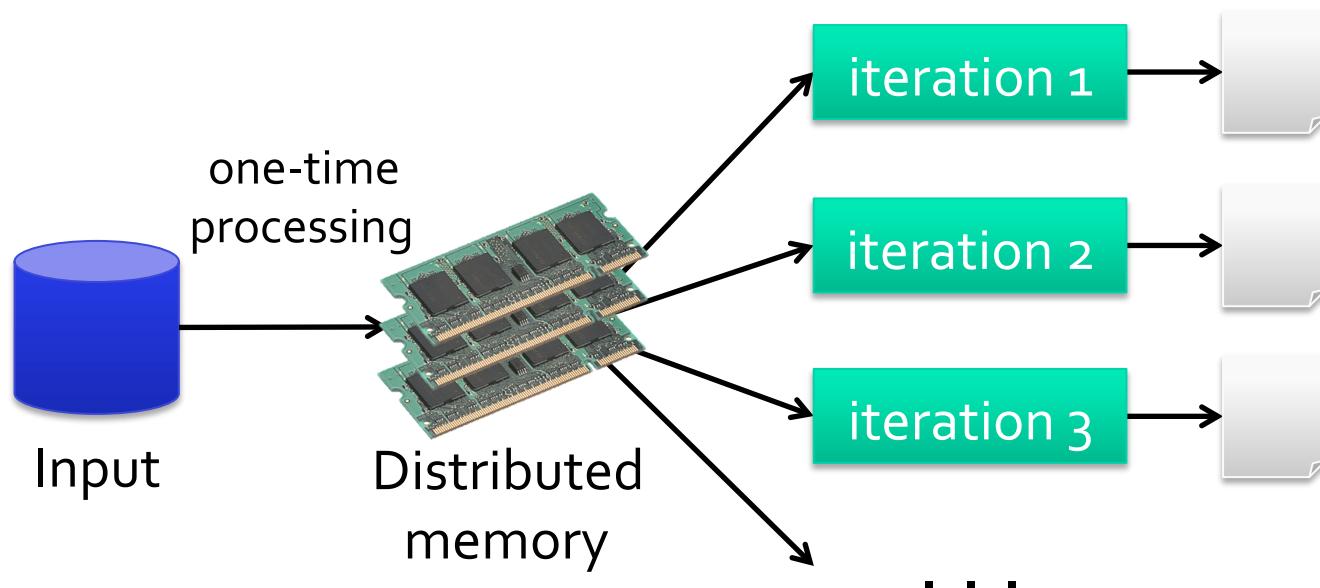
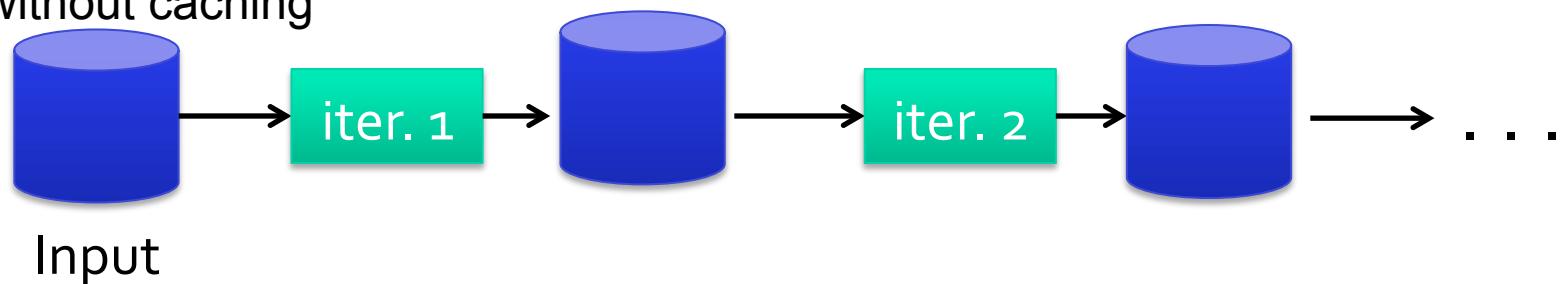
Constant Space Complexity!

---

- Caching

# Goal: Keep Working Set in RAM

Hadoop MapReduce  
Spark without caching



# Caching

While the contents of RDDs are transient by default, Spark provides a mechanism persisting the data in an RDD. After the first time an action requires computing such an RDD's contents, they are stored in memory or disk across the cluster. The next time an action depends on the RDD, it need not be recomputed from its dependencies. Its data is returned from the cached partitions directly.

```
cached.cache()
cached.count()
cached.take(10)
```

The call to `cache` indicates that the RDD should be stored the next time it's computed. The call to `count` computes it initially. The `take` action returns the first 10 elements the RDD as a local `Array`. When `take` is called, it accesses the cached elements of `cached` instead of recomputing them from their dependencies.

Spark defines a few different mechanisms, or `StorageLevel`s, for persisting RDDs. `rdd.cache()` is shorthand for `rdd.persist(StorageLevel.MEMORY)`, which stores the RDD as unserialized Java objects. When Spark estimates that a partition will not fit in memory, it simply will not store it, and it will be recomputed the next time it's needed. This level makes the most sense when the objects will be referenced frequently and/or require low-latency access, as it avoids any serialization overhead. Its drawback is that it takes up larger amounts of memory than its alternatives. Also, holding on to many small objects puts pressure on Java's garbage collection, which can result in stalls and

Large general slowness.

# The In-Memory Magic

- With cache
  - One block per RDD partition
  - LRU cache eviction
  - Locality aware
  - Evicted blocks can be *recomputed in parallel* with the help of RDD lineage DAG

# Cached intermediate (use by multiple downstream tasks if necessary instead of recalculating)

HDFS  
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status

INFO doodle

pySpark

```
cached= sc.textFile("hdfs://<some input path or any local file") \
 .filter(lambda x: 'ERROR' in x).cache() # Error anywhere in the line

cached.map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```

```
val cached = sc
 .textFile("hdfs://<input>")
 .filter(_.startsWith("ERROR"))
 .cache()

cached
 .map(_.split(" ")(1))
 .saveAsTextFile("hdfs://<output>")
```

# ERRORs are cached

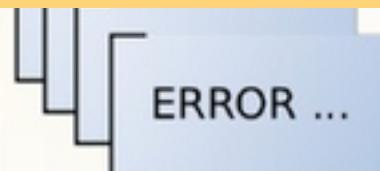
HDFS  
file

- INFO not much going on here
- ERROR 30330 task aborted with no status
- ERROR 44404 task aborted with no status

pySpark

```
cached= sc.textFile("hdfs://<some input path or any local file>") \
 .filter(lambda x: 'ERROR' in x).cache() # Error anywhere in the line
```

```
cached.map(lambda line: line.split(" ")[1]) #second word and the original line
 .saveAsTextFile("hdfs://<output> or any local directory")
```



All filtered error messages are cached in memory before being passed to the next RDD. LRU cache eviction is applied when memory is insufficient

---

# • Documentation

# help(pyspark)

<http://spark.apache.org/docs/latest/quick-start.html>

• ..

## Interactive Use

The bin/pyspark script launches a Python interpreter that is configured to run PySpark applications. To use pyspark interactively, first build Spark, then launch it directly from the command line without any options:

```
$ sbt/sbt assembly
$./bin/pyspark
```

The Python shell can be used explore data interactively and is a simple way to learn the API:

```
>>> words = sc.textFile("/usr/share/dict/words")
>>> words.filter(lambda w: w.startswith("spar")).take(5)
[u'spar', u'sparable', u'sparada', u'sparadrap', u'sparagrass']
>>> help(pyspark) # Show all pyspark functions
```

By default, the bin/pyspark shell creates SparkContext that runs applications locally on a single core. To connect to a non-local cluster, or use multiple cores, set the MASTER environment variable. For example, to use the bin/pyspark shell with a [standalone Spark cluster](#):

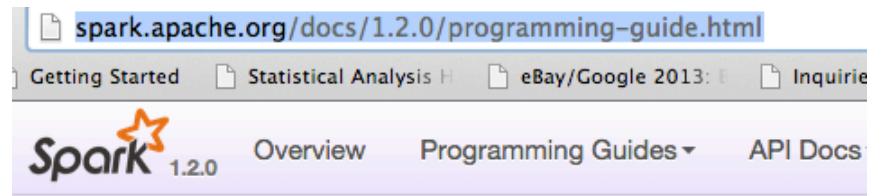
```
$ MASTER=spark://IP:PORT ./bin/pyspark
```

Or, to use four cores on the local machine:

```
$ MASTER=local[4] ./bin/pyspark
```

# Spark in one-page!

- <http://spark.apache.org/docs/1.4.0/programming-guide.html>



## Spark Programming Guide

- Overview
- Linking with Spark
- Initializing Spark
  - Using the Shell
- Resilient Distributed Datasets (RDDs)
  - Parallelized Collections
  - External Datasets
  - RDD Operations
    - Basics
    - Passing Functions to Spark
    - Working with Key-Value Pairs
    - Transformations
    - Actions
  - RDD Persistence
    - Which Storage Level to Choose?
    - Removing Data
- Shared Variables
  - Broadcast Variables
  - Accumulators
- Deploying to a Cluster
- Unit Testing
- Migrating from pre-1.0 Versions of Spark

# Summary

---

- I hope this example was useful in helping you understand the power of Spark and to ground some of the concepts

# bin/pyspark: Spark Shell

<http://spark.apache.org/docs/latest/quick-start.html>

## Interactive Use

The bin/pyspark script launches a Python interpreter that is configured to run PySpark applications. To use pyspark interactively, first build Spark, then launch it directly from the command line without any options:

```
$ sbt/sbt assembly
$./bin/pyspark
```

The Python shell can be used explore data interactively and is a simple way to learn the API:

```
>>> words = sc.textFile("/usr/share/dict/words")
>>> words.filter(lambda w: w.startswith("spar")).take(5)
[u'spar', u'sparable', u'sparada', u'sparadrap', u'sparagrass']
>>> help(pyspark) # Show all pyspark functions
```

**help(pyspark)**

By default, the bin/pyspark shell creates SparkContext that runs applications locally on a single core. To connect to a non-local cluster, or use multiple cores, set the MASTER environment variable. For example, to use the bin/pyspark shell with a [standalone Spark cluster](#):

```
$ MASTER=spark://IP:PORT ./bin/pyspark
```

**Connect to a cluster versus a local**

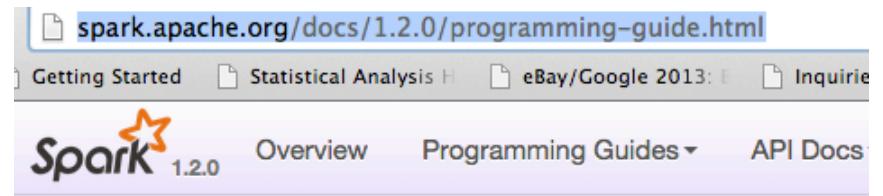
Or, to use four cores on the local machine:

```
$ MASTER=local[4] ./bin/pyspark
```

**Connect to a 4 cores on the local machine**

# Spark in one-page!

- <http://spark.apache.org/docs/1.4.0/programming-guide.html>



## Spark Programming Guide

- Overview
- Linking with Spark
- Initializing Spark
  - Using the Shell
- Resilient Distributed Datasets (RDDs)
  - Parallelized Collections
  - External Datasets
  - RDD Operations
    - Basics
    - Passing Functions to Spark
    - Working with Key-Value Pairs
    - Transformations
    - Actions
  - RDD Persistence
    - Which Storage Level to Choose?
    - Removing Data
- Shared Variables
  - Broadcast Variables
  - Accumulators
- Deploying to a Cluster
- Unit Testing
- Migrating from pre-1.0 Versions of Spark

## Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

| Transformation                                                    | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>map(func)</code>                                            | Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>filter(func)</code>                                         | Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>flatMap(func)</code>                                        | Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).                                                                                                                                                                                                                                                                                                                                                                      |
| <code>mapPartitions(func)</code>                                  | Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator&lt;T&gt; =&gt; Iterator&lt;U&gt;</code> when running on an RDD of type T.                                                                                                                                                                                                                                                                                                                |
| <code>mapPartitionsWithIndex(func)</code>                         | Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator&lt;T&gt;) =&gt; Iterator&lt;U&gt;</code> when running on an RDD of type T.                                                                                                                                                                                                                                                            |
| <code>sample(withReplacement, fraction, seed)</code>              | Sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator seed.                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>union(otherDataset)</code>                                  | Return a new dataset that contains the union of the elements in the source dataset and the argument.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>intersection(otherDataset)</code>                           | Return a new RDD that contains the intersection of elements in the source dataset and the argument.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>distinct([numTasks])</code>                                 | Return a new dataset that contains the distinct elements of the source dataset.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>groupByKey([numTasks])</code>                               | When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.<br><b>Note:</b> If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>combineByKey</code> will yield much better performance.<br><b>Note:</b> By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numTasks argument to set a different number of tasks. |
| <code>reduceByKey(func, [numTasks])</code>                        | When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) =&gt; V</code> . Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.                                                                                                                                                                                    |
| <code>aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])</code> | When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.                                                                                                  |
| <code>sortByKey([ascending], [numTasks])</code>                   | When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.                                                                                                                                                                                                                                                                                                          |
| <code>join(otherDataset, [numTasks])</code>                       | When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Actions

The following table lists some of the common actions supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

| Action                                                    | Meaning                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>reduce(func)</code>                                 | Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.                                                                                                                                                                                                |
| <code>collect()</code>                                    | Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.                                                                                                                                                                                                                            |
| <code>count()</code>                                      | Return the number of elements in the dataset.                                                                                                                                                                                                                                                                                                                                                                       |
| <code>first()</code>                                      | Return the first element of the dataset (similar to <code>take(1)</code> ).                                                                                                                                                                                                                                                                                                                                         |
| <code>take(n)</code>                                      | Return an array with the first <i>n</i> elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.                                                                                                                                                                                                                                       |
| <code>takeSample(withReplacement, num, [seed])</code>     | Return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.                                                                                                                                                                                                                                                  |
| <code>takeOrdered(n, [ordering])</code>                   | Return the first <i>n</i> elements of the RDD using either their natural order or a custom comparator.                                                                                                                                                                                                                                                                                                              |
| <code>saveAsTextFile(path)</code>                         | Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file.                                                                                                                                            |
| <code>saveAsSequenceFile(path)</code><br>(Java and Scala) | Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that either implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc). |
| <code>saveAsObjectFile(path)</code><br>(Java and Scala)   | Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using <code>SparkContext.objectFile()</code> .                                                                                                                                                                                                                                                              |
| <code>countByKey()</code>                                 | Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.                                                                                                                                                                                                                                                                                                              |
| <code>foreach(func)</code>                                | Run a function <i>func</i> on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.                                                                                                                                                                                                                 |

# Source Code available on GitHub

---

# Part 2: Spark Intro and Basics

---

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

---

# PAIR RDDS

**Key/value RDDs expose new operations (e.g., counting up reviews for each product, grouping together data with the same key, and grouping together two different RDDs).**

# Pair RDDs

---

This section covers how to work with RDDs of key/value pairs, which are a common data type required for many operations in Spark.

Key/value RDDs are commonly used to perform aggregations, and often we will do some initial ETL (extract, transform, and load) to get our data into a key/value format.

Key/value RDDs expose new operations (e.g., counting up reviews for each product, grouping together data with the same key, and grouping together two different RDDs).

# Pair RDDs

---

We also discuss an advanced feature that lets users control the layout of pair RDDs across nodes: partitioning.

Using controllable partitioning, applications can sometimes greatly reduce communication costs by ensuring that data will be accessed together and will be on the same node.

**JOINS:** (see in later sections in this lecture and subsequent lectures)

- This can provide significant speedups. We illustrate partitioning using the PageRank algorithm as an example.
- Choosing the right partitioning for a distributed dataset is similar to choosing the right data structure for a local one—in both cases, data layout can greatly affect performance.

# Pair RDDs: Key value pairs

---

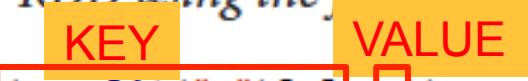
- **Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDDs.**
- **Act on each key in parallel or regroup data across the network**
  - Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel or regroup data across the network.
- **reduceByKey(): aggregate data separately for each key**
  - For example, pair RDDs have a reduceByKey() method that can aggregate data separately for each key, and a join() method that can merge two RDDs together by grouping elements with the same key.
- **It is common to extract fields from an RDD (representing, for instance, an event time, customer ID, or other identifier) and use those fields as keys in pair RDD operations**

# Pair RDDs Example

---

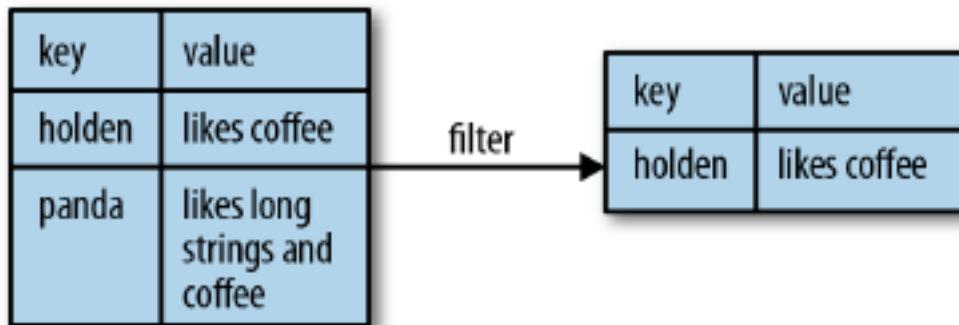
The way to build key-value RDDs differs by language. In Python, for the functions on keyed data to work we need to return an RDD composed of tuples (see [Example 4-1](#)).

*Example 4-1. Creating a pair RDD using the first word as the key in Python*

  
KEY                    VALUE  
`pairs = lines.map(lambda x: (x.split(" ")[0], x))`

**When creating a pair RDD from an in-memory collection in Scala and Python, we only need to call `SparkContext.parallelize()` on a collection of pairs.**

**Pair RDDs are allowed to use all the transformations available to standard RDDs. The same rules apply from “Passing Functions to Spark” on page 30 [Learning spark book].**



*Figure 4-1. Filter on value*

Pair RDDs are also still RDDs (of Tuple2 objects in Java/Scala or of Python tuples), and thus support the same functions as RDDs. For instance, we can take our pair RDD from the previous section and filter out lines longer than 20 characters, as shown in Examples 4-4 through 4-6 and Figure 4-1.

### Filter on Value being < 20

*Example 4-4. Simple filter on second element in Python*

```
result = pairs.filter(lambda keyValue: len(keyValue[1]) < 20)
```

*Example 4-5. Simple filter on second element in Scala*

```
pairs.filter{case (key, value) => value.length < 20}
```

# Part 2: Spark Intro and Basics

---

- **Part 2: Spark Intro and basics**

- Base RDD
- Fault tolerance (and lineage)
- Transformations and Actions
- Persistance
- Animated Example
- Pair RDDs
- Word count example

# Example 3

## Simple Spark Apps: WordCount

*Definition:*

*count how often each word appears  
in a collection of text documents*

This simple program provides a good test case for parallel processing, since it:

- requires a minimal amount of code
- demonstrates use of both symbolic and numeric values
- isn't many steps away from search indexing
- serves as a "Hello World" for Big Data apps

```
void map (String doc_id, String text):
 for each word w in segment(text):
 emit(w, "1");

void reduce (String word, Iterator group):
 int count = 0;

 for each pc in group:
 count += Int(pc);

 emit(word, String(count));
```

A distributed computing framework that can run WordCount **efficiently in parallel at scale** can likely handle much larger and more interesting compute problems

# Word Count in Map-Reduce

```
def map(key, value):
 for word in value.split():
 emit(word, 1)
```

```
def reduce(key, values):
 count = 0
 for val in values:
 count += val
 emit(key, count)
```

*# emit is a function that performs distributed I/O*

Each document is passed to a mapper, which does the tokenization. The output of the mapper is reduced by key (word) and then counted.

What is the data flow for word count?

The fast cat  
wears no hat.

The cat in the  
hat ran fast.

|      |   |
|------|---|
| cat  | 2 |
| fast | 2 |
| hat  | 2 |
| in   | 1 |
| no   | 1 |
| ran  | 1 |
| ...  |   |



# Word Frequency

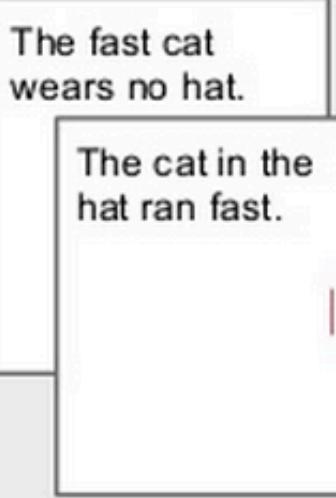
```
from operator import add

def tokenize(text):
 return text.split()

text = sc.textFile("tolstoy.txt") # Create RDD

Transform
wc = text.flatMap(tokenize)
wc = wc.map(lambda x: (x,1)).reduceByKey(add)

wc.saveAsTextFile("counts") # Action
```



|      |   |
|------|---|
| cat  | 2 |
| fast | 2 |
| hat  | 2 |
| in   | 1 |
| no   | 1 |
| ran  | 1 |
| ...  |   |

| Key   | Value |
|-------|-------|
| the   | 1     |
| fast  | 1     |
| wears | 1     |
| cat   | 1     |
| ....  | ...   |



# Word Count

## See NOTEBOOK

Write some words into a file

```
%%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Overwriting wordcount.txt

## Word Count

Attention:

flatMap works applying a function  
that returns a sequence for each  
element in the list, and flattening  
the results into the original list.

```
#Count words in README.md
logFileName = 'wordcount.txt'
text_file = sc.textFile(logFileName)
counts = text_file.flatMap(lambda line: line.split(" "))
 .map(lambda word: (word, 1))
 .reduceByKey(lambda a, b: a + b)
for v in counts.collect():
 print v

(u'ciao', 1)
(u'bonjour', 1)
(u'nihao', 2)
(u'hola', 2)
(u'konnichiwa', 1)
(u'hallo', 1)
(u'hi', 3)
(u'hello', 2)
(u'ola', 1)
```

# Word Count broken down

```
1 hello hi hi hallo
2 bonjour hola hi ciao
3 nihao konnichiwa ola
4 hola nihao hello
```

Flat  
Map

```
[hello, hi, hi, hallo, bonjour, hola, hi, ciao,
nihao, konnichiwa, ola, hola, nihao, hello]
```

Map

```
(u'ciao', 1)
(u'bonjour', 1)
(u'nihao', 2)
(u'holo', 2)
(u'konnichiwa', 1)
(u'hallo', 1)
(u'hi', 3)
(u'hello', 2)
(u'ola', 1)
```

reduce  
ByKey

```
(hello,1),
(hi,1),
(hi,1),
(hallo,1),
(bonjour,1),
(holo,1),
(hi,1),
(ciao,1),
(nihao,1),
(konnichiwa,1),
(ola,1),
(holo,1),
(nihao,1),
(hello,1)
```

---

```
]#Example 4-1. Creating a pair RDD using the first word as the key in Python
lines = sc.parallelize(["Data line 1", "Mining line 2", "data line 3", "Data line 4", "Data Mining line 5"])
pairs = lines.map(lambda x: (x.split(" ")[0], x)) #first word and the original line
pairs.collect()

] [('Data', 'Data line 1'),
 ('Mining', 'Mining line 2'),
 ('data', 'data line 3'),
 ('Data', 'Data line 4'),
 ('Data', 'Data Mining line 5')]
```

---

```
In [15]: #Paired RDD examples
logFileName='log.txt'
#Word count for error messages exercise
READ Lines And PRINT
recs = sc.textFile(logFileName)
#recs = sc.textFile(logFileName).filter(lambda l: "ERROR" in l)
wcErrors = recs.flatMap(lambda l: l.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda x, y: x + y)
for (key, value) in wcErrors.collect():
 print "WordCount-->> %s:%d"%(key, value)
```

```
WordCount-->> :1
WordCount-->> dying:1
WordCount-->> for:1
WordCount-->> reasons:1
WordCount-->> ERROR mysql:1
WordCount-->> angry:1
WordCount-->> cluster::1
WordCount-->> context:1
WordCount-->> spark:2
WordCount-->> with:1
WordCount-->> you:1
WordCount-->> me?:1
WordCount-->> WARN dave,:1
WordCount-->> ERROR php::1
WordCount-->> unknown:1
WordCount-->> it:1
WordCount-->> replace:1
WordCount-->> cluster:1
WordCount-->> missing...darn:1
WordCount-->> WARN xylons:1
WordCount-->> at:1
WordCount-->> ERROR :1
WordCount-->> approaching:1
WordCount-->> are:1
```

---

# Word count notebook

# Wordcount Notebook

---

## DEMO

- **Download the following notebook**
  - [https://www.dropbox.com/s/ul0l3q98w54dr8x/  
WordCount.ipynb?dl=0](https://www.dropbox.com/s/ul0l3q98w54dr8x/WordCount.ipynb?dl=0)
- **Cd to the directory that contains the wordcount notebook**
  - cd /Users/jshanahan/Dropbox/NativeX-Internal/Publications/  
WSDM-2016
- **Launch Notebook**
  - /Users/jshanahan/anaconda/bin/ipython notebook&

localhost:8888/notebooks/Notebooks/WordCount/WordCount.ipynb#

MIDS-MLS-2015 nbviewer.ipython.org Stanford Machine L Getting Started Statistical Analysis H eBay/Google 2013: E Inquiries InferPatents WindAlert - Coyote F SamCam www.3rdavekite.com Kiting james@yottapartners Import

# jupyter WordCount (autosaved)

File Edit View Insert Cell Kernel Help Python 2

In [2]:

```
import os
import sys
#spark_home = os.environ['SPARK_HOME'] = '/Users/liang/Downloads/spark-1.4.1-bin-hadoop2.6/'
spark_home = os.environ['SPARK_HOME'] = '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6'

if not spark_home:
 raise ValueError('SPARK_HOME environment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to

version 1.5.0

Using Python version 2.7.11 (default, Dec 6 2015 18:57:58)  
SparkContext available as sc, HiveContext available as sqlContext.

In [3]:

```
%writefile wordcount.txt
hello hi hi hallo
bonjour hola hi ciao
nihao konnichiwa ola
hola nihao hello
```

Writing wordcount.txt

In [4]:

```
cat wordcount.txt
```

hello hi hi hallo  
bonjour hola hi ciao  
nihao konnichiwa ola  
hola nihao hello

In [ ]:

In [3]:

```
#Count words in README.md
logFileName = 'wordcount.txt'
text_file = sc.textFile(logFileName)
counts = text_file.flatMap(lambda line: line.split(" ")) \
 .map(lambda word: (word, 1)) \
 .reduceByKey(lambda a, b: a + b)
for v in counts.collect():
 print v
```

(u'ciao', 1)  
(u'bonjour', 1)

---

All actions available on the Base RDD

Plus more

# Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

Table 4-1. Transformations on one pair RDD (example:  $\{(1, 2), (3, 4), (3, 6)\}$ )

| Function name                                                                         | Purpose                                                         | Example                                                   | Result                                                |
|---------------------------------------------------------------------------------------|-----------------------------------------------------------------|-----------------------------------------------------------|-------------------------------------------------------|
| reduceByKey(func)                                                                     | Combine values with the same key.                               | <code>rdd.reduceByKey(<br/>    (x, y) =&gt; x + y)</code> | $\{(1, 2), (3, 4), (3, 6)\}$<br>$\{(1, 2), (3, 10)\}$ |
| groupByKey()                                                                          | Group values with the same key.                                 | <code>rdd.groupByKey()</code>                             | $\{(1, [2]), (3, [4, 6])\}$                           |
| combineByKey<br>Key(createCombiner,<br>mergeValue,<br>mergeCombiners,<br>partitioner) | Combine values with the same key using a different result type. | See Examples 4-12 through 4-14.                           | Value is a List                                       |

# Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

`mapValues(func)`

Apply a function to each value of a pair RDD without changing the key.

`rdd.mapValues(x => x+1)`

$\{(1, 3), (3, 5), (3, 7)\}$

`flatMapValues(func)`

Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.

`rdd.flatMapValues(x => (x to 5)`

$\{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)\}$

`keys()`

Return an RDD of just the keys.

`rdd.keys()`

$\{1, 3, 3\}$

$\{1, 3, 3\}$

`values()`

Return an RDD of just the values.

`rdd.values()`

$\{2, 4, 6\}$

$\{2, 4, 6\}$

`sortByKey()`

Return an RDD sorted by the key.

`rdd.sortByKey()`

$\{(1, 2), (3, 4), (3, 5), (3, 6)\}$

[an@gmail.com](mailto:an@gmail.com)

293

# Transformations on one Pair RDD

$\{(1, 2), (3, 4), (3, 6)\}$

$\{(1, 3), (3, 5), (3, 7)\}$

| Function name            | Purpose                           | Example                      | Result                       |
|--------------------------|-----------------------------------|------------------------------|------------------------------|
| <code>values()</code>    | Return an RDD of just the values. | <code>rdd.values()</code>    | $\{2, 4, 6\}$                |
| <code>sortByKey()</code> | Return an RDD sorted by the key.  | <code>rdd.sortByKey()</code> | $\{(1, 2), (3, 4), (3, 6)\}$ |

$\{1, 3, 3\}$

# Python:

Curly braces create [dictionaries](#) or [sets](#). Square brackets create [lists](#).

The are called literals; set literals:

```
aset = {'foo', 'bar'}
```

or a dictionary literal:

```
adict = {'foo': 42, 'bar': 81}
empty_dict = {}
```

or a list literal:

```
alist = ['foo', 'bar', 'bar']
empty_list = []
```

To create an empty set, you can only use `set()`.

Sets are collections of *unique* elements and you cannot order them. Lists are ordered sequences of elements, and values can be repeated. Dictionaries map keys to values, keys must be unique. Set and dictionary keys must meet other restrictions as well, so that Python can actually keep track of them efficiently and know they are and will remain unique.

# joins

---

## Joins

Some of the most useful operations we get with keyed data comes from using it together with other keyed data. Joining data together is probably one of the most common operations on a pair RDD, and we have a full range of options including right and left outer joins, cross joins, and inner joins.

The simple `join` operator is an inner join.<sup>1</sup> Only keys that are present in both pair RDDs are output. When there are multiple values for the same key in one of the inputs, the resulting pair RDD will have an entry for every possible pair of values with that key from the two input RDDs. A simple way to understand this is by looking at [Example 4-17](#).

# Inner join: Transformation of two pair RDDs

Table 4-2. Transformations on two pair RDDs ( $rdd = \{(1, 2), (3, 4), (3, 6)\}$  other =  $\{(3, 9)\}$ )

| Function name | Purpose                                              | Example                                                                                                            | Result                                  |
|---------------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| subtractByKey | Remove elements with a key present in the other RDD. | <code>rdd.subtractByKey(other)</code><br><code>rdd = {(1, 2), (3, 4), (3, 6)}</code> other = <code>{(3, 9)}</code> | <code>{(1, 2)}</code>                   |
| join          | Perform an inner join between two RDDs.              | <code>rdd.join(other)</code>                                                                                       | <code>{(3, (4, 9)), (3, (6, 9))}</code> |

Keys must be present in both RDDs

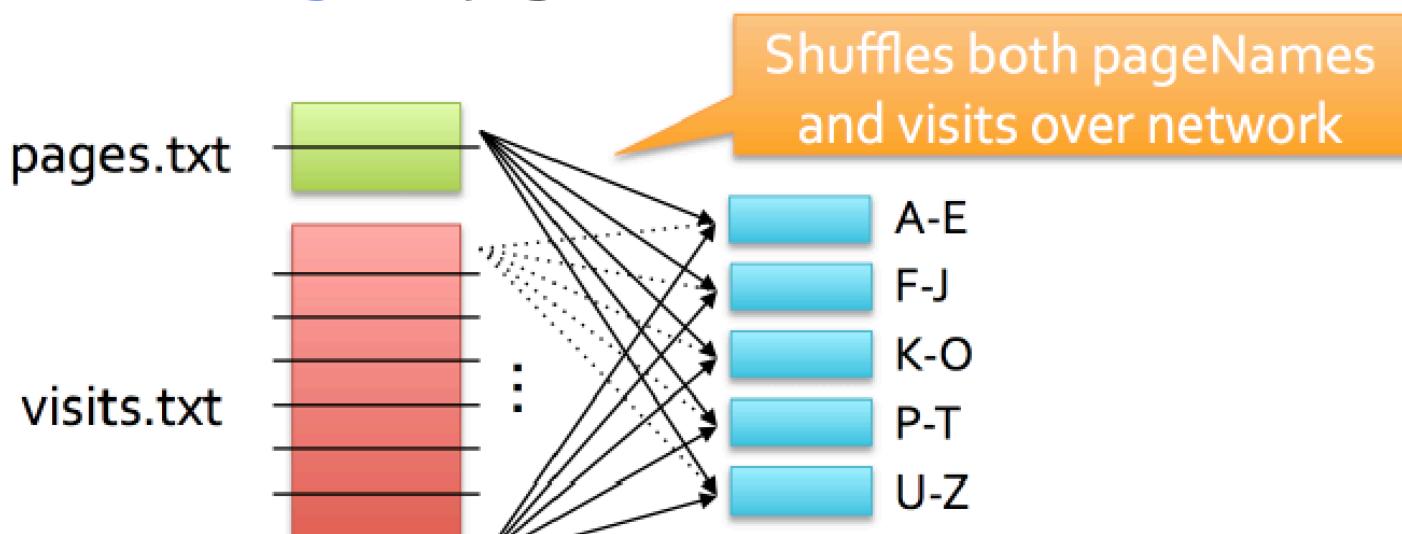
# Joins (not efficient...WHY?)

## Example Join

```
// Load RDD of (URL, name) pairs
val pageNames = sc.textFile("pages.txt").map(...)

// Load RDD of (URL, visit) pairs
val visits = sc.textFile("visits.txt").map(...)

val joined = visits.join(pageNames)
```



# HashJoin MapSide Join

---

- **MRJob (section 5.2)**
  - <http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/pjd6maluq4ogt7m/HW02-Supporting-Material.ipynb>
  - <https://www.dropbox.com/s/pjd6maluq4ogt7m/HW02-Supporting-Material.ipynb?dl=0>
- **Spark Notebook on Join (MapSide Join)**
  - <https://www.dropbox.com/s/gchzblskg7bhzxw/1-Airlines-HashsideJoinVia-Broadcast-and-EDA-Quiz.ipynb?dl=0>

# MrJob (slow join! But it scales)

```
In [117]: mkdir Join
```

•

```
In [120]: %%writefile Join/customers.dat
Alice Bob|not bad|US
Sam Sneed|valued|CA
Jon Sneed|valued|CA
Arnold Wesise|not so good|UK
Henry Bob|not bad|US
Yo Yo Ma|not so good|CA
Jon York|valued|CA
Alex Ball|valued|UK
Jim Davis|not so bad|JA
```

Overwriting Join/customers.dat

```
In [119]: %%writefile Join/countries.dat
United States|US
Canada|CA
United Kingdom|UK
Italy|IT
```

Writing Join/countries.dat

- 
- **Spark HASH Join**
  - **<https://www.dropbox.com/s/gchzb1skg7bhzxw/1-Airlines-HashsideJoinVia-Broadcast-and-EDA-Quiz.ipynb?dl=0>**

```
n [122]: %%writefile Join/MRJoin.py
Adapted for MrJob from Joe Stein's example at:
http://allthingshadoop.com/2011/12/16/simple-hadoop-streaming-tutorial-using-joins-and-keys-with-python/

import sys, os, re
from mrjob.job import MRJob

class MRJoin(MRJob):

 # Performs secondary sort
 SORT_VALUES = True

 def mapper(self, _, line):
 splits = line.rstrip("\n").split("|")

 if len(splits) == 2: # country data
 symbol = 'A' # make country sort before person data
 country2digit = splits[1]
 yield country2digit, [symbol, splits]
 else: # person data
 symbol = 'B'
 country2digit = splits[2]
 yield country2digit, [symbol, splits]

 def reducer(self, key, values):
 countries = [] # should come first, as they are sorted on artificial key 'A'
 for value in values:
 if value[0] == 'A': #if we have multiple records on left/country side
 countries.append(value)
 if value[0] == 'B':
 for country in countries: #take the crossproduct country X Person
 yield key, country[1:] + value[1:]

 if __name__ == '__main__':
 MRJoin.run()

Writing Join/MRJoin.py
```

```
n [124]: !python Join/MRJoin.py Join/countries.dat Join/customers.dat
```

```
18 ## Module Constants
19 APP_NAME = "Flight Delay Analysis"
20 DATE_FMT = "%Y-%m-%d"
21 TIME_FMT = "%H%M"
22
23 fields = ('date', 'airline', 'flightnum', 'origin', 'dest', 'dep',
24 'dep_delay', 'arr', 'arr_delay', 'airtime', 'distance')
25 Flight = namedtuple('Flight', fields)
26
27 ## Closure Functions
28 def parse(row):
29 """
30 Parses a row and returns a named tuple.
31 """
32
33 row[0] = datetime.strptime(row[0], DATE_FMT).date()
34 row[5] = datetime.strptime(row[5], TIME_FMT).time()
35 row[6] = float(row[6])
36 row[7] = datetime.strptime(row[7], TIME_FMT).time()
37 row[8] = float(row[8])
38 row[9] = float(row[9])
39 row[10] = float(row[10])
40 return Flight(*row[:11])
41
42 def split(line):
43 """
44 Operator function for splitting a line with csv module
45 """
46 reader = csv.reader(StringIO(line))
47 return reader.next()
48
49 def plot(dolays):
```

```
: # Load the airlines lookup dictionary
airlines = dict(sc.textFile("data/ontime/airlines.csv").map(split).collect())

airlines

: {'19719': 'Coastal Air Transport: CTL',
 '19718': 'Cresent Medivac: CRH',
 '19713': 'Alliance Airlines: ACN',
 '19712': 'Business Express Airlines: BEA',
 '19711': 'Eagle Airline: EGA',
 '19710': 'Armstrong Air Service Inc.: AAP',
 '19717': 'Comuter Airlines (Ca): CMR',
 '19716': 'Barken International: BRK',
 '19715': 'West Penn Commuter: BAT',
 '19714': 'Astor Air: ASR',
 '19399': 'New York Air Inc.: NY',
 '19398': 'Dauphin Island Airways: DAU',
 '19397': 'Republic Airlines Inc.: RC',
 '19396': 'Sky West Aviation Inc.: QG',
 '19395': 'Mid-South Aviation Inc. (1): RCA',
 '19394': 'Mid-South Aviation Inc.: VL',
 '19393': 'Southwest Airlines Co.: WN',
 '19392': 'Rich International Airways: RIQ',
 '19391': 'Pacific Southwest Airlines: PS (1)',
```

```

6]: # Broadcast the lookup dictionary to the cluster
airline_lookup = sc.broadcast(airlines)

Read the CSV Data into an RDD
flights = sc.textFile("data/ontime/flights.csv").map(split).map(parse)
|
Map the total delay to the airline (joined using the broadcast value)
delays = flights.map(lambda f: (airline_lookup.value[f.airline],
 add(f.dep_delay, f.arr_delay)))

Reduce the total delay for the month to the airline
delays = delays.reduceByKey(add).collect()
delays = sorted(delays, key=itemgetter(1))

Provide output from the driver
for d in delays:
 print "%0.0f minutes delayed\t%s" % (d[1], d[0])

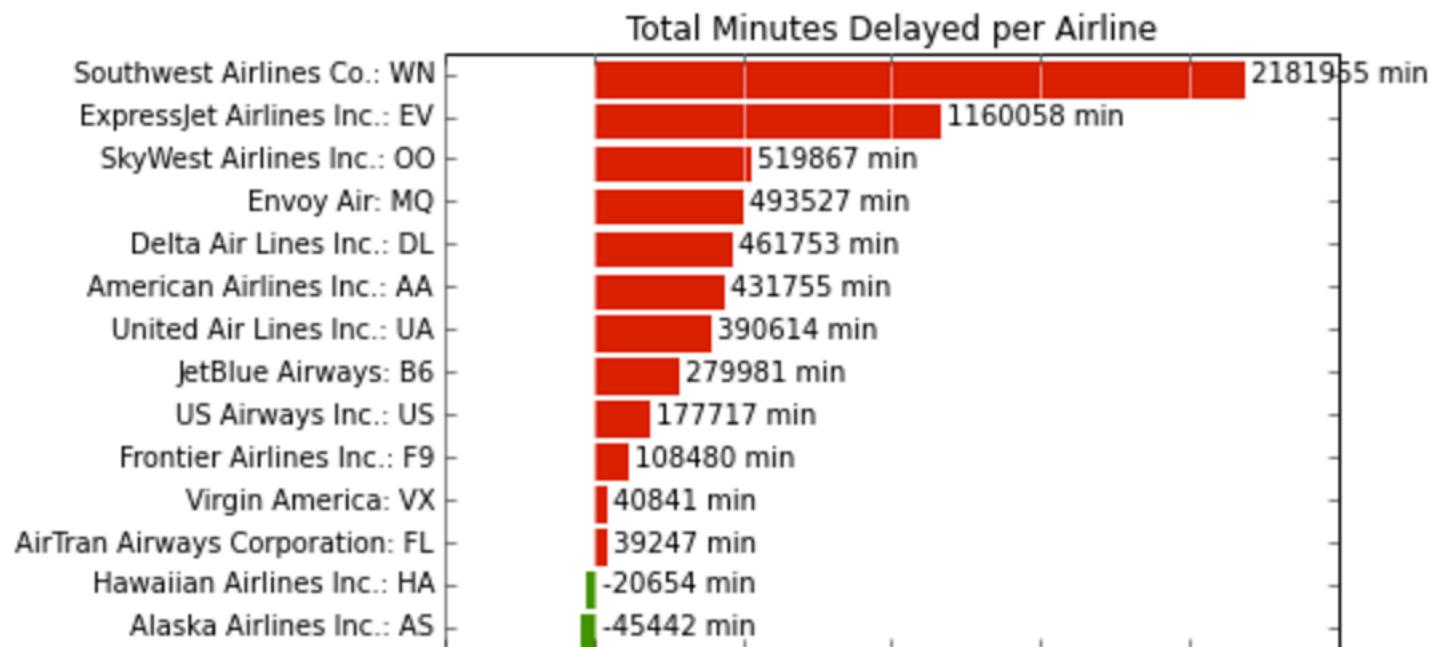
Show a bar chart of the delays
plot(delays)

```

```

-45442 minutes delayed Alaska Airlines Inc.: AS
-20654 minutes delayed Hawaiian Airlines Inc.: HA
39247 minutes delayed AirTran Airways Corporation: FL
40841 minutes delayed Virgin America: VX
108480 minutes delayed Frontier Airlines Inc.: F9
177717 minutes delayed US Airways Inc.: US
279981 minutes delayed JetBlue Airways: B6
390614 minutes delayed United Air Lines Inc.: UA
431755 minutes delayed American Airlines Inc.: AA
461753 minutes delayed Delta Air Lines Inc.: DL
493527 minutes delayed Envoy Air: MQ
519867 minutes delayed SkyWest Airlines Inc.: OO
1160058 minutes delayed ExpressJet Airlines Inc.: EV
2181955 minutes delayed Southwest Airlines Co.: WN

```



# Transformation of two pair RDDs

---

Table 4-2. Transformations on two pair RDDs ( $rdd = \{(1, 2), (3, 4), (3, 6)\}$  other =  $\{(3, 9)\}$ )

| Function name  | Purpose                                                                         | Example                                | Result                                                                          |
|----------------|---------------------------------------------------------------------------------|----------------------------------------|---------------------------------------------------------------------------------|
| subtractByKey  | Remove elements with a key present in the other RDD.                            | <code>rdd.subtractByKey(other)</code>  | $\{(1, 2)\}$                                                                    |
| join           | Perform an inner join between two RDDs.                                         | <code>rdd.join(other)</code>           | $\{(3, (4, 9)), (3, (6, 9))\}$                                                  |
| rightOuterJoin | Perform a join between two RDDs where the key must be present in the first RDD. | <code>rdd.rightOuterJoin(other)</code> | $\{(3, (\text{Some}(4), 9)), (3, (\text{Some}(6), 9))\}$                        |
| leftOuterJoin  | Perform a join between two RDDs where the key must be present in the other RDD. | <code>rdd.leftOuterJoin(other)</code>  | $\{(1, (2, \text{None})), (3, (4, \text{Some}(9))), (3, (6, \text{Some}(9)))\}$ |
| cogroup        | Group data from both RDDs sharing the same key.                                 | <code>rdd.cogroup(other)</code>        | $\{(1, ([2], [])), (3, ([4, 6], [9]))\}$                                        |

# Quiz 10.8.1 Paired RDDs

---

Given the following paired RDDs

$$\text{RDD1} = \{(1, 2), (3, 4), (3, 6)\}$$

$$\text{RDD2} = \{(3, 9) (3, 6)\}$$

Using PySpark, write code to perform an inner join of these paired RDDs. What is the resulting RDD? Make your PySpark notebook available also (using via an NBViewer link):

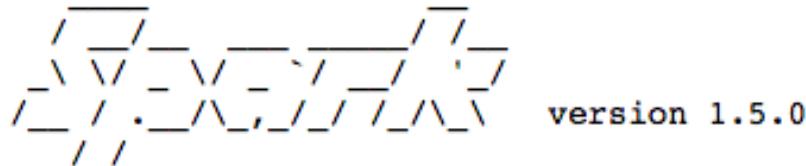
- A:  $[(3, (4, 9)), (3, (6, 9))]$
- B:  $[(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 6))]$
- C:  $[(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 9))]$
- D: None of the above

# Quiz 10.8.1 Notebook

```
import os
import sys #current as of 9/26/2015
spark_home = os.environ['SPARK_HOME'] =
 '/Users/jshanahan/Dropbox/Lectures-UC-Berkeley-ML-Class-2015/spark-1.5.0-bin-hadoop2.6/'

if not spark_home:
 raise ValueError('SPARK_HOME enviroment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip'))
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

Welcome to



Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)  
SparkContext available as sc, HiveContext available as sqlContext.

```
rdd1 = sc.parallelize({(1, 2), (3, 4), (3, 6)})
rdd2 = sc.parallelize({(3, 9), (3, 6)})
joinedRdd=rdd1.join(rdd2)
#RDD1 = {(1, 2), (3, 4), (3, 6)}
#RDD2 = {(3, 9) (3, 6)}
joinedRdd.collect()

[(3, (4, 9)), (3, (4, 6)), (3, (6, 9)), (3, (6, 6))]
```

# Aggregations

---

When datasets are described in terms of key/value pairs, it is common to want to aggregate statistics across all elements with the same key. We have looked at the `fold()`, `combine()`, and `reduce()` actions on basic RDDs, and similar per-key transformations exist on pair RDDs. Spark has a similar set of operations that combines values that have the same key. These operations return RDDs and thus are transformations rather than actions.

`reduceByKey()` is quite similar to `reduce()`; both take a function and use it to combine values. `reduceByKey()` runs several parallel reduce operations, one for each key in the dataset, where each operation combines values that have the same key. Because datasets can have very large numbers of keys, `reduceByKey()` is not implemented as an action that returns a value to the user program. Instead, it returns a new RDD consisting of each key and the reduced value for that key.

---

## Combiners

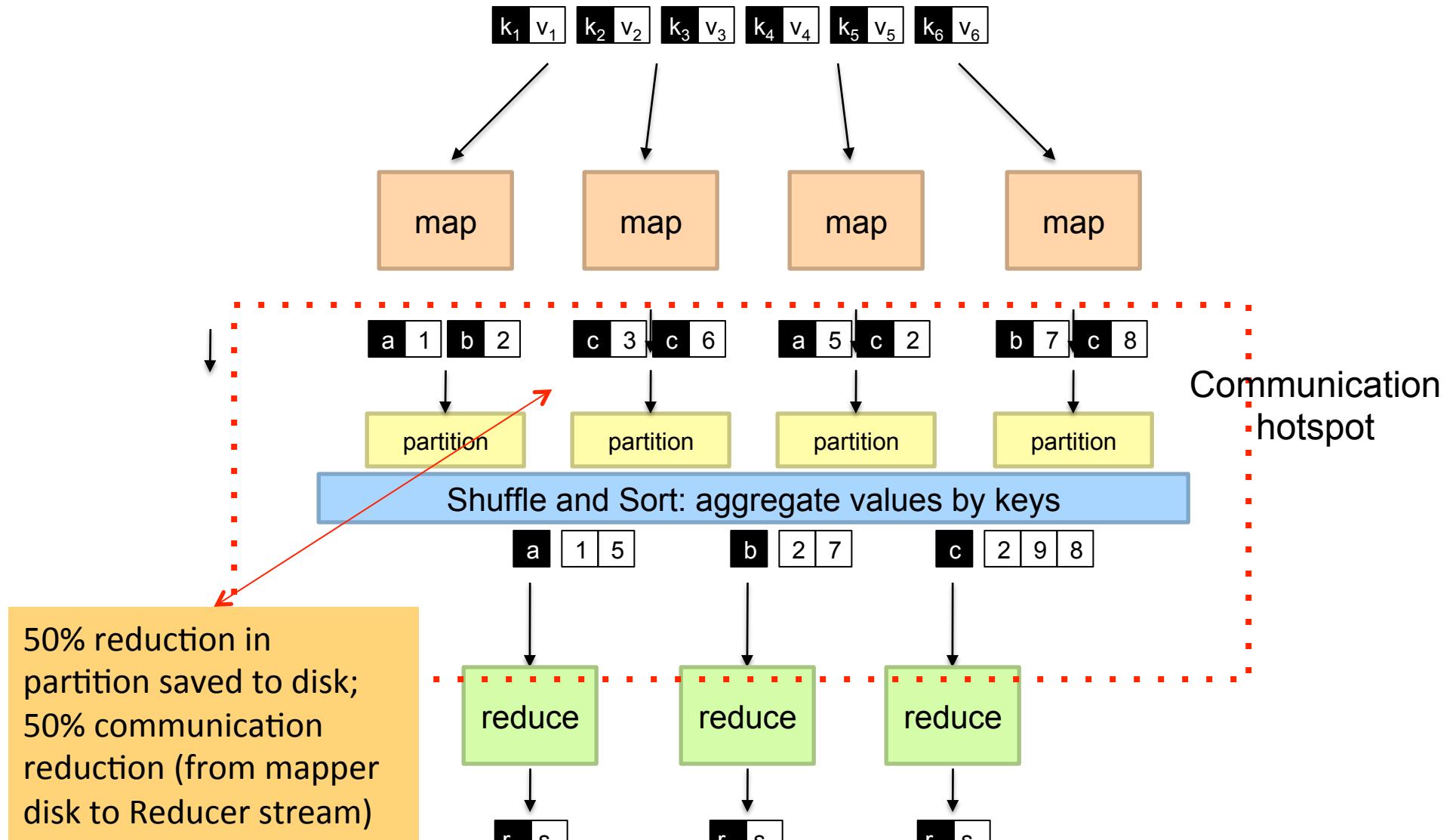
### Commutative and associative

# Tools for Synchronization

---

- **Cleverly-constructed data structures (in memory)**
  - Bring partial results together
- **Sort order of intermediate keys**
  - Control order in which reducers process keys
- **Partitioner**
  - Control which reducer processes which keys
- **Preserving state in mappers and reducers**
  - Capture dependencies across multiple keys and values

# Communication hotspot: Transmitting Mapper results to the reducer



# Word Count: Baseline

```
1: class MAPPER
2: method MAP(docid a, doc d)
3: for all term t ∈ doc d do
4: EMIT(term t, count 1)
```

## PIAT

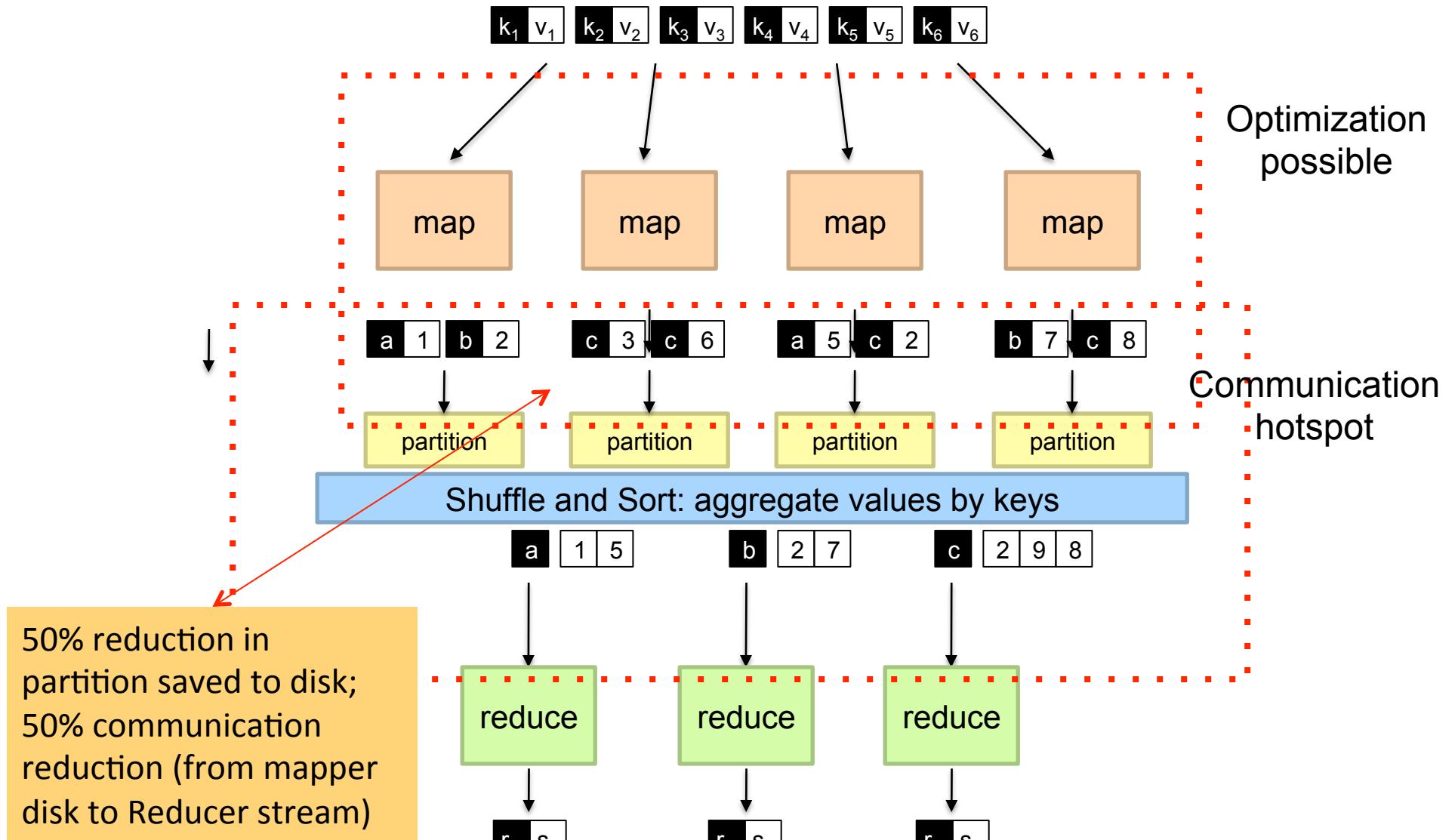
1. In mem: Partition, sort, ~~combine~~
2. Spill to Disk
3. Merge sorted spills, ~~combine~~
4. Once mapper finishes:
  - Transfer to reducer bandwidth

```
1: class REDUCER
2: method REDUCE(term t, counts [c1, c2, ...])
3: sum ← 0
4: for all count c ∈ counts [c1, c2, ...] do
5: sum ← sum + c
6: EMIT(term t, count s)
```

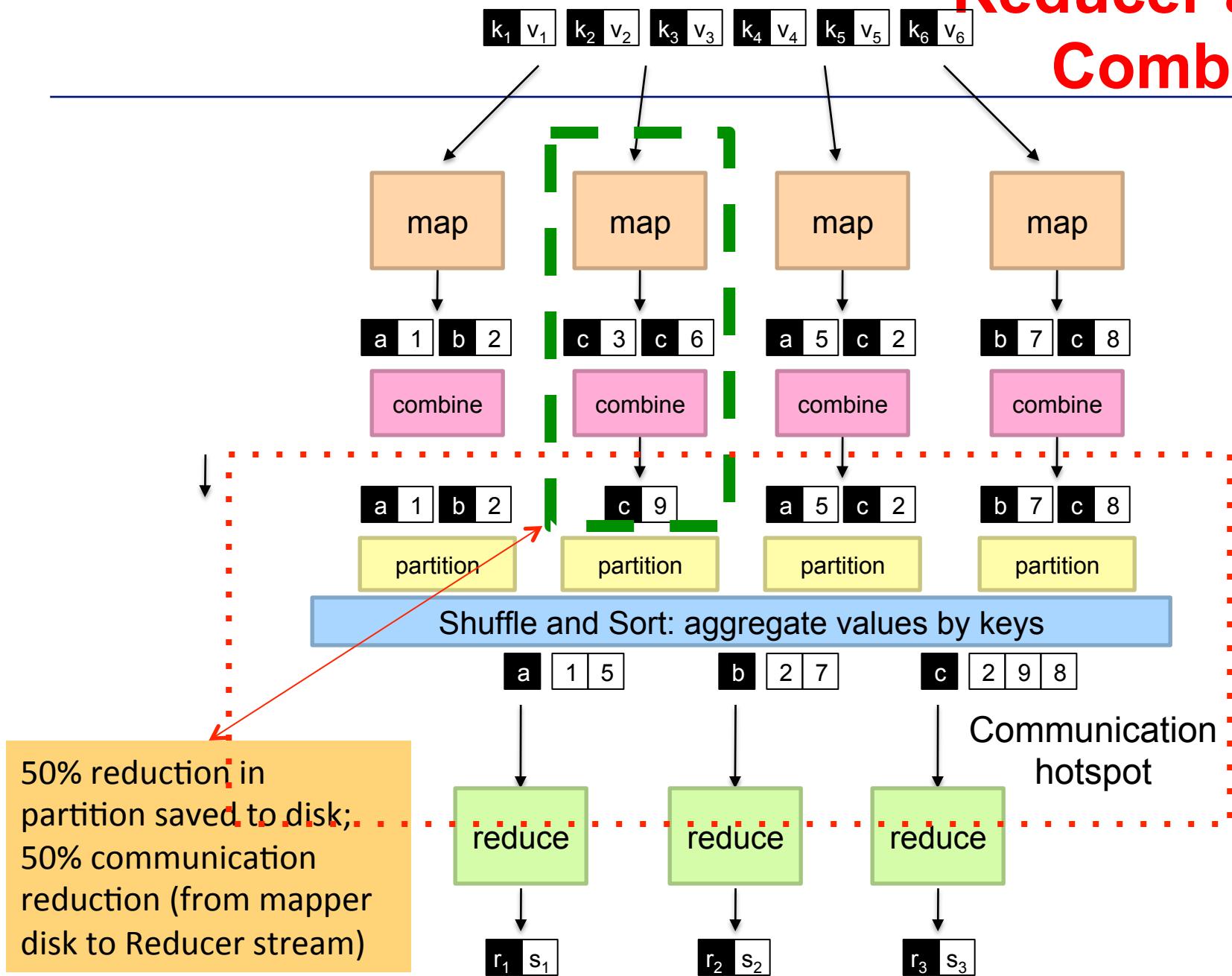
Emit each word...heavy disk storage requirements on Mapper side;  
big sorts; big transfers from mapper to reducer

What's the impact of combiners?

# Communication hotspot: Transmitting Mapper results to the reducer



# Reducer as a Combiner



# Word Count: Baseline+mini-reducer as a combiner

```
1: class MAPPER
2: method MAP(docid a, doc d)
3: for all term t ∈ doc d do
4: EMIT(term t, count 1)
```

## PIAT

1. In mem: Partition, sort, combine
2. Spill to Disk
3. Merge sorted spills, combine
4. Once mapper finishes:
  - Transfer to reducer bandwidth

```
1: class REDUCER
2: method REDUCE(term t, counts [c1, c2, ...])
3: sum ← 0
4: for all count c ∈ counts [c1, c2, ...] do
5: sum ← sum + c
6: EMIT(term t, count s)
```

Emit each word...heavy disk storage requirements on Mapper side;  
big sorts; big transfers from mapper to reducer

What's the impact of combiners?

# Other forms of local optimization

---

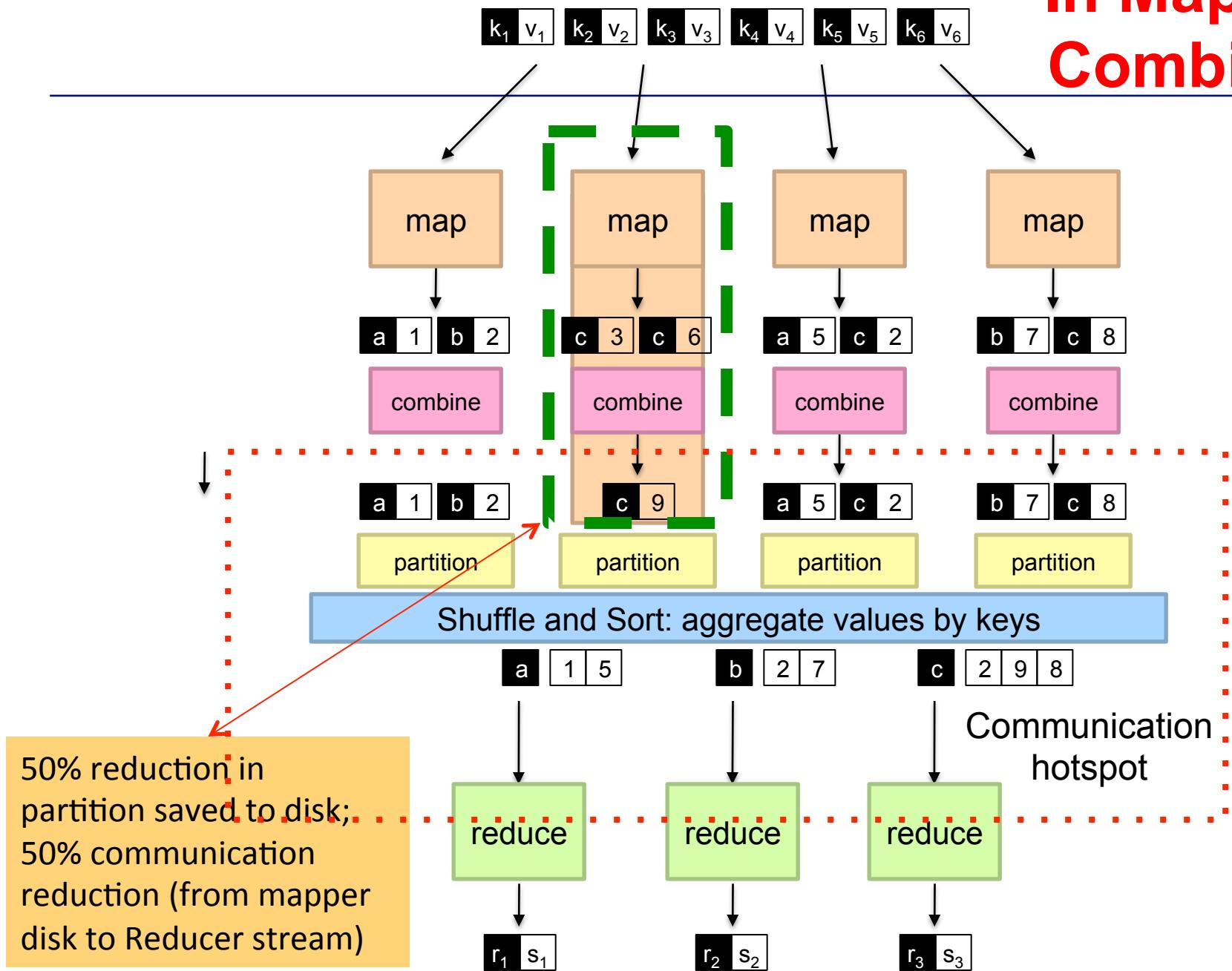
- Data has gravity
- It is heavy to move around
- In memory mappers and in-memory reducers?

# Mappers are Java objects: with init, and state

---

- In Hadoop, mappers are Java objects with a Map method (among others).
- A mapper object is instantiated for every map task by the tasktracker.
- The life-cycle of this object begins with instantiation, where a hook is provided in the API to run programmer-specified code.
  - This means that mappers can read in “side data”, providing an opportunity to load state, static data sources, dictionaries, etc.
  - After initialization, the Map method is called (by the execution framework) on all key-value pairs in the input split.
  - Since these method calls occur in the context of the same Java object, it is possible to preserve state across multiple input key-value pairs within the same map task|this is an important property to exploit in the design of MapReduce algorithms, as we will see in the next chapter.
- After all key-value pairs in the input split have been processed, the mapper object provides an opportunity to run programmer-specified termination code. This, too, will be important in the design of MapReduce algorithms.
- The actual execution of reducers is similar to that of the mappers.

# In Mapper Combiner



# Word Count: Version 1: record-level combiner

---

```
1: class MAPPER
2: method MAP(docid a, doc d)
3: H ← new ASSOCIATIVEARRAY
4: for all term t ∈ doc d do
5: H{t} ← H{t} + 1 ▷ Tally counts for entire document
6: for all term t ∈ H do
7: EMIT(term t, count H{t})
```

Issues

1. Disk
2. CPU
3. Network

If the same output key appears multiple times in a record,  
e.g., the same word appears multiple times

Are combiners still needed?

Scales very well but still have the same issues as the baseline  
mapper-reducer solution

# Word Count: Version 1: record-level combiner

Within each document combiner

```
1: class MAPPER
2: method MAP(docid a, doc d)
3: H ← new ASSOCIATIVEARRAY
4: for all term t ∈ doc d do
5: H{t} ← H{t} + 1 ▷ Tally counts for entire document
6: for all term t ∈ H do
7: EMIT(term t, count H{t})
```

Issues

1. Disk
2. CPU
3. Network

Are combiners still needed?  
Scales very well but still have the  
mapper-reducer solution

Yes

There are some savings here but there is still a  
lot of instances of the same word in the  
stream that can be combined

# Word Count: Version 2: in-memory mapper combiner

```
1: class MAPPER
2: method INITIALIZE
3: $H \leftarrow$ new ASSOCIATIVEARRAY
4: method MAP(docid a , doc d)
5: for all term $t \in$ doc d do
6: $H\{t\} \leftarrow H\{t\} + 1$
7: method CLOSE
8: for all term $t \in H$ do
9: EMIT(term t , count $H\{t\}$)
```

NB: preserve state across  
input key-value pairs!

▷ Tally counts *across* documents

That is, emission of intermediate data is deferred until the Close method in the pseudo-code.

Recall that this API hook provides an opportunity to execute user-specified code after the Map method has been applied to all input key-value pairs of the input data split to which the map task was assigned

Resolves

1. Disk
2. CPU
3. Network

BUT

- Memory issues?

## Are combiners still needed?

# Word Count: Version 2: in-memory mapper combiner

```
1: class MAPPER
2: method INITIALIZE
3: $H \leftarrow$ new ASSOCIATIVEARRAY
4: method MAP(docid a , doc d)
5: for all term $t \in$ doc d do
6: $H\{t\} \leftarrow H\{t\} + 1$
7: method CLOSE
8: for all term $t \in H$ do
9: EMIT(term t , count $H\{t\}$)
```

NB: preserve state across  
input key-value pairs!

▷ Tally counts *across* documents

Resolves  
1. Disk  
2. CPU

That is, emission of intermediate data is defered until the Close method in the pseudo-code.

Recall that this API hook provides an opportunity to execute user-specified code after the Map method has been applied to all input key-value pairs in the input data split to which the map task was assigned.

Yes

On the reduce side combiners can be run to consolidate results before putting them into the REDUCE Stream

Are combiners still needed?

# Design Pattern for Local Aggregation

---

- “**In-mapper combining**”
  - Fold the functionality of the combiner into the mapper by preserving state across multiple map calls
- **Advantages**
  - Speed
  - Why is this faster than actual combiners?
- **Disadvantages**
  - Explicit memory management required
  - Potential for order-dependent bugs

# L3: Map Reduce Algorithm Design

- 3.1 Background
  - RAM vs. disk vs. bandwidth (5)
  - Priority Queues and Merge sort (5)
  - The internals of the Hadoop Shuffle (15)
- 3.2 Local Aggregation
  - Combiners and In-Mapper Combining (15)
  - Algorithmic Correctness with Local Aggregation (15)
- 3.3 Pairs and Stripes (15)
- 3.4 Computing Relative Frequencies (10)
- 3.4 Secondary Sorting (5) (85)
- 3.5 Scaling Tricks (5)
- END of Lecture

V2

- 
- **Combiners can bring huge gains whether they come in the form in-memory mappers or just as a separate combiner.**
  - **So how can we design combiner?**

# Associative (grouping) Property

---

- Within an expression containing two or more occurrences in a row of the same associative operator, the order in which the operations are performed does not matter as long as the sequence of the operands is not changed.
- That is, rearranging the parentheses in such an expression will not change its value.

$$(2 + 3) + 4 = 2 + (3 + 4) = 9$$

$$2 \times (3 \times 4) = (2 \times 3) \times 4 = 24$$

Change the grouping of the data

# Commutative Property

---

In mathematics, a binary operation is commutative if changing the order of the operands does not change the result.

The term "commutative" is used in several related senses.<sup>[7][8]</sup>

1. A binary operation  $*$  on a set  $S$  is called *commutative* if:

$$x * y = y * x \quad \text{for all } x, y \in S$$

An operation that does not satisfy the above property is called **noncommutative**.

2. One says that  $x$  *commutes* with  $y$  under  $*$  if:

$$x * y = y * x$$

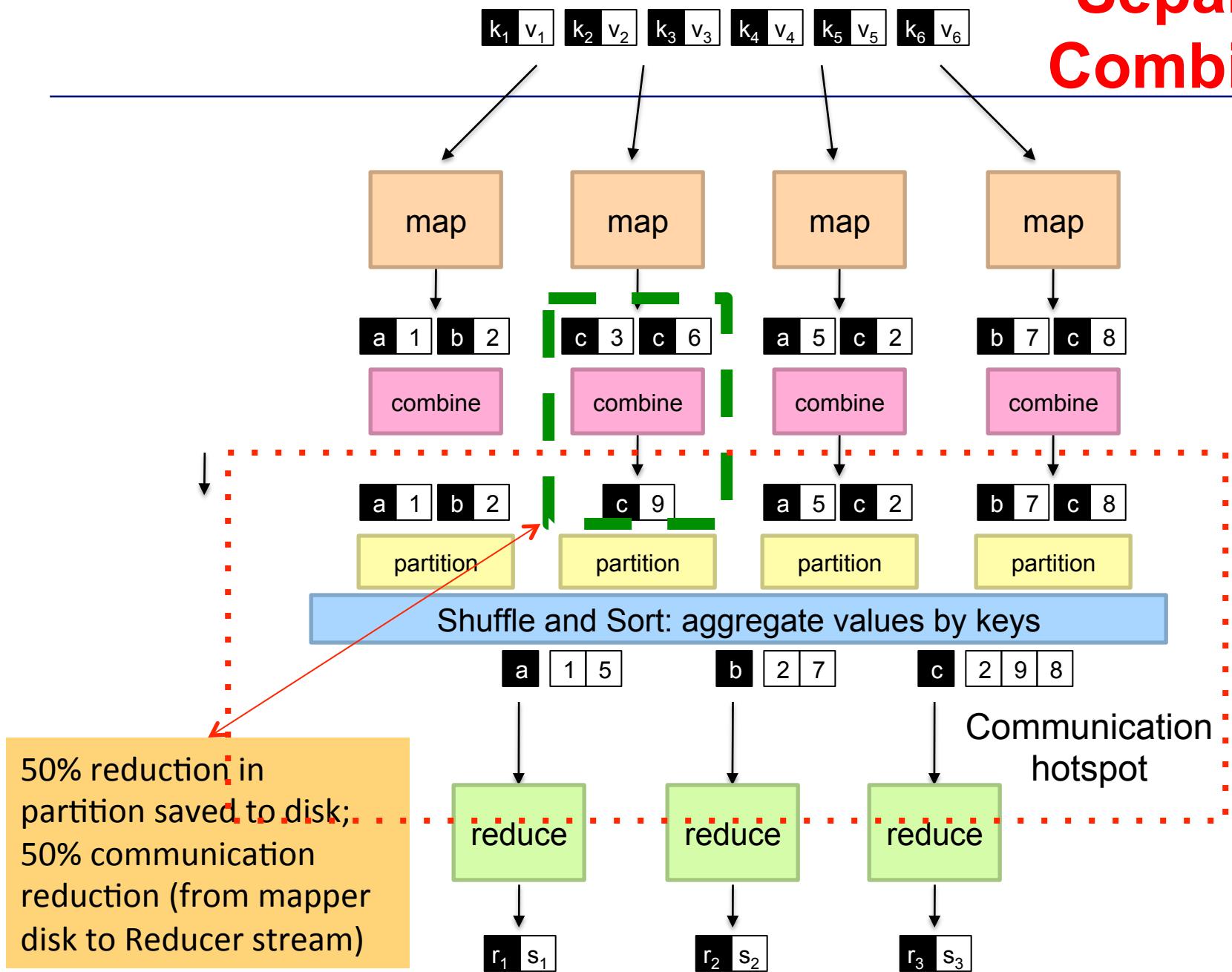
Change the order of the data

# Combiner Design

---

- **Combiners and reducers share same method signature**
  - Sometimes, reducers can serve as combiners
  - Often, not...
- **Remember: combiner are optional optimizations**
  - Should not affect algorithm correctness
  - Hadoop makes no guarantees: could be run 0, 1, or multiple times
- **Combiners need to be associative and commutative**
- **Example: find average of all integers associated with the same key**

# Separate Combiner



# combineByKey

---

- **combineByKey is designed for when your return type from the aggregation is different from the values being aggregated (e.g. you group together objects)**

# GroupByKey is expensive (vs mapside ops vs combiners)

---

- When working in a map-reduce framework such Spark or Hadoop one of the steps we can take to ensure maximum performance is to limit the amount of data sent across the network during the shuffle phase.
- The best option is when all operations can be performed on the map-side exclusively, meaning no data is sent at all to reducers.
- In most cases though, it's not going to be realistic to do map-side operations only. If you need to do any sort of grouping, sorting or aggregation you'll need to send data to reducers. But that doesn't mean we still can't attempt to make some optimizations.

# groupByKey is expensive

---

- In the case of a groupByKey call, every single key-value pair will be shuffled across the network with identical keys landing on the same reducer.
- To state the obvious, when grouping by key, the need for all matching keys to end up on the same reducer can't be avoided. But one optimization we can attempt is to combine/merge values so we end up sending fewer key-value pairs in total.
- Additionally, less key-value pairs means reducers won't have as much work to do, leading to additional performance gains.
- The groupByKey call makes no attempt at merging/combing values, so it's an expensive operation.

# **reduceByKey : returns an pair RDD (key, value)**

---

- **reduceByKey()** is quite similar to **reduce()**; both take a function and use it to combine values.
- **reduceByKey()** runs several parallel reduce operations, one for each key in the dataset, where each operation combines values that have the same key.
- Because datasets can have very large numbers of keys, **reduceByKey()** is not implemented as an action that returns a value to the user program.
- Instead, it returns a new RDD consisting of each key and the reduced value for that key.

| key    | value |
|--------|-------|
| panda  | 0     |
| pink   | 3     |
| pirate | 3     |
| panda  | 1     |
| pink   | 4     |

mapValues

| key    | value  |
|--------|--------|
| panda  | (0, 1) |
| pink   | (3, 1) |
| pirate | (3, 1) |
| panda  | (1, 1) |
| pink   | (4, 1) |

reduceByKey will automatically do a local combine

reduceByKey

| key    | value  |
|--------|--------|
| panda  | (1, 2) |
| pink   | (7, 2) |
| pirate | (3, 1) |

Per-key average with reduceByKey() and mapValues() in Python

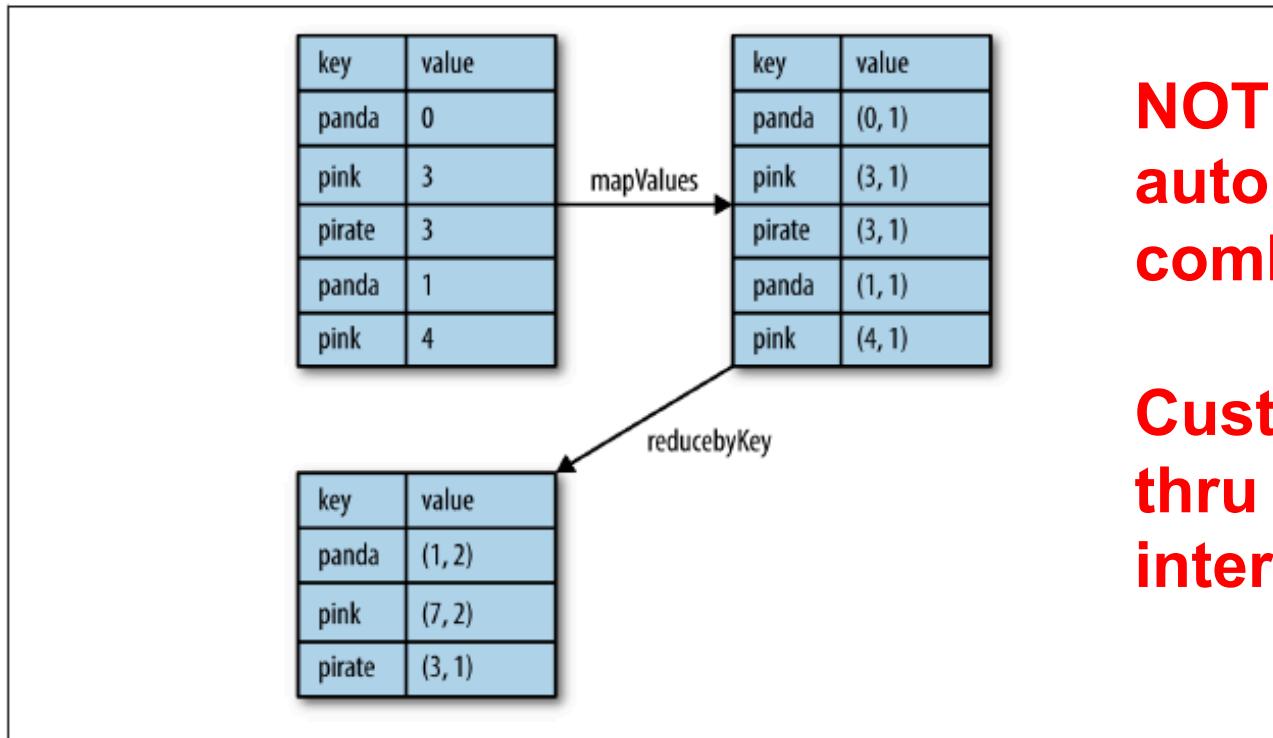
`rdd.mapValues(lambda x: (x, 1)).reduceByKey( lambda x, y: (x[0] + y[0], x[1] + y[1]))`

*Example 4-7. Per-key average with reduceByKey() and mapValues() in Python*

```
rdd.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
```

*Example 4-8. Per-key average with reduceByKey() and mapValues() in Scala*

```
rdd.mapValues(x => (x, 1)).reduceByKey((x, y) => (x._1 + y._1, x._2 + y._2))
```



*Figure 4-2. Per-key average data flow*



Those familiar with the combiner concept from MapReduce should note that calling `reduceByKey()` and `foldByKey()` will automatically perform combining locally on each machine before computing global totals for each key. The user does not need to specify a combiner. The more general `combineByKey()` interface allows you to customize combining behavior.

# Pair RDD for wordcount

---

We can use a similar approach in Examples 4-9 through 4-11 to also implement the classic distributed word count problem. We will use `flatMap()` from the previous chapter so that we can produce a pair RDD of words and the number 1 and then sum together all of the words using `reduceByKey()` as in Examples 4-7 and 4-8.

*Example 4-9. Word count in Python*

```
rdd = sc.textFile("s3://...")
words = rdd.flatMap(lambda x: x.split(" "))
result = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
```

Since each partition is processed independently, we can have multiple accumulators for the same key. When we are merging the results from each partition, if two or more partitions have an accumulator for the same key we merge the accumulators using the user-supplied `mergeCombiners()` function.

### Customize the combiner thru `combineByKey` interface

We can disable map-side aggregation in `combineByKey()` if we know that our data won't benefit from it. For example, `groupByKey()` disables map-side aggregation as the aggregation function (appending to a list) does not save any space. If we want to disable map-side combines, we need to specify the partitioner; for now you can just use the partitioner on the source RDD by passing `rdd.partition`.

Since `combineByKey()` has a lot of different parameters it is a great candidate for an explanatory example. To better illustrate how `combineByKey()` works, we will look at computing the average value for each key, as shown in Examples 4-12 through 4-14 and illustrated in Figure 4-3.

*Example 4-12. Per-key average using `combineByKey()` in Python*

```
sumCount = nums.combineByKey((lambda x: (x,1)),
 (lambda x, y: (x[0] + y, x[1] + 1)),
 (lambda x, y: (x[0] + y[0], x[1] + y[1])))
Large Sc sumCount.map(lambda key, xy: (key, xy[0]/xy[1])).collectAsMap()
```

# combineByKey sample data flow

Partition 1

|        |   |
|--------|---|
| coffee | 1 |
| coffee | 2 |
| panda  | 3 |

Partition 2

|        |   |
|--------|---|
| coffee | 9 |
|--------|---|

```
def createCombiner(value):
 (value, 1)
```

```
def mergeValue(acc, value):
 (acc[0] + value, acc[1] + 1)
```

```
def mergeCombiners(acc1, acc2):
 (acc1[0] + acc2[0], acc1[1] + acc2[1])
```

Partition 1 trace:

(coffee, 1) -> new key

accumulators[coffee] = createCombiner(1)

(coffee, 2) -> existing key

accumulators[coffee] = mergeValue(accumulators[coffee], 2)

(panda, 3) -> new key

accumulators[panda] = createCombiner(3)

Partition 2 trace:

(coffee, 9) -> new key

accumulators[coffee] = createCombiner(9)

Merge Partitions:

```
mergeCombiners(partition1.accumulators[coffee],
 partition2.accumulators[coffee])
```

# All actions available on the Base RDD pls ....

## Actions Available on Pair RDDs

As with the transformations, all of the traditional actions available on the base RDD are also available on pair RDDs. Some additional actions are available on pair RDDs to take advantage of the key/value nature of the data; these are listed in **Table 4-3**.

*Table 4-3. Actions on pair RDDs (example ( $\{(1, 2), (3, 4), (3, 6)\}$ ))*

| Function       | Description                                         | Example            | Result                      |
|----------------|-----------------------------------------------------|--------------------|-----------------------------|
| countByKey()   | Count the number of elements for each key.          | rdd.countByKey()   | $\{(1, 1), (3, 2)\}$        |
| collectAsMap() | Collect the result as a map to provide easy lookup. | rdd.collectAsMap() | Map{(1, 2), (3, 4), (3, 6)} |
| lookup(key)    | Return all values associated with the provided key. | rdd.lookup(3)      | [4, 6]                      |

# Parallelism

---

In a distributed program, communication is very expensive, so laying out data to minimize network traffic can greatly improve performance.

Spark's partitioning is available on all RDDs of key/value pairs, and causes the system to group elements based on a function of each key

Sometimes, we want to change the partitioning of an RDD outside the context of grouping and aggregation operations. For those cases, Spark provides the repartition() function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation. Spark also has an optimized version of repartition() called coalesce() that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

*Example 4-15. reduceByKey() with custom parallelism in Python*

```
data = [("a", 3), ("b", 4), ("a", 1)]
sc.parallelize(data).reduceByKey(lambda x, y: x + y) # Default parallelism
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10) # Custom parallelism
```

# Partitioning the data

---

*Example 4-15. reduceByKey() with custom parallelism in Python*

```
data = [("a", 3), ("b", 4), ("a", 1)]
sc.parallelize(data).reduceByKey(lambda x, y: x + y) # Default parallelism
sc.parallelize(data).reduceByKey(lambda x, y: x + y, 10) # Custom parallelism
```

*Example 4-16. reduceByKey() with custom parallelism in Scala*

```
val data = Seq(("a", 3), ("b", 4), ("a", 1))
sc.parallelize(data).reduceByKey((x, y) => x + y) // Default parallelism
sc.parallelize(data).reduceByKey((x, y) => x + y) // Custom parallelism
```

Sometimes, we want to change the partitioning of an RDD outside the context of grouping and aggregation operations. For those cases, Spark provides the `repartition()` function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation. Spark also has an optimized version of `repartition()` called `coalesce()` that allows avoiding data movement, but only if you are decreasing the number of RDD partitions. To know whether you can safely call `coalesce()`, you can check the size of the RDD using `rdd.partitions.size()` in Java/Scala and `rdd.getNumPartitions()` in Python and make sure that you are coalescing it to fewer partitions than it currently has.

# Repartition means shuffling the data across the network: EXPENSIVE!

---

For those cases, Spark provides the repartition() function, which shuffles the data across the network to create a new set of partitions. Keep in mind that repartitioning your data is a fairly expensive operation.

Spark also has an optimized version of repartition() called coalesce() that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

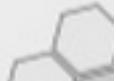
To know whether you can safely call coalesce(), you can check the size of the RDD using rdd.partitions.size() in Java/Scala and rdd.getNumPartitions() in Python and make sure that you are coalescing it to fewer partitions than it currently has.

Grouping Data  
With keyed

---

Was that comprehensive list really necessary?

Remember in MapReduce you only get two operators - map and reduce; so maybe I'm just excited at the 80+ operations in Spark!



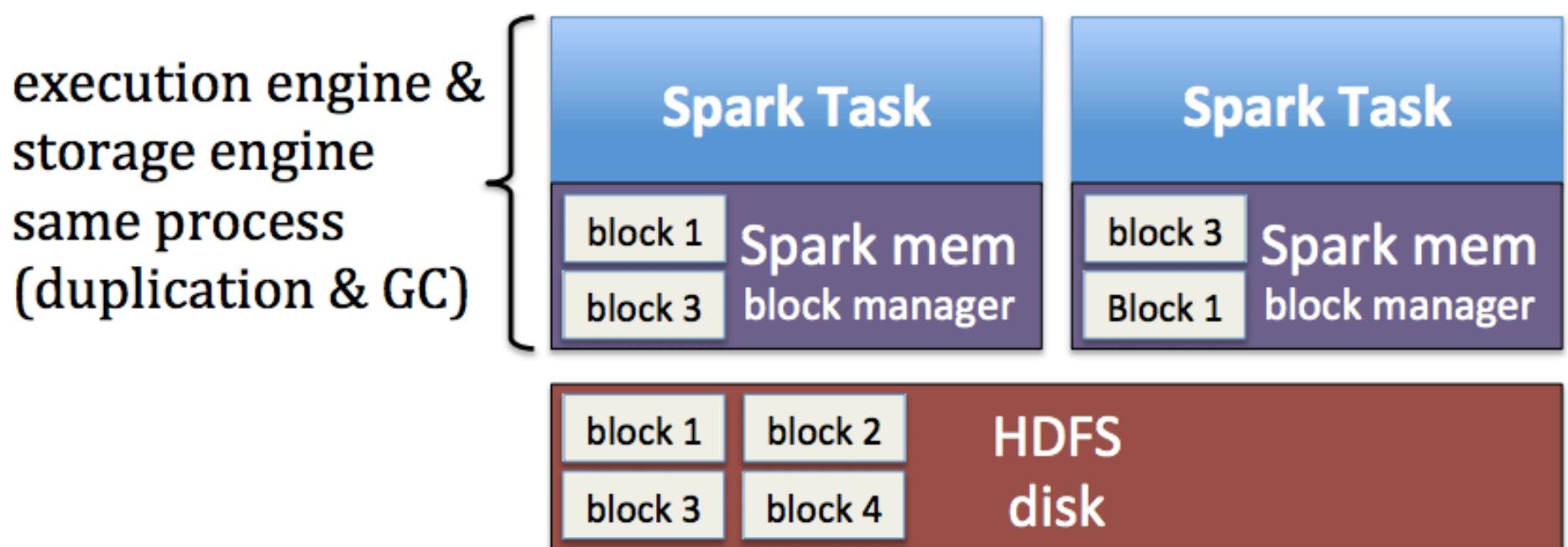
# Tachyon

---

- Tachyon is a memory-centric distributed storage system enabling reliable data sharing at memory-speed across cluster frameworks, such as Spark and MapReduce.
- It achieves high performance by leveraging lineage information and using memory aggressively.
- Tachyon caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

# Limitations of Spark, Hadoop

*duplicate memory per job & GC*



Computation Frameworks  
(Spark, MapReduce, Impala, Tez, ...)

Tachyon

Existing Storage Systems  
(HDFS, S3, GlusterFS, ...)

# Tachyon

---

- Tachyon is a memory-centric distributed storage system enabling reliable data sharing at memory-speed across cluster frameworks, such as Spark and MapReduce.
- It achieves high performance by leveraging lineage information and using memory aggressively.
- Tachyon caches working set files in memory, thereby avoiding going to disk to load datasets that are frequently read. This enables different jobs/queries and frameworks to access cached files at memory speed.

# Patterns

---

- **Map-Reduce**
- **CombineBy**

# Part 3: High level Machine Learning in Spark

- ML Overview
- Data Frames
- MLLib
- Pipelines

# Types of Learning

---

- Supervised learning - Generates a function that maps inputs to desired outputs. For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- Unsupervised learning - Models a set of inputs: like clustering
- Semi-supervised learning - Combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- Reinforcement learning - Learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.  
Transduction - Tries to predict new outputs based on training inputs, training outputs, and test inputs.

# Machine Learning Background

Machine Learning (ML): "a computer program that improves its performance at some task through experience" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)

GOAL: Approximate  $f(x_1, \dots, x_n) \rightarrow y$

Y is categorical

| Instance\Attr | $x_1$ | $x_2$ | ... | $x_n$ | $y$ |
|---------------|-------|-------|-----|-------|-----|
| 1             | 3     | 0     | ..  | 7     | -1  |
| 2             |       |       |     |       | +1  |
| ...           | ...   | ...   | ... | ...   | ... |
| L (aka m)     | 0     | 4     | ... | 8     | -1  |

# Machine Learning: Regression

Machine Learning (ML): "a computer program that improves its performance at some task through experience" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)

GOAL: Approximate  $f(x_1, \dots, x_n) \rightarrow y$

Y is real valued

| Instance\Attr | $x_1$ | $x_2$ | ... | $x_n$ | $y$ |
|---------------|-------|-------|-----|-------|-----|
| 1             | 3     | 0     | ..  | 7     | 73  |
| 2             |       |       |     |       | 76  |
| ...           | ...   | ...   | ... | ...   | ... |
| L (aka m)     | 0     | 4     | ... | 8     | 97  |

# Machine Learning semi-supervised

Machine Learning (ML): "a computer program that improves its performance at some task through experience" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)

GOAL: Approximate  $f(x_1, \dots, x_n) \rightarrow y$

Y is only partially available

| Instance\Attr | $x_1$ | $x_2$ | ... | $x_n$ | $y$           |
|---------------|-------|-------|-----|-------|---------------|
| 1             | 3     | 0     | ..  | 7     | 73            |
| 2             |       |       |     |       | <del>76</del> |
| ...           | ...   | ...   | ... | ...   | ...           |
| L (aka m)     | 0     | 4     | ... | 8     | 97            |

# Machine Learning Unsupervised

Machine Learning (ML): "a computer program that improves its performance at some task through experience" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)

GOAL: Approximate  $f(x_1, \dots, x_n) \rightarrow y$

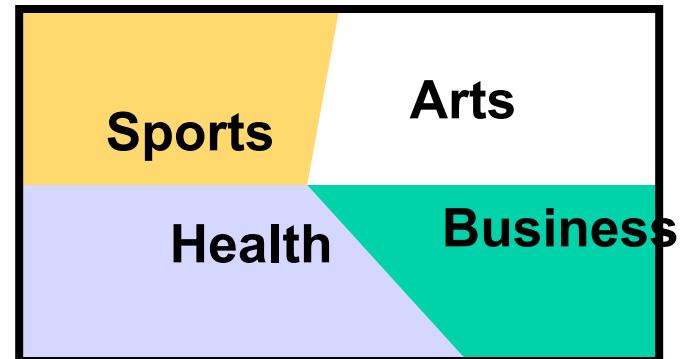
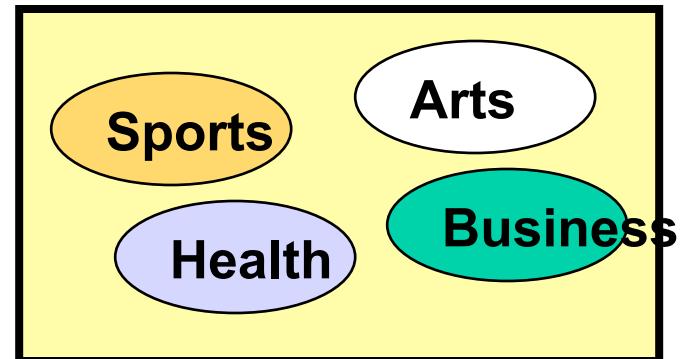
Y is not available

| Instance\Attr | $x_1$ | $x_2$ | ... | $x_n$ | $y$ |
|---------------|-------|-------|-----|-------|-----|
| 1             | 3     | 0     | ..  | 7     | 73  |
| 2             |       |       |     |       | 76  |
| ...           | ...   | ...   | ... | ...   | ... |
| L (aka m)     | 0     | 4     | ... | 8     | 97  |

# Families of Supervised Learning

---

- **Generative Classifier  
(Bottom-up learning)**
  - Build model of each class
  - Assume the underlying form of the classes and estimate their parameters (e.g., a Gaussian)
- **Discriminative Classifier  
(Top down)**
  - Build model of boundary between classes
  - Assume the underlying form of the discriminant and estimate its parameters (e.g., a hyperplane)



# Generative vs. Discriminative

---

- **Generative learning (e.g., Bayesian Networks, HMM, Naïve Bayes, EM GMM) typically more flexible**
  - More complex problems
  - More flexible predictions
- **Discriminative learning (e.g., ANN, SVM) typically more accurate**
  - Better with small datasets
  - Faster to train

# Parametric vs. Non-Parametric ML Algorithms

---

- **Parametric ML Algorithms (e.g., OLS, Decision Trees; SVMs, NNs)**
  - Model-based methods, such as neural networks and the mixture of Gaussians, use the data to build a parameterized model. After training, the model is used for predictions and the data are generally discarded.
- **Non-Parametric (`lowess()`; `knn`; some flavours of SVMs)**
  - In contrast, “memory-based” methods are non-parametric approaches that explicitly retain the training data, and use it each time a prediction needs to be made.
  - The term “non-parametric” (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis/model grows linearly with the size of the training set.

# Tutorial Outline

- **Part 1: Introduction**
  - Welcome Survey
  - Install Spark
  - Background and motivation
- **Part 2: Spark Intro and basics**
  - fundamental Spark concepts, including Spark Core, data frames, the Spark Shell, Spark Streaming, Spark SQL and vertical libraries such as MLlib and GraphX;
- **Part 3: Machine learning in Spark**
  - will focus on hands-on algorithmic design and development with Spark developing algorithms from scratch such as linear regression, logistic regression, graph processing algorithms such as pagerank/shortest path, etc.
- **Part 4: Wrap up**
  - Spark 1.5 and beyond
  - Summary

# Machine Learning in Spark

---

- **Data Frames**
- **MLLib**
- **Write your own algos**
  - Linear regression (Ridge and Lasso)
  - Logistic Regression
- **Pipelines**
- **R and Spark**
  - Linear Regression example
- **Graphs in Spark**
- **Case studies**
  - Flight delay
  - Mobile advertising
  - Social Networks



# DataFrames (new API introduced in 2/2015)

---

- A DataFrame is a distributed collection of data organized into named columns.
- It is conceptually equivalent to a table in a relational database or a data frame in R/Python, or a table in MySql, but with richer optimizations under the hood.
- DataFrames can be constructed from a wide array of sources such as:
  - structured data files, tables in Hive,
  - external databases, or existing RDDs
  - JSON, Parquet and more
- The DataFrame API is available in Scala, Java, Python, and R.

- 
- **The entry point into all relational functionality in Spark is the SQLContext class, or one of its decedents. To create a basic SQLContext, all you need is a SparkContext.**

Branch: master ▾

## [spark](#) / [examples](#) / [src](#) / [main](#) / [resources](#) / **people.json**



yhuai on Jun 17, 2014 [SPARK-2060][SQL] Querying JSON Datasets with SQL and DSL in Spark SQL

1 contributor

4 lines (3 sloc) | 0.073 kB

Raw

```
1 {"name":"Michael"}
2 {"name":"Andy", "age":30}
3 {"name":"Justin", "age":19}
```

- 
- As an example, the following creates a DataFrame based on the content of a JSON file:

Scala

Java

Python

R

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.read.json("examples/src/main/resources/people.json")

Displays the content of the DataFrame to stdout
df.show()
```

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

Create the DataFrame
df = sqlContext.read.json("examples/src/main/resources/people.json")

Show the content of the DataFrame
df.show()
age name
null Michael
30 Andy
19 Justin

Print the schema in a tree format
df.printSchema()
root
|-- age: long (nullable = true)
|-- name: string (nullable = true)

Select only the "name" column
df.select("name").show()
name
Michael
Andy
Justin
```

```
Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
name (age + 1)
Michael null
• ## Andy 31
Justin 20

Select people older than 21
df.filter(df['age'] > 21).show()
age name
30 Andy

Count people by age
df.groupBy("age").count().show()
age count
null 1
19 1
30 1
```

# JSON Datasets

[Scala](#)[Java](#)[Python](#)[R](#)[Sql](#)

```
CREATE TEMPORARY TABLE jsonTable
USING org.apache.spark.sql.json
OPTIONS (
 path "examples/src/main/resources/people.json"
)

SELECT * FROM jsonTable
```

# DataFrames

```
{ "name": "Michael"}
people.json
 { "name": "Andy", "age": 30}
 { "name": "Justin", "age": 19}
```

Load data from json file and show the schema

```
df = sqlContext.read.json("people.json")
df.show()
```

```
+----+-----+
| age| name |
+----+-----+
null	Michael
30	Andy
19	Justin
+----+-----+
```

```
df.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

# DataFrames

```
: df.select("name").show()
```

```
+-----+
| name |
+-----+
| Michael |
| Andy |
| Justin |
+-----+
```

```
: df.select(df['name'], df['age'] + 1).show()
```

```
+-----+-----+
| name | (age + 1) |
+-----+-----+
Michael	null
Andy	31
Justin	20
+-----+-----+
```

```
: df.filter(df['age'] > 21).show()
```

```
+----+
| age | name |
+----+
| 30 | Andy |
+----+
```

```
: df.groupBy("age").count().show()
```

```
+----+-----+
| age | count |
+----+-----+
null	1
19	1
30	1
+----+-----+
```

# DataFrames

```
people.txt'
```

```
Michael, 29
Andy, 30
Justin, 19
```

## Load data from a text file

Load data into a PythonRDD and then transform an PythonRDD to an DataFrame

```
from pyspark.sql import Row
Load a text file and convert each line to a Row.
lines = sc.textFile("people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
```

Create an RDD

```
people
PythonRDD[152] at RDD at PythonRDD.scala:43
```

```
[Row(age=29, name=u'Michael'),
 Row(age=30, name=u'Andy'),
 Row(age=19, name=u'Justi...]
```

```
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople = people.toDF() is another way to create DataFrame
schemaPeople
```

Transform RDD to a  
dataframe

```
DataFrame[age: bigint, name: string]
```

```
[Row(age=29, name=u'Michael'),
 Row(age=30, name=u'Andy'),
 Row(age=19, name=u'Justi...]
```

# DataFrames

---

- Register the DataFrame as a table and then run SQL queries

```
Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

SQL can be run over DataFrames that have been registered as a table.
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

The results of SQL queries are RDDs and support all the normal RDD operations.
teenNames = teenagers.map(lambda p: "Name: " + p.name)
for teenName in teenNames.collect():
 print teenName
```

Name: Justin

# Spark SQL

---

## Spark SQL

Mix any SQL query with Python, Scala and Java

Unified data access

Compatible with Hive

Standard Connectivity

# Spark SQL

---

- Native SQL Interface to Spark
- Hive, DataFrame, JSON, RDD, etc...
- JDBC Server (B.I. Interface)
- UDFs (Spark SQL and Hive)
- Columnar storage, Predicate Pushdowns, Tuning options

---

# Spark SQL Example

---

- ```
hiveCtx = HiveContext(sc)
allData = hiveCtx.jsonFile(filein)
allData.registerTempTable("customers")

query1 = hiveCtx.sql("""
    SELECT last, first
    FROM customers
    ORDER BY last
    LIMIT 50")
```

Spark SQL

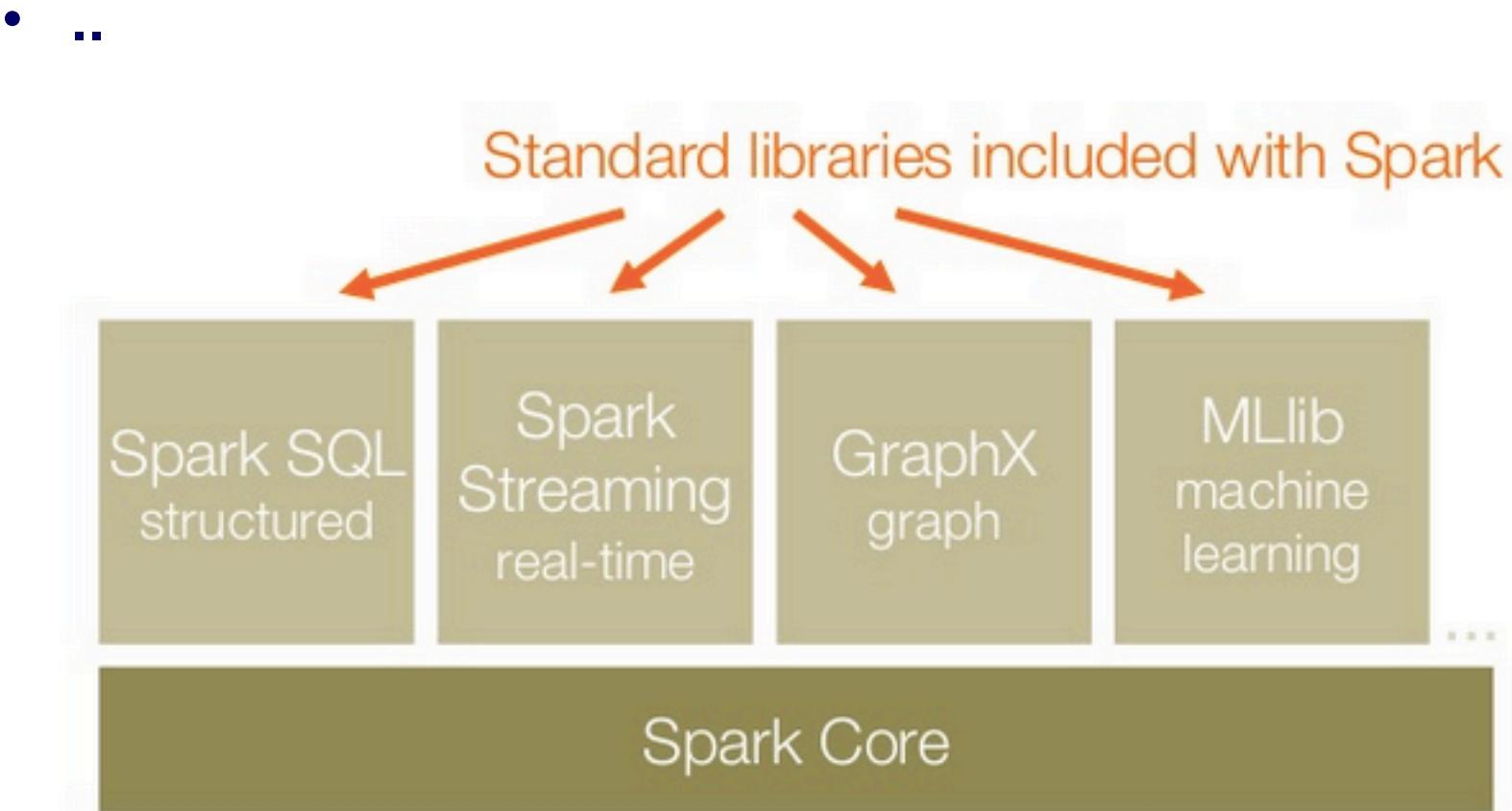
Part 3: Machine Learning in Spark

- Data Frames
- **MLLib**
- Write your own algos
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- Pipelines
- R and Spark
 - Linear Regression example
- Graphs in Spark
- Case studies
 - Flight delay
 - Mobile advertising
 - Social Networks

- **Machine Learning in Spark**

- Mllib
- Write your own!

Spark Machine Learning



Spark MLLib Functionality 1/2

MLlib is Apache Spark's scalable machine learning library

- **Supervised ML**

- linear SVM and logistic regression
- classification and regression tree
- multinomial naive Bayes
- random forest and gradient-boosted trees
- linear regression with L1- and L2-regularization
- isotonic regression

- **recommendation via alternating least squares**

- **Unsupervised learning**

- clustering via k-means, Gaussian mixtures, and power iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition

- **frequent itemset mining via FP-growth**

- **Statistics**

- Summary statistics, Correlation, Chi-square

- **Feature transformations**

Spark MLLib Functionality 2/2

- **Pipeline API (Train, Evaluation, Testing)**
 - EDA
 - Feature engineering
 - Data engineering
 - Learning
 - Hyperparameter tuning
 - Testing

-
- ..

Machine Learning Library (MLlib)

```
points = context.sql("select latitude, longitude from tweets")
model = KMeans.train(points, 10)
```

- **MLlib is Apache Spark's scalable machine learning library.**

<https://spark.apache.org/mllib/>

Algorithms

MLlib 1.3 contains the following algorithms:

- linear SVM and logistic regression
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- linear regression with L₁- and L₂-regularization
- isotonic regression
- multinomial naive Bayes
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

- | | | |
|--|-----|---|
| <ul style="list-style-type: none">• Data types• Basic statistics<ul style="list-style-type: none">◦ summary statistics◦ correlations◦ stratified sampling◦ hypothesis testing◦ random data generation• Classification and regression<ul style="list-style-type: none">◦ linear models (SVMs, logistic regression, linear regression)◦ naive Bayes◦ decision trees◦ ensembles of trees (Random Forests and Gradient-Boosted Trees)◦ isotonic regression• Collaborative filtering<ul style="list-style-type: none">◦ alternating least squares (ALS)• Clustering<ul style="list-style-type: none">◦ k-means◦ Gaussian mixture◦ power iteration clustering (PIC)◦ latent Dirichlet allocation (LDA)◦ streaming k-means• Dimensionality reduction<ul style="list-style-type: none">◦ singular value decomposition (SVD)◦ principal component analysis (PCA)• Feature extraction and transformation• Frequent pattern mining<ul style="list-style-type: none">◦ FP-growth◦ association rules◦ PrefixSpan• Optimization (developer)<ul style="list-style-type: none">◦ stochastic gradient descent◦ limited-memory BFGS (L-BFGS)• PMML model export | 1.4 | <ul style="list-style-type: none">• Data types, algorithms, and utilities• Basic statistics<ul style="list-style-type: none">◦ summary statistics◦ correlations◦ stratified sampling◦ hypothesis testing◦ streaming significance testing◦ random data generation• Classification and regression<ul style="list-style-type: none">◦ linear models (SVMs, logistic regression, linear regression)◦ naive Bayes◦ decision trees◦ ensembles of trees (Random Forests and Gradient-Boosted Trees)◦ isotonic regression• Collaborative filtering<ul style="list-style-type: none">◦ alternating least squares (ALS)• Clustering<ul style="list-style-type: none">◦ k-means◦ Gaussian mixture◦ power iteration clustering (PIC)◦ latent Dirichlet allocation (LDA)◦ bisecting k-means◦ streaming k-means• Dimensionality reduction<ul style="list-style-type: none">◦ singular value decomposition (SVD)◦ principal component analysis (PCA)• Feature extraction and transformation• Frequent pattern mining<ul style="list-style-type: none">◦ FP-growth◦ association rules◦ PrefixSpan• Evaluation metrics• PMML model export• Optimization (developer)<ul style="list-style-type: none">◦ stochastic gradient descent◦ limited-memory BFGS (L-BFGS) |
|--|-----|---|

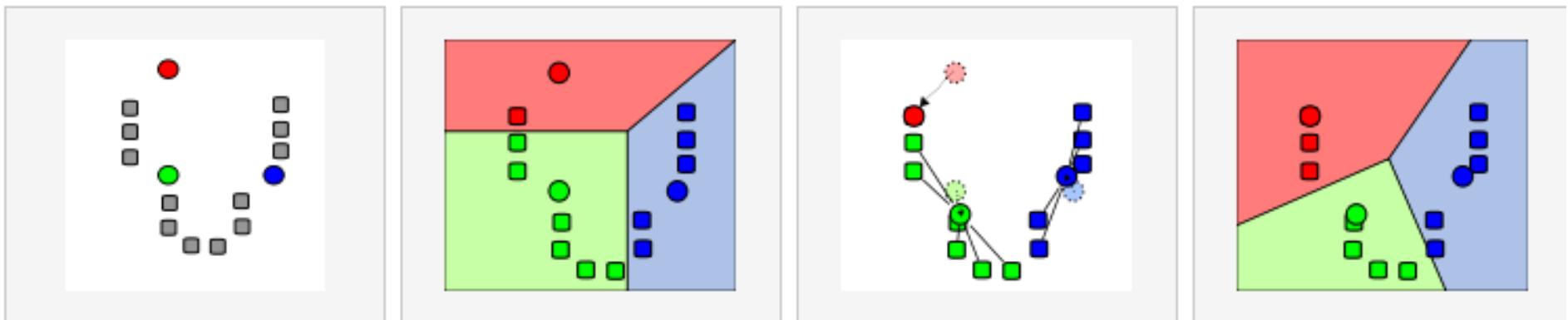
MLlib Example

Linear Regression Model Build

```
model = LinearRegressionWithSGD.train(  
    data,  
    iterations = 100,  
    intercept = True
```

K-Means Clustering

- k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.



1. k initial "means" (in this case $k=3$) are randomly generated within the data domain (shown in color).

2. k clusters are created by associating every observation with the nearest mean. The partitions here represent the [Voronoi diagram](#) generated by the means.

3. The [centroid](#) of each of the k clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Kmeans in MLLib

The following examples can be tested in the PySpark shell.

In the following example after loading and parsing data, we use the KMeans object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm. We then compute Within Set Sum of Squared Error (WSSSE). You can reduce this error measure by increasing k . In fact the optimal k is usually one where there is an "elbow" in the WSSSE graph.

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
    runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

Evaluate clustering by
computing Within Set Sum of
Squared Errors
Assign point to cluster centre

Gaussian Model: Estimate Parameters

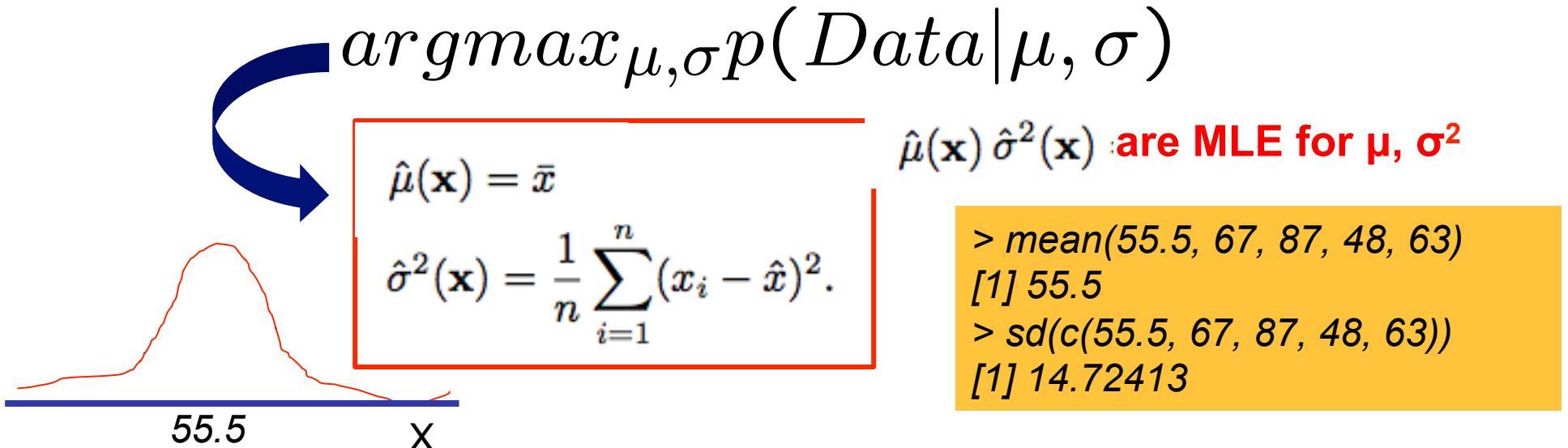
Suppose the following are marks in a course

55.5, 67, 87, 48, 63

Marks typically follow a Normal distribution whose density function is

$$N(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

Now, we want to find the best μ, σ such that



Maximum-likelihood estimation

- Maximum-likelihood estimation (MLE) is a method of estimating the parameters of a statistical model. When applied to a data set and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters.
- Expectation–maximization (EM) algorithm is an iterative method for finding maximum likelihood, where the model depends on unobserved latent variables.
 - The EM iteration alternates between performing
 - an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters,
 - and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the Estep.
 - These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

MLE Estimates for GMM

If we knew z: class assignments for each example

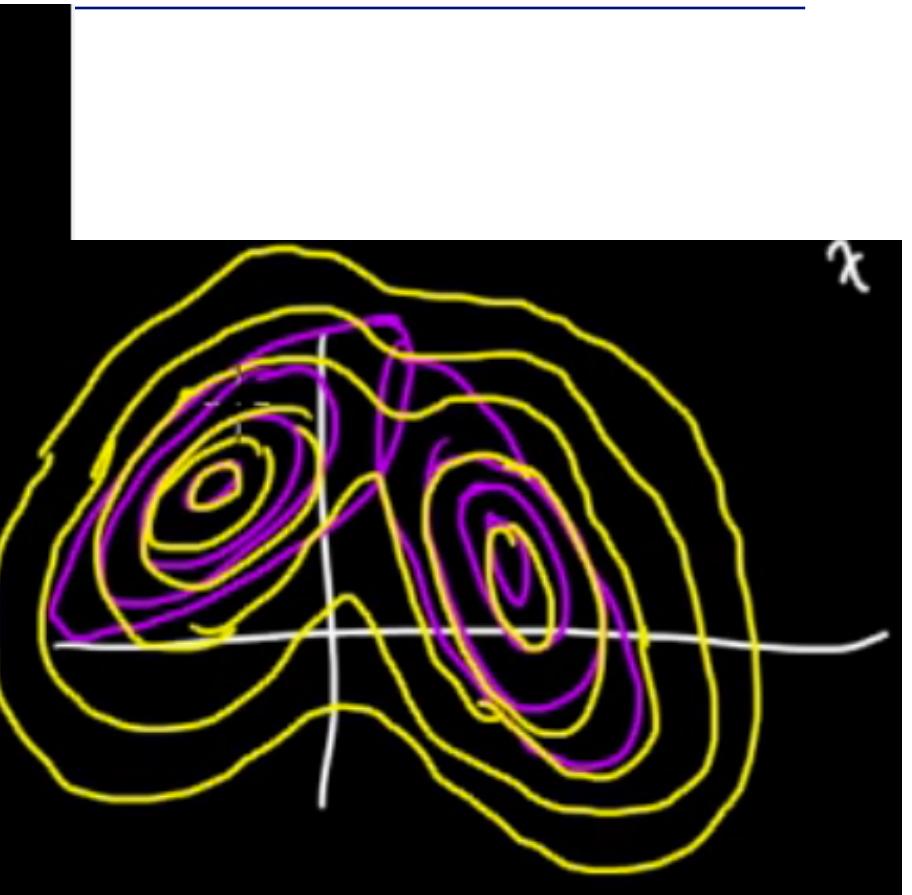
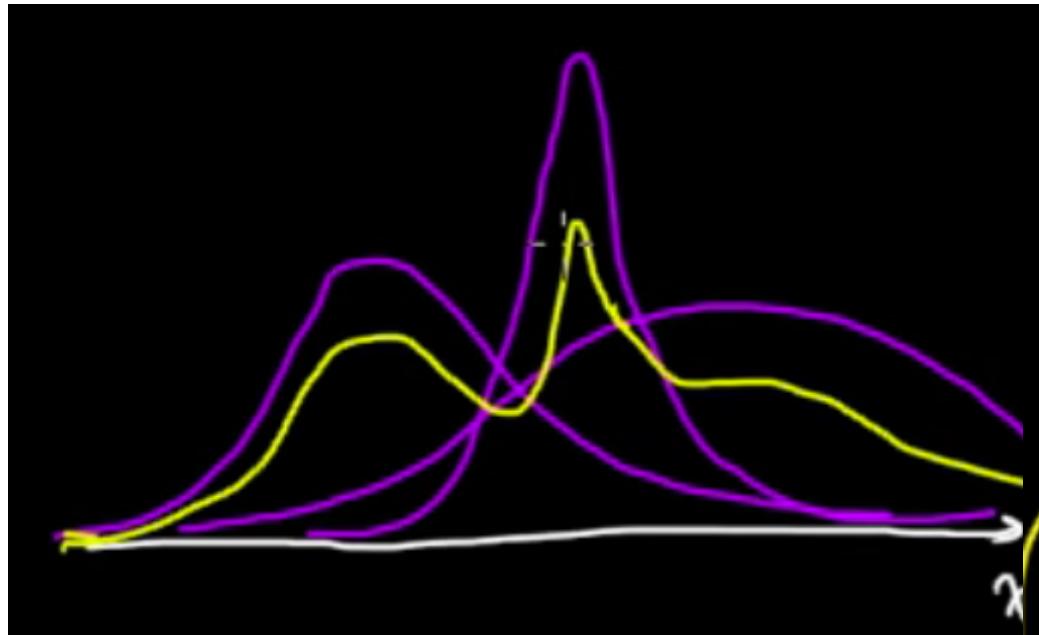
- likelihood problem would have been easy. Specifically, we could then write down the likelihood as

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^m \log p(x^{(i)} | z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi).$$

Maximizing this with respect to ϕ , μ and Σ gives the parameters:

$$\begin{aligned}\phi_j &= \frac{1}{m} \sum_{i=1}^m 1\{z^{(i)} = j\}, \\ \mu_j &= \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{z^{(i)} = j\}}, \\ \Sigma_j &= \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m 1\{z^{(i)} = j\}}.\end{aligned}$$

PDF of a mixture of Gaussians



[https://www.youtube.com/watch?
v=Rkl30Fr2S38](https://www.youtube.com/watch?v=Rkl30Fr2S38)

Yellow curve is convex combination of the individual Gaussian

But $z^{(i)}$'s are not known: so use EM

- However, in our density estimation problem, the $z^{(i)}$'s are not known.
- What can we do?
- The EM algorithm is an iterative algorithm that has two main steps.
 - Applied to our problem, in the E-step, it tries to “guess” the values of the $z_{(i)}$'s.
 - In the M-step, it updates the parameters of our model based on our guesses. Since in the M-step we are pretending that the guesses in the first part were correct, the maximization becomes easy. Here's the algorithm:

EM for GMM

Driver: Initialize μ, Σ, ϕ

- ..

Repeat until convergence: {

(E-step) For each i, j , set J class; I example index

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\text{phi} \quad \phi_j := \frac{1}{m} \sum_{i=1}^m w_j^{(i)},$$

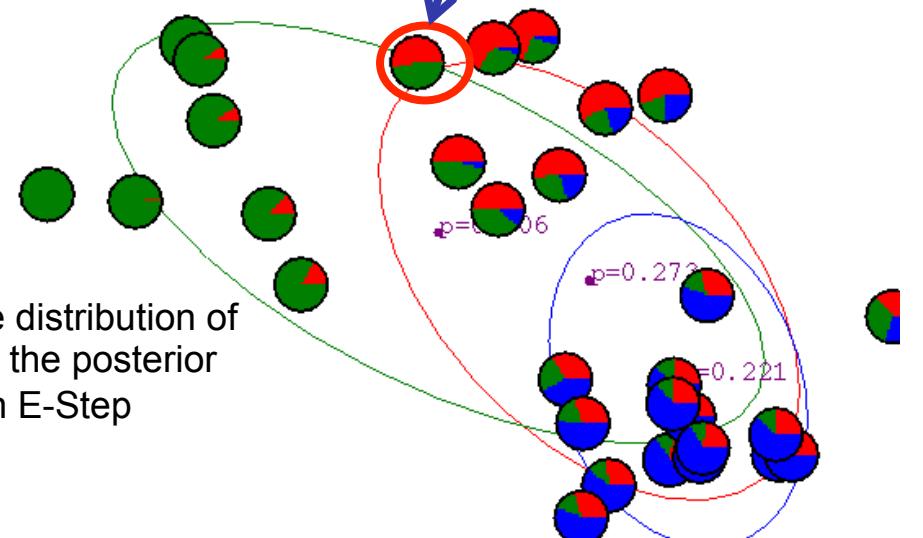
Compute relative distribution of each class using the posterior probabilities from E-Step

$$\mu_j := \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}},$$

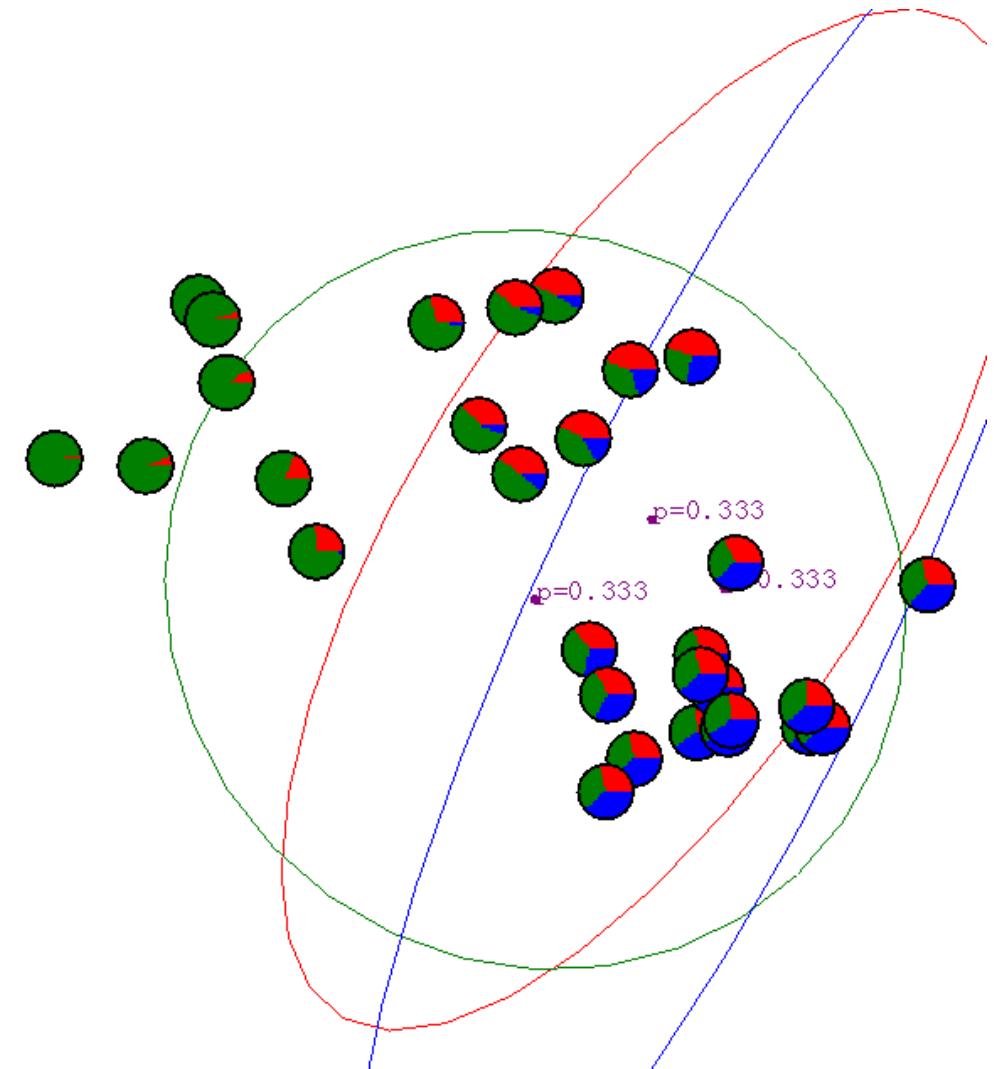
$$\Sigma_j := \frac{\sum_{i=1}^m w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m w_j^{(i)}}$$

E-Step

$\Pr(\text{Class} = \text{Red} | X) = W_{\text{red}} = 0.5;$
 $W_{\text{green}} = 0.5;$
 $W_{\text{blue}} = 0.0$



Random init: and assignment



EM for GMM: write the mapper

Commutative and associative ops are great candidates for Mapper/combiner

Driver: Initialize μ, Σ, ϕ

• ..

Repeat until convergence: {

(E-step) For each i, j , set J class; I example index

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\text{phi} \quad \phi_j := \frac{1}{m} \sum_{i=1}^m w_j^{(i)},$$

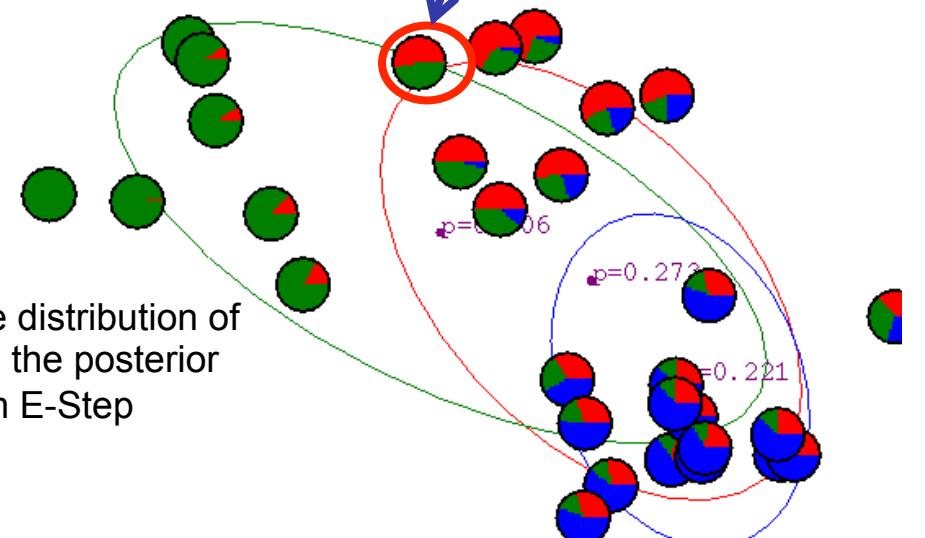
Compute relative distribution of each class using the posterior probabilities from E-Step

$$\mu_j := \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^m w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m w_j^{(i)}}$$

E-Step

$\Pr(\text{Class} = \text{Red} | X) = W_{\text{red}} = 0.5;$
 $W_{\text{green}} = 0.5;$
 $W_{\text{blue}} = 0.0$



EM for GMM: write the REDUCER

Commutative and associative ops are great candidate for Mapper/combiner
Partition key for reducer?

Driver: Initialize μ, Σ, ϕ

Partial sums for

- ..

Repeat until convergence:

(E-step) For each i, j , set $w_j^{(i)} := p(\text{Class } j | \text{Example index } i)$

μ , numerator, denominator

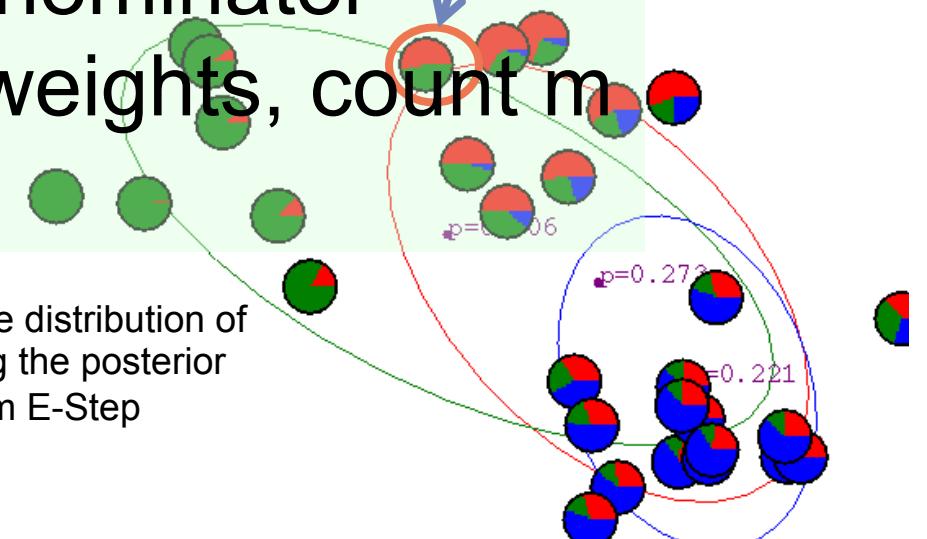
$$w_j^{(i)} := p(x^{(i)} = j | \omega, \phi, \mu, \Sigma)$$

Σ , numerator, denominator

ϕ partial sum of weights, count m

E-Step

$\Pr(\text{Class} = \text{Red} | X) = W_{\text{red}} = 0.5;$
 $W_{\text{green}} = 0.5;$
 $W_{\text{blue}} = 0.0$



(M-step) Update the parameters:

$$\phi_j := \frac{1}{m} \sum_{i=1}^m w_j^{(i)},$$

Compute relative distribution of each class using the posterior probabilities from E-Step

$$\mu_j := \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^m w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m w_j^{(i)}}$$

In the E-step, we calculate the posterior probability of our parameters the $z^{(i)}$'s, given the $x^{(i)}$ and using the current setting of our parameters. I.e., using Bayes rule, we obtain:

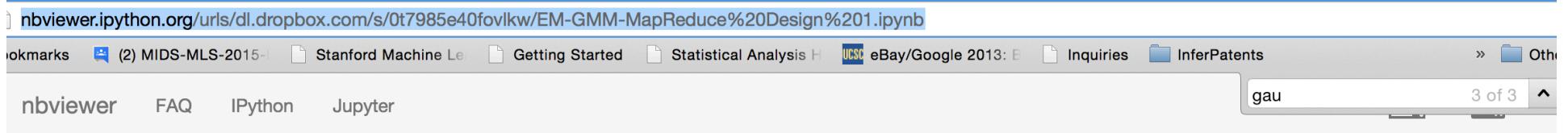
- $$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)p(z^{(i)} = j; \phi)}{\sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; \mu, \Sigma)p(z^{(i)} = l; \phi)}$$

Here, $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$ is given by evaluating the density of a Gaussian with mean μ_j and covariance Σ_j at $x^{(i)}$; $p(z^{(i)} = j; \phi)$ is given by ϕ_j , and so on. The values $w_j^{(i)}$ calculated in the E-step represent our “soft” guesses² for the values of $z^{(i)}$.

Also, you should contrast the updates in the M-step with the formulas we had when the $z^{(i)}$'s were known exactly. They are identical, except that instead of the indicator functions “ $1\{z^{(i)} = j\}$ ” indicating from which Gaussian each datapoint had come, we now instead have the $w_j^{(i)}$'s.

The EM-algorithm is also reminiscent of the K-means clustering algorithm, except that instead of the “hard” cluster assignments $c(i)$, we instead have the “soft” assignments $w_j^{(i)}$. Similar to K-means, it is also susceptible to local optima, so reinitializing at several different initial parameters may be a good idea.

EM Algorithm for GMM



Introduction

This is a map-reduce version of expectation maximization algo for a mixture of **Gaussians** model. There are two mrJob MR packages, `mr_GMixEmlterate` and `mr_GMixEmInitialize`. The driver calls the mrJob packages and manages the iteration.

E Step: Given mean vector and covariance matrix, calculate the probability of that each data point belongs to a class

P(Cluster_k| Xⁱ; θ)

$$p(\omega_k | \mathbf{x}^{(i)}, \theta) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Estimate class assignments (responsibilities) or weights
Use the current model to estimate the class assignment

M Step: Given probabilities, update mean and covariance

Centroid_k

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) \mathbf{x}^{(i)}$$

Centroid is just a weighted sum of soft assigned examples

Covariance_k

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)^T$$

Weighted covariance

Prior_k

$$\hat{\pi}_k = \frac{n_k}{n} \text{ where } n_k = \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta)$$

Class prior is based on the sum of the partial weights

Gaussian Mixture Model

2001 lines (2000 sloc) | 62.5 KB

```
1 2.59470454e+00 2.12298217e+00
2 1.15807024e+00 -1.46498723e-01
3 2.46206638e+00 6.19556894e-01
4 -5.54845070e-01 -7.24700066e-01
5 -3.23111426e+00 -1.42579084e+00
6 3.02978115e+00 7.87121753e-01
7 1.97365907e+00 1.15914704e+00
```

In the following example after loading and parsing data, we use a [GaussianMixture](#) object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm. We then output the parameters of the mixture model.

Refer to the [GaussianMixture Python docs](#) and [GaussianMixtureModel Python docs](#) for more details on the API.

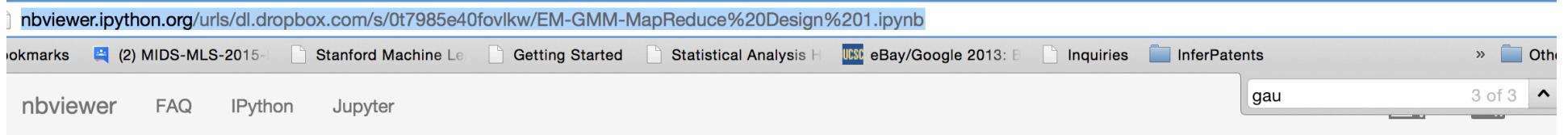
```
from pyspark.mllib.clustering import GaussianMixture
from numpy import array

# Load and parse the data
data = sc.textFile("data/mllib/gmm_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in line.strip().split(' ')]))

# Build the model (cluster the data)
gmm = GaussianMixture.train(parsedData, 2)

# output parameters of model
for i in range(2):
    print ("weight = ", gmm.weights[i], "mu = ", gmm.gaussians[i].mu,
          "sigma = ", gmm.gaussians[i].sigma.toArray())
```

EM Algorithm for GMM



Introduction

This is a map-reduce version of expectation maximization algo for a mixture of **Gaussians** model. There are two mrJob MR packages, `mr_GMixEmlterate` and `mr_GMixEmInitialize`. The driver calls the mrJob packages and manages the iteration.

E Step: Given mean vector and covariance matrix, calculate the probability of that each data point belongs to a class

P(Cluster_k| Xⁱ; θ)

$$p(\omega_k | \mathbf{x}^{(i)}, \theta) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(i)} | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Estimate class assignments (responsibilities) or weights
Use the current model to estimate the class assignment

M Step: Given probabilities, update mean and covariance

Centroid_k

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) \mathbf{x}^{(i)}$$

Centroid is just a weighted sum of soft assigned examples

Covariance_k

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta) (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)^T$$

Weighted covariance

Prior_k

$$\hat{\pi}_k = \frac{n_k}{n} \text{ where } n_k = \sum_{i=1}^n p(\omega_k | \mathbf{x}^{(i)}, \theta)$$

Class prior is based on the sum of the partial weights

Classification

The example below demonstrates how to load a [LIBSVM data file](#), parse it as an RDD of `LabeledPoint` and then perform classification using a Random Forest. The test error is calculated to measure the algorithm accuracy.

Scala

Java

Python

Classification using Random Forests

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
# Empty categoricalFeaturesInfo indicates all features are continuous.
# Note: Use larger numTrees in practice.
# Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},
                                      numTrees=3, featureSubsetStrategy="auto",
                                      impurity='gini', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = RandomForestModel.load(sc, "myModelPath")
```

Gradient Boosted DTs

Gradient-Boosted Trees (GBTs) are ensembles of [decision trees](#). GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, GBTs handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

MLlib supports GBTs for binary classification and for regression, using both continuous and categorical features. MLlib implements GBTs using the existing [decision tree](#) implementation. Please see the decision tree guide for more information on trees.

Note: GBTs do not yet support multiclass classification. For multiclass problems, please use [decision trees](#) or [Random Forests](#).

Basic algorithm

Gradient boosting iteratively trains a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label. The dataset is re-labeled to put more emphasis on training instances with poor predictions. Thus, in the next iteration, the decision tree will help correct for previous mistakes.

The specific mechanism for re-labeling instances is defined by a loss function (discussed below). With each iteration, GBTs further reduce this loss function on the training data.

Losses

The table below lists the losses currently supported by GBTs in MLlib. Note that each loss is applicable to one of classification or regression, not both.

Notation: N = number of instances. y_i = label of instance i . x_i = features of instance i . $F(x_i)$ = model's predicted label for instance i .

Loss	Task	Formula	Description
Log Loss	Classification	$2 \sum_{i=1}^N \log(1 + \exp(-2y_i F(x_i)))$	Twice binomial negative log likelihood.
Squared Error	Regression	$\sum_{i=1}^N (y_i - F(x_i))^2$	Also called L2 loss. Default loss for regression tasks.
Absolute Error	Regression	$\sum_{i=1}^N y_i - F(x_i) $	Also called L1 loss. Can be more robust to outliers than Squared Error.

The example below demonstrates how to load a [LIBSVM data file](#), parse it as an RDD of LabeledPoint and then perform classification using Gradient-Boosted Trees with log loss. The test error is calculated to measure the algorithm accuracy.

Scala Java Python

```
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file.
data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a GradientBoostedTrees model.
# Notes: (a) Empty categoricalFeaturesInfo indicates all features are continuous.
#        (b) Use more iterations in practice.
model = GradientBoostedTrees.trainClassifier(trainingData,
                                              categoricalFeaturesInfo={}, numIterations=3)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification GBT model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "myModelPath")
sameModel = GradientBoostedTreesModel.load(sc, "myModelPath")
```

Part 3: Machine Learning in Spark

- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

- **Pipelines**

Pipelines

- In machine learning, it is common to run a sequence of algorithms to process and learn from data.
- E.g., a simple text document processing workflow might include several stages:
 - Split each document's text into words.
 - Convert each document's words into a numerical feature vector.
 - Normalization
 - Learn a prediction model using the feature vectors and labels.
 - Spark ML represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this simple workflow as a running example in this section.

Pipeline-Training data

In Python, The single asterisk form (*args) is used to pass a non-keyworded, variable-length argument list in function definition. It can also unpacks the sequence/collection into positional arguments

Example of Unpacking

```
def sum(a, b):
    return a + b

values = (1, 2)

s = sum(*values)
```

```
# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0, "a b c d e spark", 1.0),
                           (1, "b d", 0.0),
                           (2, "spark f g h", 1.0),
                           (3, "hadoop mapreduce", 0.0),
                           (4, "b spark who", 1.0),
                           (5, "g d a y", 0.0),
                           (6, "spark fly", 1.0),
                           (7, "was mapreduce", 0.0),
                           (8, "e spark program", 1.0),
                           (9, "a e c l", 0.0),
                           (10, "spark compile", 1.0),
                           (11, "hadoop software", 0.0)
                          ]) \
    .map(lambda x: LabeledDocument(*x)).toDF()
training.collect()
```

```
[Row(id=0, text=u'a b c d e spark', label=1.0),
Row(id=1, text=u'b d', label=0.0),
Row(id=2, text=u'spark f g h', label=1.0),
Row(id=3, text=u'hadoop mapreduce', label=0.0),
Row(id=4, text=u'b spark who', label=1.0),
Row(id=5, text=u'g d a y', label=0.0),
Row(id=6, text=u'spark fly', label=1.0),
Row(id=7, text=u'was mapreduce', label=0.0),
Row(id=8, text=u'e spark program', label=1.0),
Row(id=9, text=u'a e c l', label=0.0),
Row(id=10, text=u'spark compile', label=1.0),
Row(id=11, text=u'hadoop software', label=0.0)]
```

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

There are several variants on the definition of term frequency and document frequency. In MLlib, we separate TF and IDF to make them flexible.

Our implementation of term frequency utilizes the [hashing trick](#). A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1,048,576$.

Note: MLlib doesn't provide tools for text segmentation. We refer users to the [Stanford NLP Group](#) and [scalanlp/chalk](#).



Scala

Python

TF and IDF are implemented in [HashingTF](#) and [IDF](#). HashingTF takes an RDD of list as the input. Each record could be an iterable of strings or other types.

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

sc = SparkContext()

# Load documents (one per line).
documents = sc.textFile("...").map(lambda line: line.split(" "))

hashingTF = HashingTF()
tf = hashingTF.transform(documents)
```

While applying HashingTF only needs a single pass to the data, applying scale the term frequencies by IDF.

```
from pyspark.mllib.feature import IDF

# ... continue from the previous example
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

MLLib's IDF implementation provides an option for ignoring terms which IDF for these terms is set to 0. This feature can be used by passing the `minDocFreq` parameter.

```
# ... continue from the previous example
tf.cache()
idf = IDF(minDocFreq=2).fit(tf)
tfidf = idf.transform(tf)
```

A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1,048,576$.

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D).$$

There are several variants on the definition of term frequency and document frequency. In MLlib, we separate TF and IDF to make them flexible.

Our implementation of term frequency utilizes the [hashing trick](#). A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20} = 1,048,576$.

Note: MLlib doesn't provide tools for text segmentation. We refer users to the [Stanford NLP Group](#) and [scalanlp/chalk](#).



Scala

Python

TF and IDF are implemented in [HashingTF](#) and [IDF](#). HashingTF takes an RDD of list as the input. Each record could be an iterable of strings or other types.

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

sc = SparkContext()

# Load documents (one per line).
documents = sc.textFile("...").map(lambda line: line.split(" "))

hashingTF = HashingTF()
tf = hashingTF.transform(documents)
```

While applying HashingTF only needs a single pass to the data, applying IDF needs two passes: first to compute the IDF vector and second to scale the term frequencies by IDF.

```
from pyspark.mllib.feature import IDF

# ... continue from the previous example
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

MLLib's IDF implementation provides an option for ignoring terms which occur in less than a minimum number of documents. In such cases, the IDF for these terms is set to 0. This feature can be used by passing the minDocFreq value to the IDF constructor.

```
# ... continue from the previous example
tf.cache()
idf = IDF(minDocFreq=2).fit(tf)
tfidf = idf.transform(tf)
```

A raw feature is mapped into an index (term) by applying a hash function. Then term frequencies are calculated based on the mapped indices. This approach avoids the need to compute a global term-to-index map, which can be expensive for a large corpus, but it suffers from potential hash collisions, where different raw features may become the same term after hashing. To reduce the chance of collision, we can increase the target feature dimension, i.e., the number of buckets of the hash table. The default feature dimension is $2^{20}=1,048,576$.

Training Pipeline

Pipeline
(Estimator)



Pipeline.fit()



Raw
text

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext
```

```
sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)
```

```
# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
    (1L, "b d", 0.0),
    (2L, "spark f g h", 1.0),
    (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()
```

```
# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)
```



Words



Feature



Logistic
Regression
Model

Training Pipeline

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row, SQLContext

sc = SparkContext(appName="SimpleTextClassificationPipeline")
sqlContext = SQLContext(sc)

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

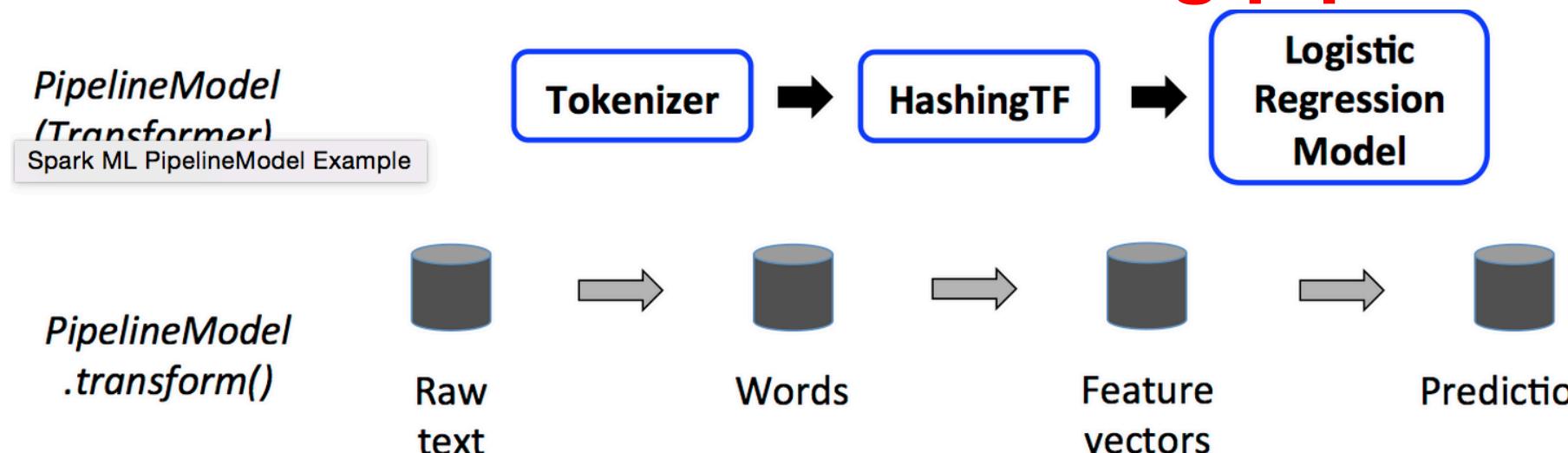
# Fit the pipeline to training documents.
model = pipeline.fit(training)
```

.ogistic
gression



Logistic
Regression
Model

Testing pipeline



Above, the PipelineModel has the same number of stages as the original Pipeline, but all operations in the original Pipeline

```
# Prepare test documents, which are unlabeled.  
Document = Row("id", "text")  
test = sc.parallelize([(4L, "spark i j k"),  
                      (5L, "l m n"),  
                      (6L, "mapreduce spark"),  
                      (7L, "apache hadoop"))]\br/>.map(lambda x: Document(*x)).toDF()  
  
# Make predictions on test documents and print columns of interest.  
prediction = model.transform(test)  
selected = prediction.select("id", "text", "prediction")  
for row in selected.collect():  
    print row
```

ML Pipelines: use pipeline-based CV

Hyper-parameter Tuning

```
// Build a parameter grid.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))  
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))  
  .build()  
  
// Set up cross-validation.  
val cv = new CrossValidator()  
  .setNumFolds(3)  
  .setEstimator(pipeline)  
  .setEstimatorParamMaps(paramGrid)  
  .setEvaluator(new BinaryClassificationEvaluator)  
  
// Fit a model with cross-validation.  
val cvModel = cv.fit(trainingDataset)
```

Pipeline - Stages & Cross Validation



```
# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
from pyspark import SparkContext
from pyspark.ml import Pipeline
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Prepare training documents, which are labeled.
LabeledDocument = Row("id", "text", "label")
training = sc.parallelize([(0L, "a b c d e spark", 1.0),
                           (1L, "b d", 0.0),
                           (2L, "spark f g h", 1.0),
                           (3L, "hadoop mapreduce", 0.0)]) \
    .map(lambda x: LabeledDocument(*x)).toDF()

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
```

Pipeline - Stages&Cross Validation

3 hashingTF.number of features: 10, 100, 1000

2 logistic regression regularization parameters: 0.1, 0.001

```
# Use a ParamGridBuilder to construct a grid of parameters to search over.  
paramGrid = ParamGridBuilder() \  
    .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \  
    .addGrid(lr.regParam, [0.1, 0.01]) \  
    .build()
```

regParam is for L2 regularization

```
# Run cross-validation, and choose the best set of parameters  
crossval = CrossValidator(estimator=pipeline,  
                          estimatorParamMaps=paramGrid,  
                          evaluator=BinaryClassificationEvaluator(),  
                          numFolds=2) # use 3+ folds in practice  
  
cvModel = crossval.fit(training)
```

For each iteration in cross validation, follow this pipeline

By default, BinaryclassificationEvaluator is areaUnderROC

Pipeline-Prediction

```
# Prepare test documents, which are unlabeled.  
Document = Row("id", "text")  
test = sc.parallelize([(4, "spark i j k"),  
                      (5, "l m n"),  
                      (6, "spark hadoop spark"),  
                      (7, "apache hadoop"))]) \  
    .map(lambda x: Document(*x)).toDF()  
  
# Make predictions on test documents and print columns of interest.  
prediction = cvModel.transform(test)  
selected = prediction.select("id", "text", "prediction")  
for row in selected.collect():  
    print(row)  
  
sc.stop()
```

```
Row(id=4, text=u'spark i j k', prediction=1.0)  
Row(id=5, text=u'l m n', prediction=0.0)  
Row(id=6, text=u'spark hadoop spark', prediction=1.0)  
Row(id=7, text=u'apache hadoop', prediction=0.0)
```

Part 4: Low-level Machine Learning in Spark

- **Motivation**
- **Optimization Theory**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

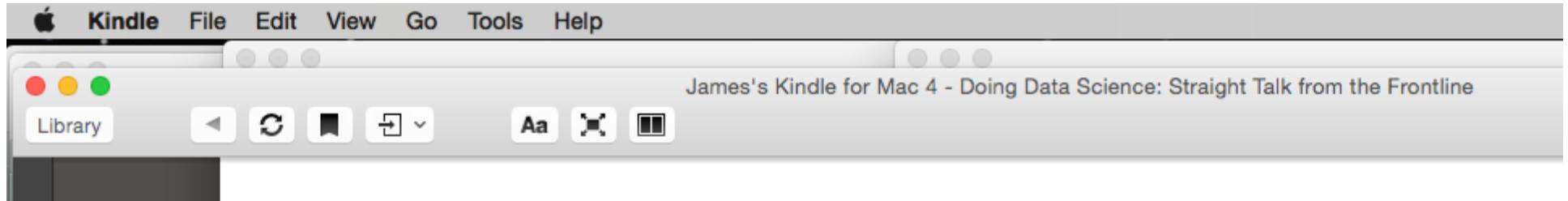
Thought Experiment: Learning by Example

Let's start by looking at a bunch of text shown in [Figure 4-1](#), whose rows seem to contain the subject and first line of an email in an inbox.

You may notice that several of the rows of text look like spam.

How did you figure this out? Can you write code to automate the spam filter that your brain represents?

<input type="checkbox"/>	<input type="checkbox"/>	Pure Saffron Extract	Melt Fat Away - Drop 11-lbs in 7 Days!
<input type="checkbox"/>	<input type="checkbox"/>	Blue Sky Auto	Car Loans Available - Bad Credit Accepted
<input type="checkbox"/>	<input type="checkbox"/>	Watch The Video	Shocking Discovery Gets You Laid - Scientists at Harvard University have discovered a strange secret that all
<input type="checkbox"/>	<input type="checkbox"/>	Casino	Casino Promotions - With the Slots of Vegas Instant-Win Scratch Ticket Game you can get \$100 on the ho
<input type="checkbox"/>	<input type="checkbox"/>	Designer Watch Replica	Replica Watches On Sale - Replica Watches: Swiss Luxury Watch Replicas, Rolex, Omega, Breitling Chec
<input type="checkbox"/>	<input type="checkbox"/>	A.C., me (10)	I'm late to this party - I'm free and interested. Tell me more! I'd have to think about the students, but I know s
<input type="checkbox"/>	<input type="checkbox"/>	Rachel .. Christoforos (18)	Fwd: Invitation to speak at upcoming Big Data Workshop, hosted by Imperial College London - Dear Rachel,
<input type="checkbox"/>	<input type="checkbox"/>	Fat Burning Hormone	17 Foods that GET RID of stomach fat
<input type="checkbox"/>	<input type="checkbox"/>	Kaplan University	Kaplan University online and campus degree programs
<input type="checkbox"/>	<input type="checkbox"/>	Dinn Trophy	Sport Plaques - As Low As \$4.29 - View this message in a browser. Shop Sport Plaques Shop Now> Chang
<input type="checkbox"/>	<input type="checkbox"/>	me, Philipp (2)	checking in - Hi Rachel, I know I had started writing a few emails to you, but then I (obviously) didn't sent
<input type="checkbox"/>	<input type="checkbox"/>	me, Matthew (3)	doing data science - Hi Matt, Not a duplicate (just FYI if that helps debug) Well, so the status is that we're in
<input type="checkbox"/>	<input type="checkbox"/>	Luxury Replicas	Rolex, Breitling, Chanel, Omega, LV, and muchMore! - Super Replicas - Luxury Watches, Bags, Jewelry, Ph
<input type="checkbox"/>	<input type="checkbox"/>	Watch this video and wom.	Watch this video and women will adore you - Can you get laid using just the words in this video? Click Here '
<input type="checkbox"/>	<input type="checkbox"/>	Adriana	I ADDED YOU to my Private Wish List - Sorry, I've been out of town but I am back and I'm looking for a good



This similarly answers the question: given the data I saw, which parameter θ is the most likely?

Here we will apply the spirit of Bayes's Law to transform θ_{MAP} to get something that is, up to a constant, equivalent to $p(D \mid \theta) \cdot p(\theta)$. The term $p(\theta)$ is referred to as the "prior," and we have to make an assumption about its form to make this useful. If we make the assumption that the probability distribution of θ is of the form $\theta^{\alpha}(1-\theta)^{\beta}$, for some α and β , then we recover the Laplace Smoothed result.

IS THAT A REASONABLE ASSUMPTION?

Recall that θ is the chance that a word is in spam if that word is in some email. On the one hand, as long as both $\alpha > 0$ and $\beta > 0$, this distribution vanishes at both 0 and 1. This is reasonable: you want very few words to be expected to never appear in spam or to always appear in spam.

On the other hand, when α and β are large, the shape of the distribution is bunched in the middle, which reflects the prior that most words are equally likely to appear in spam or outside spam. That doesn't seem true either.

A compromise would have α and β be positive but small, like 1/5. That would keep your spam filter from being too overzealous without having the wrong idea. Of course, you could relax this prior as you have more and better data; in general, strong priors are only needed when you don't have sufficient data.

Comparing Naive Bayes to k-NN

Sometimes α and β are called "pseudocounts." Another common name is "hyperparameters." They're fancy but also simple. It's up to you, the data scientist, to set the values of these two hyperparameters in the numerator and denominator for smoothing, and it gives you two knobs to tune. By contrast, k-NN has one knob, namely k , the number of neighbors. Naive Bayes is a linear classifier, while k-NN is not. The curse of dimensionality and large feature sets are a problem for k-NN, while Naive Bayes performs well. k-NN requires no training (just load in the dataset), whereas Naive Bayes does. Both are examples of supervised learning (the data comes labeled).

Sample Code in bash

```
#!/bin/bash
#
# file: enron_naive_bayes.sh
#
# description: trains a simple one-word naive bayes spam
# filter using enron email data
#
# usage: ./enron_naive_bayes.sh <word>
#
# requirements:
```

```
fi

# if the file doesn't exist, download from the web
if ! [ -e enron1.tar.gz ]
then
wget 'http://www.aueb.gr/users/ion/data/enron-spam/preprocessed/enron1.tar.gz'
fi

# if the directory doesn't exist, uncompress the .tar.gz
if ! [ -d enron1 ]
then
tar zxvf enron1.tar.gz
fi

# change into enron1
cd enron1

# get counts of total spam, ham, and overall msgs
Nspam=`ls -l spam/*.txt | wc -l`
Nham=`ls -l ham/*.txt | wc -l`
Ntot=$Nspam+$Nham

echo $Nspam spam examples
echo $Nham ham examples

# get counts containing word in spam and ham classes
Nword_spam=`grep -il $word spam/*.txt | wc -l`
Nword_ham=`grep -il $word ham/*.txt | wc -l`

echo $Nword_spam "spam examples containing $word"
echo $Nword_ham "ham examples containing $word"

# calculate probabilities using bash calculator "bc"
Pspam=`echo "scale=4; $Nspam / ($Nspam+$Nham)" | bc`
Pham=`echo "scale=4; 1-$Pspam" | bc`
echo
echo "estimated P(spam) =" $Pspam
echo "estimated P(ham) =" $Pham

Pword_spam=`echo "scale=4; $Nword_spam / $Nspam" | bc`
Pword_ham=`echo "scale=4; $Nword_ham / $Nham" | bc`
echo
echo "estimated P($word|spam) =" $Pword_spam
echo "estimated P($word|ham) =" $Pword_ham

Pspam_word=`echo "scale=4; $Pword_spam*$Pspam" | bc`
Pham_word=`echo "scale=4; $Pword_ham*$Pham" | bc`
Pword=`echo "scale=4; $Pspam_word+$Pham_word" | bc`
Pspam_word=`echo "scale=4; $Pspam_word / $Pword" | bc`
echo
echo "P(spam|$word) =" $Pspam_word

# return original directory
cd ..
```

Naïve Bayes Classifier

- In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the input features.
- Naive Bayes for Discrete-Valued Inputs
 - We can estimate these parameters using either maximum likelihood estimates (based on calculating the relative frequencies of the different events in the data), or using Bayesian MAP estimates (augmenting this observed data with prior distributions over the values of these parameters).
 - Laplace smoothing
 - Dirichlet prior distribution over the θ_{ijk} parameters, with equal-valued parameters. Laplace smoothing.
- Naive Bayes for real-Valued Inputs
 - One common approach is to assume that for each possible discrete value y_k of Y , the distribution of each continuous X_i is Gaussian, and is defined by a mean and standard deviation specific to X_i and y_k .
 - either use maximum likelihood estimates (MLE) or maximum a posteriori (MAP) estimates for these parameters.

<https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>

Naïve Bayes Classifier

- In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the input features.
- Naive Bayes for Discrete-Valued Inputs
 - We can estimate these parameters using either maximum likelihood estimates (based on calculating the relative frequencies of the different events in the data), or using Bayesian MAP estimates (augmenting this observed data with prior distributions over the values of these parameters).
 - Laplace smoothing
 - Dirichlet prior distribution over the θ_{ijk} parameters, with equal-valued parameters. Laplace smoothing.
- Naive Bayes for real-Valued Inputs

The maximum likelihood estimator for σ_{ik}^2 is

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

This maximum likelihood estimator is biased, so the minimum variance unbiased estimator (MVUE) is sometimes used instead. It is

$$\hat{\sigma}_{ik}^2 = \frac{1}{(\sum_j \delta(Y^j = y_k)) - 1} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k) \quad (15)$$

possible discrete value y_k of Y^j , and is defined by a mean

<https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>

Gaussian naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a [Gaussian](#) distribution. For example, suppose the training data contain a continuous attribute, x . We first segment the data by the class, and then compute the mean and [variance](#) of x in each class. Let μ_c be the mean of the values in x associated with class c , and let σ_c^2 be the variance of the values in x associated with class c . Then, the probability *distribution* of some value given a class, $p(x = v|c)$, can be computed by plugging v into the equation for a [Normal distribution](#) parameterized by μ_c and σ_c^2 . That is,

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

Another common technique for handling continuous values is to use binning to [discretize](#) the feature values, to obtain a new set of Bernoulli-distributed features; some literature in fact suggests that this is necessary to apply naive Bayes, but it is not, and the discretization may [throw away discriminative information](#).^[4]

Naïve Bayes

- A generative, parametric classification model
- Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using Bayes rule.
- Make a conditional independence assumption; this leads to a Naïve Bayes classifier
 - Reduces the number of parameters from $2^{n-1} * 2$ parameters to $2n$
- ***Definition: Given random variables X; Y and Z, we say X is conditionally independent of Y given Z, if and only if the probability distribution governing X is independent of the value of Y given Z;***
 - $(\forall i; j; k) P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$

Training NB: One pass; Linear time

- Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.
- Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Semi-supervised parameter estimation

Given a way to train a naive Bayes classifier from labeled data, it's possible to construct a semi-supervised training algorithm that can learn from a combination of labeled and unlabeled data by running the supervised learning algorithm in a loop:^[11]

Given a collection $D = L \uplus U$ of labeled samples L and unlabeled samples U , start by training a naive Bayes classifier on L .

Until convergence, do:

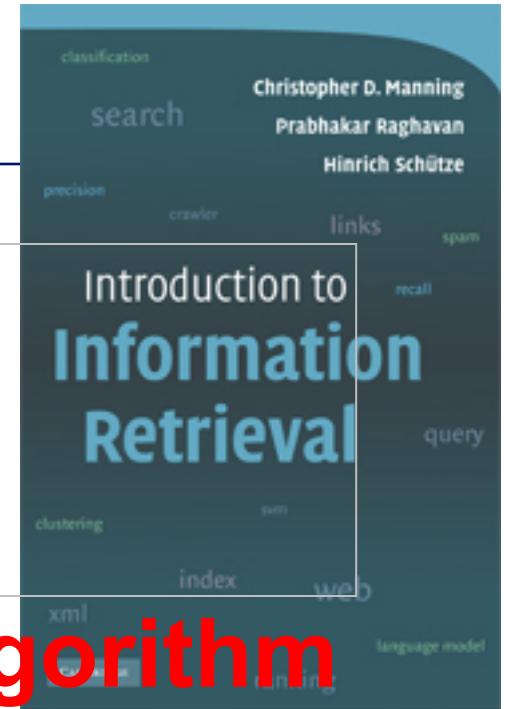
Predict class probabilities $P(C|x)$ for all examples x in D .

Re-train the model based on the *probabilities* (not the labels) predicted in the previous step.

Convergence is determined based on improvement to the model likelihood $P(D|\theta)$, where θ denotes the parameters of the naive Bayes model.

This training algorithm is an instance of the more general expectation–maximization algorithm (EM): the prediction step inside the loop is the *E*-step of EM, while the re-training of naive Bayes is the *M*-step. The algorithm is formally justified by the assumption that the data are generated by a mixture model, and the components of this mixture model are exactly the classes of the classification problem.^[11]

[Nigam, Kamal; McCallum, Andrew; Thrun, Sebastian; Mitchell, Tom (2000). "Learning to classify text from labeled and unlabeled documents using EM" (PDF). Machine Learning.]



The Naïve Bayes algorithm

***Some slides Adapted from Lectures
by
Prabhakar Raghavan (Yahoo and Stanford)
and Christopher Manning (Stanford)***

Reading material

- **PDF and HTML versions of Chapter 13 are available here**
 - [Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.] <http://nlp.stanford.edu/IR-book/>
 - <http://www.cs.cmu.edu/~epxing/Class/10701-10s/Lecture/lecture5.pdf>
 - http://www.cs.cmu.edu/~tom/10701_sp11/lectures.shtml

Spam filtering: Another text classification task

From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

=====

Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>

=====

Classification Tasks

- **Text**
 - Sentiment detection
 - Email sorting
 - Vertical search
 - NLP tasks (e.g., POS, Chunking)
- **Image understanding**
 - Scene classification

Categorization/Classification

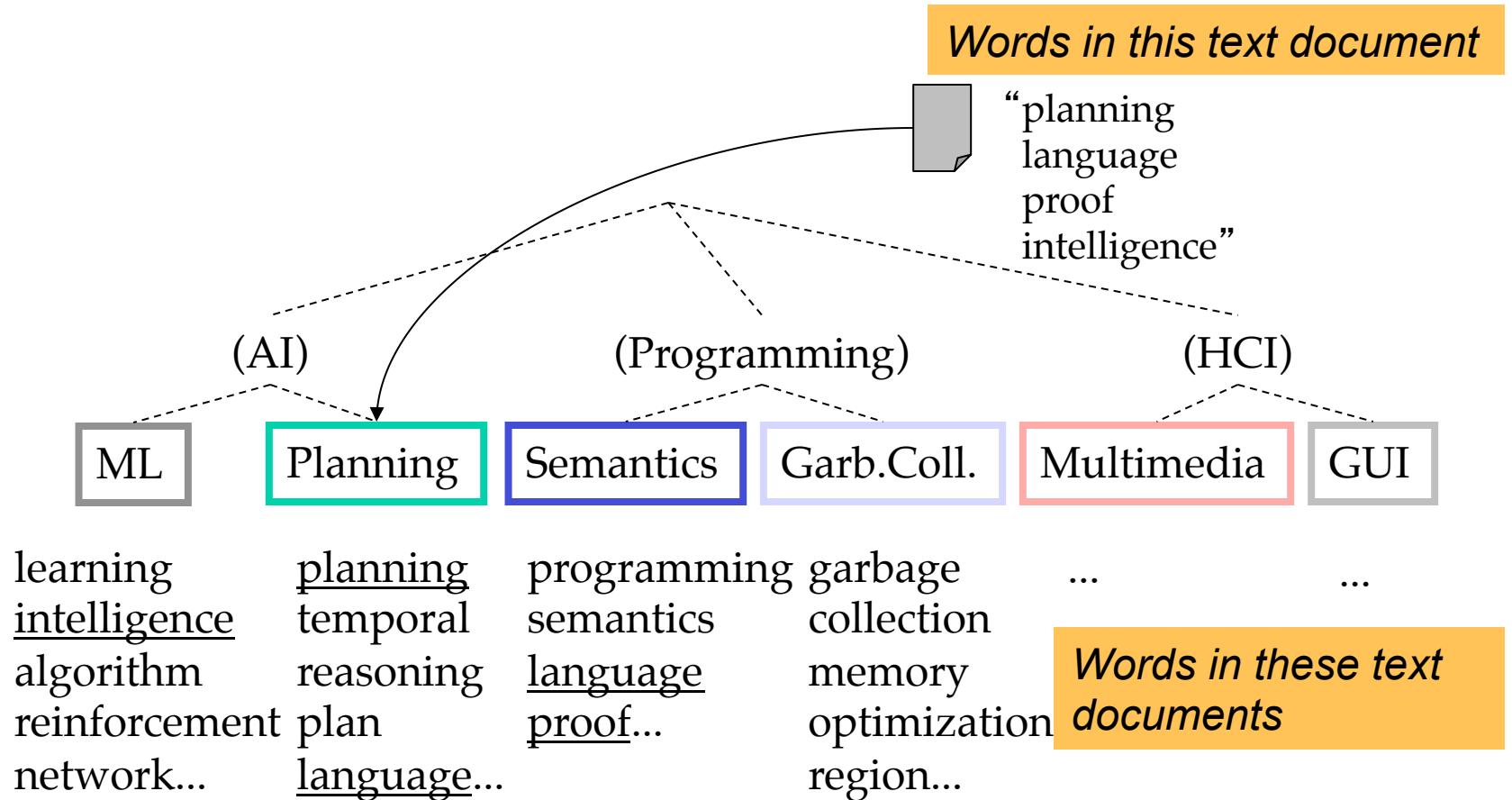
- **Given:**
 - A description of an instance, $x \in X$, where X is the *instance language* or *instance space*.
 - *Issue:* how to represent text documents.
 - A fixed set of classes:
 $C = \{c_1, c_2, \dots, c_J\}$
- **Determine:**
 - The category of x : $c(x) \in C$, where $c(x)$ is a *classification function* whose domain is X and whose range is C .
 - We want to know how to build classification functions (“classifiers”).

Document Classification

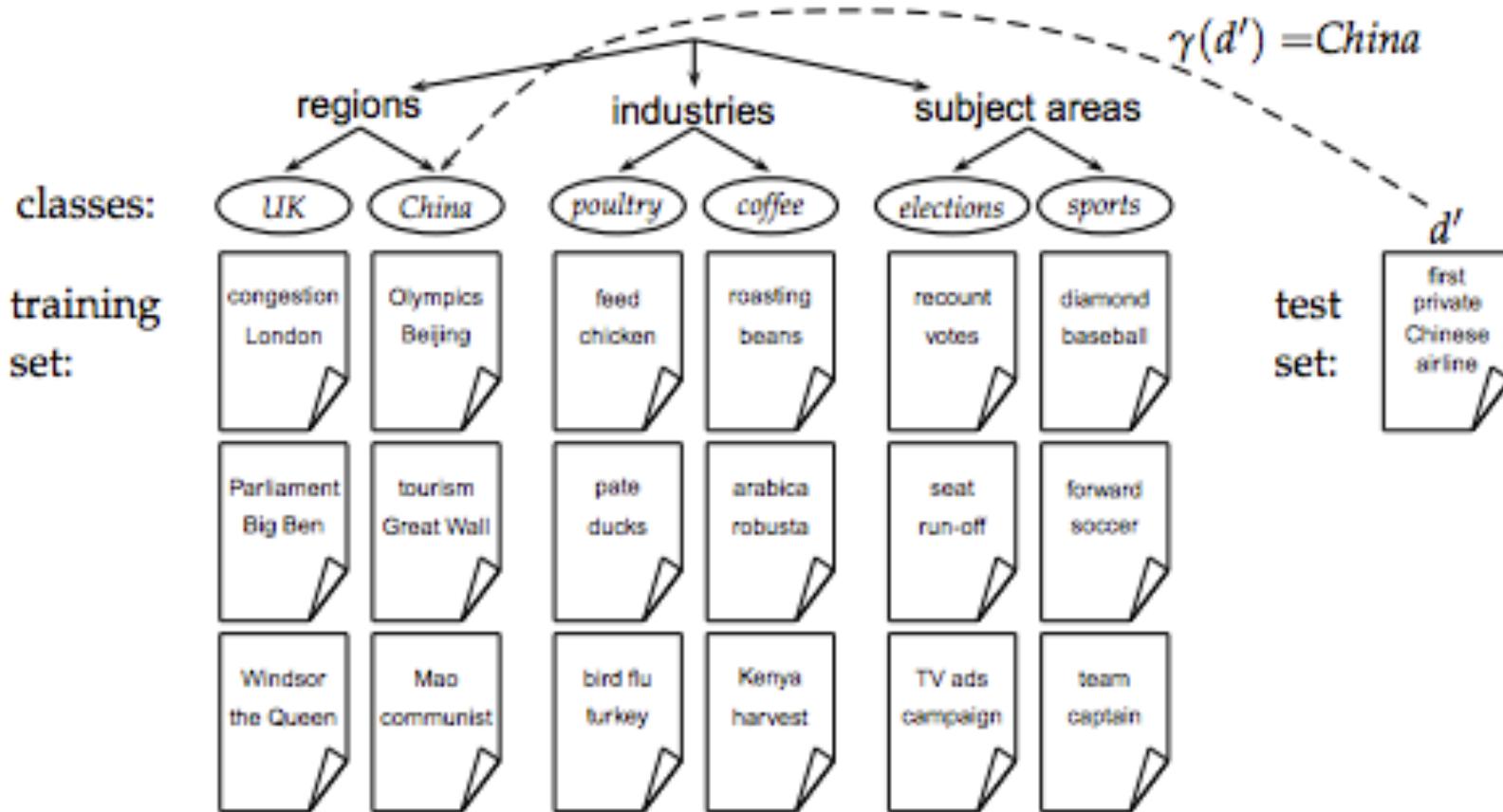
Test Data:

Classes:

Training Data:



(Note: in real life there is often a hierarchy, not present in the above problem statement; and also, you get papers on ML approaches to Garb. Coll.)



► Figure 13.1 Classes, training set, and test set in text classification .

text classification from the Reuters-RCV1 Collection: Test Doc: assign to China class

More Text Classification Examples:

Many search engine functionalities use classification

Soft Assignment

Assign labels to each document or web-page:

- Labels are most often topics such as Yahoo-categories
e.g., "finance," "sports," "news>world>asia>business"
- Labels may be genres
e.g., "editorials" "movie-reviews" "news"
- Labels may be opinion on a person/product
e.g., "like", "hate", "neutral"
- Labels may be domain-specific
 - e.g., "interesting-to-me" : "not-interesting-to-me"
 - e.g., "contains adult language" : "doesn't"
 - e.g., *language identification: English, French, Chinese, ...*
- e.g., *search vertical: about Linux versus not*
e.g., "link spam" : "not link spam"

Classification Methods (1)

- **Manual classification**
 - Used by Yahoo! (originally; now downplayed), Looksmart, about.com, ODP, PubMed
 - Very *accurate* when job is done by experts
 - *Consistent* when the problem size and team is small
 - *Difficult and expensive to scale*
 - Means we need automatic classification methods for big problems

Classification Methods (2)

- **Automatic document classification**
 - Hand-coded rule-based systems
 - Used by CS dept's spam filter, Reuters, CIA, etc.
 - Companies (Verity) provide “IDE” for writing such rules
 - E.g., assign category if document contains a given boolean combination of words
 - Standing queries: Commercial systems have complex query languages (everything in IR query languages + accumulators)
 - Accuracy is often very high if a rule has been carefully refined over time by a subject expert
 - **Building and maintaining these rules is expensive**

Classification Methods (3)

- **Supervised learning of a document-label assignment function**
 - Many systems partly rely on machine learning (Autonomy, MSN, Verity, Enkata, Yahoo!, ...)
 - k-Nearest Neighbors (simple, powerful)
 - Naive Bayes (simple, common method)
 - Support-vector machines (new, more powerful)
 - ... plus many other methods
 - No free lunch: requires hand-classified training data
 - But data can be built up (and refined) by amateurs
- **Note that many commercial systems use a mixture of methods**

Recall a few probability basics

- For events a and b :
- Bayes' Rule

$$\Pr(\text{Spade}) = \frac{1}{4}$$

$$\Pr(\text{King}) = \frac{1}{13}$$

$$\Pr(\text{Spade}) \text{ and } \Pr(\text{King}) = \frac{1}{52}$$

$$\Pr(\text{king|spade}) = \frac{1}{13}$$

$$\Pr(\text{king|spade}) = \Pr(\text{Spade}) \text{ and } \Pr(\text{King}) / \Pr(\text{Spade})$$

$$\Pr(\text{spade|King}) = \Pr(\text{Spade}) \text{ and } \Pr(\text{King}) / \Pr(\text{King})$$

$$p(a, b) = p(a \cap b) = p(a | b)p(b) = p(b | a)p(a)$$

$$p(\bar{a} | b)p(b) = p(b | \bar{a})p(\bar{a})$$

$$p(a | b) = \frac{p(b | a)p(a)}{p(b)} = \frac{p(b | a)p(a)}{\sum_{x=a, \bar{a}} p(b | x)p(x)}$$

Posterior

Prior

- Odds:

$$O(a) = \frac{p(a)}{p(\bar{a})} = \frac{p(a)}{1 - p(a)}$$

Prasad

438

Bayesian Methods

- Learning and classification methods based on probability theory.
 - Bayes theorem plays a critical role in probabilistic learning and classification.
- Build a *generative model* that approximates how data is produced.
- Uses *prior* probability of each category given no information about an item.
- Categorization produces a *posterior* probability distribution over the possible categories given a description of an item (and prior probabilities).

Bayes' Rule

$$\Pr(\text{Spade}) = \frac{1}{4}$$

$$\Pr(\text{King}) = \frac{1}{13}$$

$$\Pr(\text{Spade}) \text{ and } \Pr(\text{King}) = \frac{1}{52}$$

$$\Pr(\text{king|spade}) = \frac{1}{13}$$

$$\Pr(\text{king|spade}) = \Pr(\text{Spade}) \text{ and } \Pr(\text{King}) / \Pr(\text{Spade})$$

$$\Pr(\text{spade|King}) = \Pr(\text{Spade}) \text{ and } \Pr(\text{King}) / \Pr(\text{King})$$

Theorem of total probabilities

$$P(D) = P(D|C=\text{TRUE}) + P(D|C=\text{FALSE})$$

$$P(C, D) = P(C | D)P(D) = P(D | C)P(C)$$

$$P(C | D) = \frac{P(D | C)P(C)}{P(D)}$$

TASK $P(\text{class=china} | \{\text{T}, \text{C}\})$ vs $P(\text{class=not china} | \{\text{T}, \text{C}\})$

Naive Bayes Classifiers

Task: Classify a new instance D based on a tuple of attribute values into one of the classes $c_j \in C$

$$D = \langle x_1, x_2, \dots, x_n \rangle$$

$$c_{MAP} = \operatorname{argmax}_{c_j \in C} P(c_j | x_1, x_2, \dots, x_n)$$

$$= \operatorname{argmax}_{c_j \in C} \frac{P(x_1, x_2, \dots, x_n | c_j) P(c_j)}{P(x_1, x_2, \dots, x_n)}$$

$$= \operatorname{argmax}_{c_j \in C} P(x_1, x_2, \dots, x_n | c_j) P(c_j)$$

MAP = Maximum Aposteriori Probability

Naïve Bayes Classifier: Naïve Bayes Assumption

- $P(c_j)$
 - Can be estimated from the frequency of classes in the training examples.
- $P(x_1, x_2, \dots, x_n | c_j)$
 - $O(|X|^n \cdot |C|)$ parameters
 - Could only be estimated if a very, very large number of training examples was available.

Naïve Bayes Conditional Independence Assumption:

- **Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(x_i | c_j)$.**

Naïve Bayes

- Two different ways to set up a Naïve Bayes classifier
- Different type of input variables (X_i):
 - Multivariate Bernoulli or Bernoulli Naïve Bayes Model

	docID	words in document	in $c = China?$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$\hat{P}(\text{Chinese}|c) = (3+1)/(3+2) = 4/5$$

Count 3 documents with Chinese for YES class

- Multinomial Naïve Bayes

Parameter estimation for Naïve Bayes

- **Multivariate Bernoulli model:** $X_w = t$ if word present document

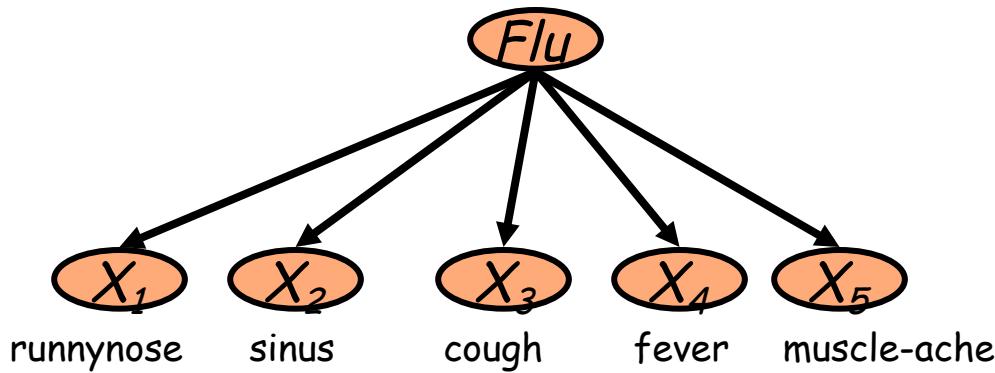
$$\hat{P}(X_w = t \mid c_j) = \text{fraction of documents of topic } c_j \text{ in which word } w \text{ appears}$$

- **Multinomial model:**

$$\hat{P}(X_i = w \mid c_j) = \text{fraction of times in which word } w \text{ appears across all documents of topic } c_j$$

- Can create a mega-document for topic j by concatenating all documents on this topic
- Use frequency of w in mega-document

The Naïve Bayes Classifier

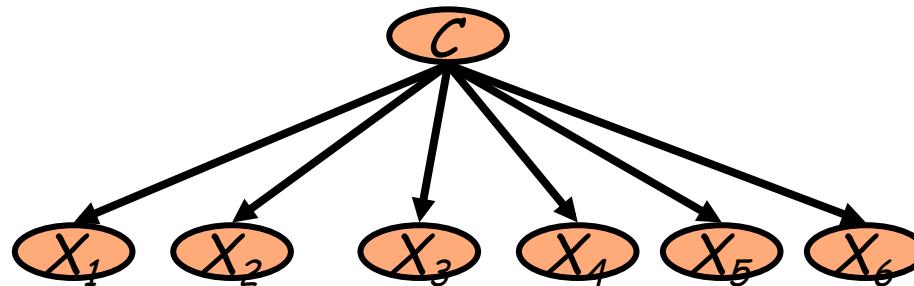


- **Conditional Independence Assumption:** Features (term presence) are *independent* of each other given the class:

$$P(X_1, \dots, X_5 | C) = P(X_1 | C) \cdot P(X_2 | C) \cdot \dots \cdot P(X_5 | C)$$

- This model is appropriate for binary variables
 - Multivariate Bernoulli model

Learning the Model

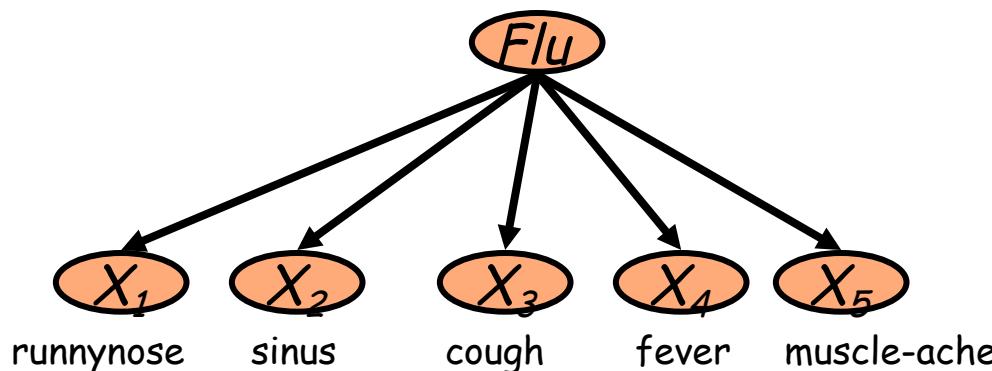


- ***First attempt: maximum likelihood estimates***
 - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{N(C = c_j)}{N}$$

$$\hat{P}(x_i | c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

Problem with Max Likelihood



$$P(X_1, \dots, X_5 | C) = P(X_1 | C) \cdot P(X_2 | C) \cdot \dots \cdot P(X_5 | C)$$

- What if we have seen no training cases where patient had no flu and muscle aches?

$$\hat{P}(X_5 = t | C = nf) = \frac{N(X_5 = t, C = nf)}{N(C = nf)} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$\ell = \arg \max_c \hat{P}(c) \prod_i \hat{P}(x_i | c)$$

Bernoulli model: Smoothing to Avoid Overfitting and Zero probabilities

$$\hat{P}(x_i | c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}$$

$\# \text{ of values of } X_i$

Laplace smoothing for Bernoulli model

1 for every class, as there are two cases to consider for each word

Eqn 13.7 and also page 264 in <http://nlp.stanford.edu/IR-book/pdf/13bayes.pdf>

Classification

- **Multinomial vs Multivariate Bernoulli?**
- **Multinomial model is almost always more effective in text applications!**
- **See *IR Book* sections 13.2 and 13.3 for worked examples with each model**

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Vocabulary = {Chinese, Beijing, Shanghai, Macao, Tokyo}



Example 13.2: Applying the Bernoulli model to the example in Table 13.1, we have the same estimates for the priors as before: $\hat{P}(c) = 3/4$, $\hat{P}(\bar{c}) = 1/4$. The conditional probabilities are:

Japan does NOT occur class C

$$\hat{P}(\text{Chinese}|c) = (3+1)/(3+2) = 4/5$$

$$\hat{P}(\text{Japan}|c) = \hat{P}(\text{Tokyo}|c) = (0+1)/(3+2) = 1/5$$

$$\hat{P}(\text{Beijing}|c) = \hat{P}(\text{Macao}|c) = \hat{P}(\text{Shanghai}|c) = (1+1)/(3+2) = 2/5$$

$$\hat{P}(\text{Chinese}|\bar{c}) = (1+1)/(1+2) = 2/3$$

$$\hat{P}(\text{Japan}|\bar{c}) = \hat{P}(\text{Tokyo}|\bar{c}) = (1+1)/(1+2) = 2/3$$

$$\hat{P}(\text{Beijing}|\bar{c}) = \hat{P}(\text{Macao}|\bar{c}) = \hat{P}(\text{Shanghai}|\bar{c}) = (0+1)/(1+2) = 1/3$$

The denominators are $(3+2)$ and $(1+2)$ because there are three documents in c and one document in \bar{c} and because the constant B in Equation (13.7) is 2 – there are two cases to consider for each term, occurrence and nonoccurrence.

The scores of the test document for the two classes are

$$\begin{aligned} P(c|d_5) &\propto \hat{P}(c) \cdot \hat{P}(\text{Chinese}|c) \cdot \hat{P}(\text{Japan}|c) \cdot \hat{P}(\text{Tokyo}|c) \\ &\quad \cdot (1 - \hat{P}(\text{Beijing}|c)) \cdot (1 - \hat{P}(\text{Shanghai}|c)) \cdot (1 - \hat{P}(\text{Macao}|c)) \\ &= 3/4 \cdot 4/5 \cdot 1/5 \cdot 1/5 \cdot (1 - 2/5) \cdot (1 - 2/5) \cdot (1 - 2/5) \\ &\approx 0.005 \end{aligned}$$

and, analogously,

$$\begin{aligned} \hat{P}(\bar{c}|d_5) &\propto 1/4 \cdot 2/3 \cdot 2/3 \cdot 2/3 \cdot (1 - 1/3) \cdot (1 - 1/3) \cdot (1 - 1/3) \\ &\approx 0.022 \end{aligned}$$

Thus, the classifier assigns the test document to $\bar{c} = \text{not-China}$. When looking only at binary occurrence and not at term frequency, Japan and Tokyo are indicators for \bar{c} ($2/3 > 1/5$) and the conditional probabilities of Chinese for c and \bar{c} are not different enough ($4/5$ vs. $2/3$) to affect the classification decision.

Bernoulli NB:
Smoothing is
based on the
number of
classes

- 6 terms in Vocabulary 6 terms in calculation of $P(C|X)$
- Count single occurrence of term, e.g., Chinese is just counted once

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Decision Rule for Bernoulli Model

Use all vocabulary for classification (regardless if the word occurred or not in the test example)

► **Table 13.3** Multinomial versus Bernoulli model.

	multinomial model	Bernoulli model
event model	generation of token	generation of document
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
document representation	$d = \langle t_1, \dots, t_k, \dots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \dots, e_i, \dots, e_M \rangle, e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t c)$	$\hat{P}(U_i = e c)$
decision rule: maximize multiple occurrences	$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k c)$ taken into account	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i c)$ ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for term the	$\hat{P}(X = \text{the} c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1 c) \approx 1.0$

Multinomial versus Bernoulli NB

To reduce the number of parameters, we make the Naive Bayes *conditional independence assumption*. We assume that attribute values are independent of each other given the class:

$$\text{Multinomial} \quad P(d|c) = P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

$$\text{Bernoulli} \quad P(d|c) = P(\langle e_1, \dots, e_M \rangle | c) = \prod_{1 \leq i \leq M} P(U_i = e_i | c).$$

We have introduced two random variables here to make the two different generative models explicit. X_k is the random variable for position k in the document and takes as values terms from the vocabulary. $P(X_k = t | c)$ is the probability that in a document of class c the term t will occur in position k . U_i is the random variable for vocabulary term i and takes as values 0 (absence) and 1 (presence). $P(U_i = 1 | c)$ is the probability that in a document of class c the term t_i will occur – in any position and possibly multiple times.

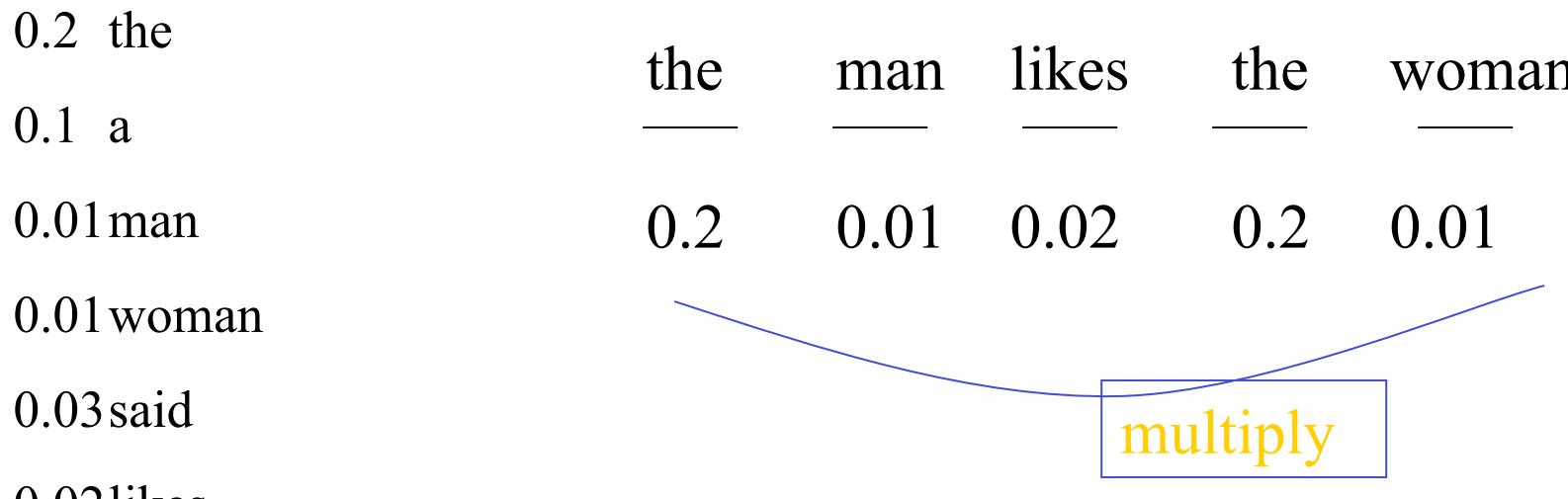
We illustrate the conditional independence assumption in Figures 13.4 and 13.5. The class *China* generates values for each of the five term attributes (multinomial) or six binary attributes (Bernoulli) with a certain probability, independent of the values of the other attributes. The fact that a document in the class *China* contains the term *Taipei* does not make it more likely or less likely that it also contains *Beijing*.

- Multinomial Naïve Bayes

Stochastic Language Models

- Models *probability* of generating strings (each word in turn) in the language (commonly all strings over Σ). E.g., unigram model

Model M



Stochastic Language Models

- Model *probability* of generating any string

Model M1	Model M2	the	class	pleaseth	yon	maiden
0.2 the	0.2 the	—	—	—	—	—
0.01 class	0.0001 class	0.2	0.01	0.0001	0.0001	0.0005
0.0001 sayst	0.03 sayst	0.2	0.0001	0.02	0.1	0.01
0.0001 pleaseth	0.02 pleaseth	0.2	0.0001	0.02	0.1	0.01
0.0001 yon	0.1 yon	—	—	—	—	—
0.0005 maiden	0.01 maiden	—	—	—	—	—
0.01 woman	0.0001 woman	—	—	—	—	—

$P(s|M2) > P(s|M1)$

Unigram and higher-order models

$$\begin{aligned} & P(\bullet \bullet \bullet \bullet) \\ \cdot & = P(\bullet) P(\bullet | \bullet) P(\bullet | \bullet \bullet) P(\bullet | \bullet \bullet \bullet) \end{aligned}$$

- **Unigram Language Models**

$$P(\bullet) P(\bullet) P(\bullet) P(\bullet)$$

- **Bigram (generally, n -gram) Language Models**

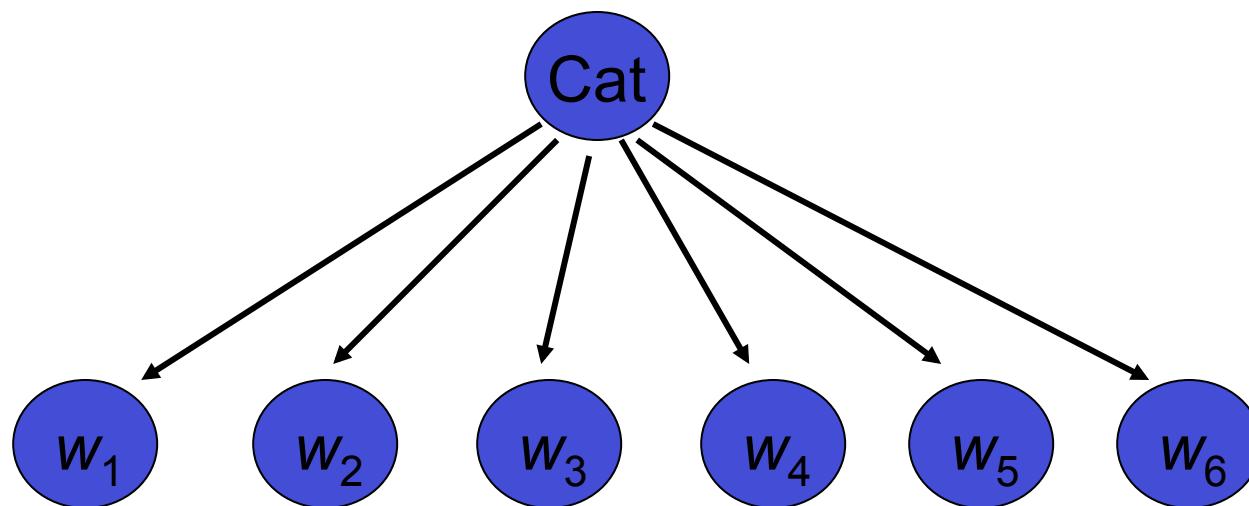
- **Other Language Models**

- Grammar-based models (PCFGs), etc.

- Probably not the first thing to try in IR



Naïve Bayes via a class conditional language model = multinomial NB



- Effectively, the probability of each class is done as a class-specific unigram language model

Using Multinomial Naive Bayes Classifiers to Classify Text: Basic method

- Attributes are text positions, values are words.

$$\begin{aligned} c_{NB} &= \operatorname{argmax}_{c_j \in C} P(c_j) \prod_i P(x_i | c_j) \\ &= \operatorname{argmax}_{c_j \in C} P(c_j) P(x_1 = "our" | c_j) \cdots P(x_n = "text" | c_j) \end{aligned}$$

- Still too many possibilities
- Assume that classification is *independent* of the positions of the words
 - Use same parameters for each position
 - Result is bag of words model (over tokens not types)

Naïve Bayes: Learning Algorithm

- From training corpus, extract *Vocabulary*
- Calculate required $P(c_j)$ and $P(x_k | c_j)$ terms
 - For each c_j in C do
 - $docs_j \leftarrow$ subset of documents for which the target class is c_j
 - $P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$

Words can be sparse: Smoothing: e.g., Laplace or Add-one smoothing can be interpreted as a uniform prior (each term occurs once for each class) that is then updated as evidence from the training data comes in

- $Text_j \leftarrow$ single document containing all $docs_j$
- for each word x_k in *Vocabulary*
 - $n_k \leftarrow$ number of occurrences of x_k in $Text_j$
 - $P(x_k | c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |Vocabulary|}$

n = total number of tokens
(alpha is “tuning/smoothing” factor that can be set to 1)
Multinomial Conditional Independence Naïve Bayes:
Learning from Training Set

Naïve Bayes: Classifying

- **positions** \leftarrow all word positions in current document which contain tokens found in *Vocabulary*
- **Return** c_{NB} , where

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j)$$

Naïve Bayes: Classifying

- positions \leftarrow all word positions in current document which contain tokens found in *Vocabulary*
- Return c_{NB} , where

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j)$$

Classify a NEW document D with 100 words (not seen in the class=TRUE)
 $P(W|Class=TRUE) = 1/(10^6 + 10^7)$ ##default probability of a word given Class=TRUE

Assume a Vocabulary of 10^6 words; number of words in class=TRUE is 10^7

THEN

$$\Pr(\text{Class} = \text{TRUE} | D) \sim (1/(10^6 + 10^7))^{100} \times \text{Prior}$$

Underflow Prevention: log space

- Multiplying lots of probabilities, which are between 0 and 1, can result in floating-point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, it is better to perform all computations by *summing logs of probabilities rather than multiplying probabilities*.
- Class with highest final un-normalized log probability score is still the most probable

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j) \quad \# \operatorname{Log}\left(\frac{a}{b}\right) = \log(a) - \log(b); \text{Note : } \log(1) = 0$$

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j)$$

- Note that model is now just max of sum of weights...

Log Rules Refresher

$$\log_a(bc) = \log_a(b) + \log_a(c)$$

$$\log_a(b^c) = c \log_a(b)$$

$$\log_a(1/b) = -\log_a(b)$$

$$\log_a(1) = 0$$

$$\log_a(a) = 1$$

<http://grockit.com/blog/act-logarithms-explained/>

$$\log_a(a^r) = r$$

$$\log_{1/a}(b) = -\log_a(b)$$

$$\log_a(b) \log_b(c) = \log_a(c)$$

$$\log_b(a) = \frac{1}{\log_a(b)}$$

$$\log_{a^m}(a^n) = \frac{n}{m}, \quad m \neq 0$$

Naïve Bayes: Classifying: Extreme example!

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j)$$

- **positions** \leftarrow all word positions in current document which contain tokens found in *Vocabulary*
- Return c_{NB} , where

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j) \quad \# \operatorname{Log}\left(\frac{a}{b}\right) = \log(a) - \log(b); \text{Note: } \log(1) = 0$$

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j)$$

In R
> log (1/(10⁶ + 10⁷)) * 100
[1] -1621.341
> exp(-1621.341)
[1] 0 #super small number; we underflow

Classify a NEW document D with 100 words (all 100 words are not seen in the class=TRUE)
 $P(W|Class=TRUE) = 1/(10^6 + 10^7)$ ##default probability of a word given Class=TRUE

Assume a Vocabulary of 10^6 words; number of words in class=TRUE is 10^7

THEN

$$\Pr(\text{Class} = \text{TRUE} | D) \sim (1/(10^6 + 10^7))^{100} \times \text{Prior}$$

Approximate with log base 10 (assume $(10^6 + 10^7) = 10^7$)
 $P(W|Class=TRUE) \approx (0 - \log(10^7, 10)) \times 100 = \sim -7 \times 100 = -700$ #in log space

Naive Bayes: Time Complexity

- **Training Time:** $O(|D|L_d + |C||V|)$ where
 L_d is the average length of a document in D .
 - Assumes V and all D_i , n_i , and n_{ij} pre-computed in $O(|D|L_d)$ time during one pass through all of the data.
 - Generally just $O(|D|L_d)$ since usually $|C||V| < |D|L_d$
- **Test Time:** $O(|C| L_t)$ where
 L_t is the average length of a test document.
 - Very efficient overall, linearly proportional to the time needed to just read in all the data.
 - Plus, robust in practice



- **Exercise**

Exercise: Multinomial Naive Bayes

	docID	words in document	in $c = China?$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

- Estimate parameters of Multinomial Naive Bayes classifier
- Classify test document

Exercise: Multinomial Naive Bayes

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Priors $\Pr(\text{Class}=\text{China}) = \frac{3}{4}$ $\Pr(\text{Class}=\text{Not China}) = \frac{1}{4}$

Class conditional probabilities; likelihoods

$\Pr(\text{Chinese} | \text{Class}=\text{China}) = \frac{5}{8}$

$\Pr(\text{Chinese} | \text{Class}=\text{NOT China}) = \frac{1}{3}$

What is the class of this document {Tokyo,chinese}

$\Pr(\text{Class} = \text{China} | \{\text{Tokyo},\text{chinese}\}) = \frac{3}{4} \times \frac{5}{8} \times \frac{1}{3} = 0$

$\Pr(\text{Class} = \text{Not China} | \{\text{Tokyo},\text{chinese}\}) = \frac{1}{4} \times \frac{1}{3} \times \frac{2}{3} = \frac{1}{36}$

$\Pr(\text{Class} = \text{China} | \{\text{Tokyo},\text{chinese}\}) = 0 / 0 + \frac{1}{36} = 0$

$\Pr(\text{Class} = \text{Not China} | \{\text{Tokyo},\text{chinese}\}) = \frac{1}{36} / 0 + \frac{1}{36} = 1$

$$P(C | D) = \frac{P(D | C)P(C)}{P(D)}$$

$\Pr(\text{Class} = \text{China} | \{\text{Tokyo},\text{chinese}\}) = \frac{3}{4} \times 0 + \frac{1}{8} \times 6 \times \frac{5}{8} \times \frac{1}{3} = 0.023$

$\Pr(\text{Class} = \text{Not China} | \{\text{Tokyo},\text{chinese}\}) = \frac{1}{4} \times 1 + \frac{1}{8} \times 6 \times \frac{1}{3} + \frac{1}{3} = 0.012$

$\Pr(\text{Class} = \text{China} | \{\text{Tokyo},\text{chinese}\}) = 0.023 / (0.023 + 0.012) = 0.65$

$\Pr(\text{Class} = \text{Not China} | \{\text{Tokyo},\text{chinese}\}) = 1 - 0.65 = 0.35$

Exercise: Multinomial Naive Bayes

	docID	words in document	in $c = China?$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$\Pr(\text{Class}=\text{China}) = \frac{3}{4} \quad \Pr(\text{Class} = \text{not China}) = \frac{1}{4}$$

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

$$\Pr(\text{Chinese}|\text{Class} = \text{China}) = 5 / 8, 1/8, 1/8, 1/8 = 1$$

TEST = {Japan, Chinese}

$$\Pr(\text{class} = \text{China} | \{\text{Japan}, \text{Chinese}\}) = \text{Prior} \times \text{Likelihood} = \frac{3}{4} \times (5/14 \times 1/14) = 0.022; \quad P(C|D) = 0.022/(0.022+0.0123) = 0.64$$

$$\Pr(\text{class} = \text{NOT China} | \{\text{Japan}, \text{Chinese}\}) = \text{Prior} \times \text{Likelihood} = \frac{1}{4} \times (1+1/(3+6) \times 1+1/(3+6) = 0.0123; \quad P(C|D) = 1 - 0.64$$

6 unique

$$\Pr(\text{Chinese}|\text{Class} = \text{China}) = 5+1/ (8+6)$$

- Estimate parameters of Multinomial Naive Bayes classifier

- Classify test document

Example: Parameter estimates

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditional probabilities:

$$\hat{P}(\text{CHINESE}|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) = (0 + 1)/(8 + 6) = 1/14$$

$$\hat{P}(\text{CHINESE}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

$$\hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Classification of d_5

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to $c = China$. The reason for this classification decision is that the three occurrences of the positive indicator CHINESE in d_5 outweigh the occurrences of the two negative indicators JAPAN and TOKYO.

Note: Two Models: Multivariate Bernoulli

- **Model 1: Multivariate Bernoulli**
 - One feature X_w for each word in dictionary
 - X_w = true in document d if w appears in d
 - Naive Bayes assumption:
 - Given the document's topic, appearance of one word in the document tells us nothing about chances that another word appears
- **This is the model used in the binary independence model in classic probabilistic relevance feedback in hand-classified data**

Two Models: Multinomial NB

- **Model 2: Multinomial = Class conditional unigram**
 - One feature X_i for each word pos in document
 - feature's values are all words in dictionary
 - Value of X_i is the word in position i
 - Naïve Bayes assumption:
 - Given the document's topic, word in one position in the document tells us nothing about words in other positions
 - Second assumption:
 - Word appearance does not depend on position

$$P(X_i = w | c) = P(X_j = w | c)$$

for all positions i, j , word w , and class c

- **Puzzle**

Example: Parameter estimates

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	taiwan

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditional probabilities:

$$\hat{P}(\text{CHINESE}|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) = (0 + 1)/(8 + 6) = 1/14$$

$$\hat{P}(\text{CHINESE}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

$$\hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Parameter estimates

	docID	words in document	in $c = \text{China}$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditioned on the training set, we have:

$$\hat{P}(\text{CHINESE}|c) = (3+1)/(8+6) = 4/14 = 2/7$$

$$\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) = (0+1)/(8+6) = 1/14$$

$$\hat{P}(\text{CHINESE}|\bar{c}) = (1+1)/(3+6) = 2/9$$

$$\hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) = (1+1)/(3+6) = 2/9$$

Which type of Naïve Bayes model is this?

The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.

Example: Parameter estimates

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditional probabilities:

$$\hat{P}(\text{CHINESE}|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{Tokyo}|\bar{c}) = \hat{P}(\text{Japan}|\bar{c}) = (0 + 1)/(8 + 6) = 1/14$$

Which type of Naïve Bayes model is this?

Which type of Naïve Bayes model is this?

Multinomial....look at $Pr(\text{Chinese}|c)$ is made up of 5 occurrences of the word in the class c (versus 3 in the Bernoulli model for the same class conditional, $Pr(\text{Chinese}|\bar{c})$)

Underflow Prevention: log space

- Multiplying lots of probabilities, which are between 0 and 1, can result in floating-point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, it is better to perform all computations by *summing logs of probabilities rather than multiplying probabilities*.
- Class with highest final un-normalized log probability score is still the most probable.

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j)$$

- Note that model is now just max of sum of weights...

Multinomial vs Bernoulli Naïve Bayes

► Table 13.3 Multinomial versus Bernoulli model.

	multinomial model	Bernoulli model
event model	generation of token	generation of document
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
document representation	$d = \langle t_1, \dots, t_k, \dots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \dots, e_i, \dots, e_M \rangle, e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t c)$	$\hat{P}(U_i = e c)$
decision rule: maximize	$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for term the	$\hat{P}(X = \text{the} c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1 c) \approx 1.0$

Multinomial Naïve Bayes for text

• ..

$$\begin{aligned} P(Y = y_k | X_1, X_2, \dots, X_N) &= \frac{P(Y=y_k)P(X_1, X_2, \dots, X_N | Y=y_k)}{\sum_j P(Y=y_j)P(X_1, X_2, \dots, X_N | Y=y_j)} \\ &= \frac{P(Y=y_k)\Pi_i P(X_i | Y=y_k)}{\sum_j P(Y=y_j)\Pi_i P(X_i | Y=y_j)} \end{aligned}$$

$$Y \leftarrow argmax_{y_k} P(Y = y_k)\Pi_i P(X_i | Y = y_k)$$

Naïve Bayes Classifier for Text

- Given the training data what are the parameters to be estimated?

$$P(Y)$$

Diabetes : 0.8
Hepatitis : 0.2

$$P(X|Y_1)$$

the: 0.001
diabetic : 0.02
blood : 0.0015
sugar : 0.02
weight : 0.018
...

$$P(X|Y_2)$$

the: 0.001
diabetic : 0.0001
water : 0.0118
fever : 0.01
weight : 0.008
...

13.2 Naive Bayes text classification

MULTINOMIAL NAIVE
BAYES

The first supervised learning method we introduce is the *multinomial Naive Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document d being in class c is computed as

$$(13.2) \quad P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c .¹ We interpret $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class. $P(c)$ is the prior probability of a document occurring in class c . If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability. $\langle t_1, t_2, \dots, t_{n_d} \rangle$ are the tokens in d that are part of the vocabulary we use for classification and n_d is the number of such tokens in d . For example, $\langle t_1, t_2, \dots, t_{n_d} \rangle$ for the one-sentence document *Beijing and Taipei join the WTO* might be $\langle \text{Beijing}, \text{Taipei}, \text{join}, \text{WTO} \rangle$, with $n_d = 4$, if we treat the terms and and the as stop words.

MAXIMUM A
POSTERIORI CLASS

In text classification, our goal is to find the *best* class for the document. The best class in NB classification is the most likely or *maximum a posteriori* (MAP) class c_{map} :

$$(13.3) \quad c_{\text{map}} = \arg \max_{c \in \mathcal{C}} \hat{P}(c|d) = \arg \max_{c \in \mathcal{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c).$$

[http://nlp.stanford.edu/IR-book/pdf/
13bayes.pdf](http://nlp.stanford.edu/IR-book/pdf/13bayes.pdf)

We write \hat{P} for P because we do not know the true values of the parameters $P(c)$ and $P(t_k|c)$, but estimate them from the training set as we will see in a moment.

In Equation (13.3), many conditional probabilities are multiplied, one for each position $1 \leq k \leq n_d$. This can result in a floating point underflow. It is therefore better to perform the computation by adding logarithms of probabilities instead of multiplying probabilities. The class with the highest log probability score is still the most probable; $\log(xy) = \log(x) + \log(y)$ and the logarithm function is monotonic. Hence, the maximization that is

• ..

ADD-ONE SMOOTHING

To eliminate zeros, we use *add-one* or *Laplace smoothing*, which simply adds one to each count (cf. Section 11.3.2):

$$(13.7) \quad \hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B},$$

where $B = |V|$ is the number of terms in the vocabulary. Add-one smoothing can be interpreted as a uniform prior (each term occurs once for each class) that is then updated as evidence from the training data comes in. Note that this is a prior probability for the occurrence of a *term* as opposed to the prior probability of a *class* which we estimate in Equation (13.5) on the document level.

► Table 13.1 Data for parameter estimation examples.

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

NB example

► Table 13.2 Training and test times for NB.

mode	time complexity
training	$\Theta(D L_{ave} + C V)$
testing	$\Theta(L_a + C M_a) = \Theta(C M_a)$

We have now introduced all the elements we need for training and applying an NB classifier. The complete algorithm is described in Figure 13.2.

Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\ \hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\ \hat{P}(\text{Chinese}|\bar{c}) &= (1+1)/(3+6) = 2/9 \\ \hat{P}(\text{Tokyo}|\bar{c}) = \hat{P}(\text{Japan}|\bar{c}) &= (1+1)/(3+6) = 2/9\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\bar{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms.

We then get:

$$\begin{aligned}\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003. \\ \hat{P}(\bar{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.\end{aligned}$$

Thus, the classifier assigns the test document to $c = China$. The reason for this classification decision is that the three occurrences of the positive indicator Chinese in d_5 outweigh the occurrences of the two negative indicators Japan and Tokyo.

<http://nlp.stanford.edu/IR-book/pdf/13bayes.pdf>

Multinomial NaiveBayes: Notebook

- <https://www.dropbox.com/s/2qsnmcsxphkcowl/Naive%20Bayes-Not-Scaleable.ipynb?dl=0>

Data

class:word1,word2,...

```
|: %%writefile NaiveBayes.txt  
China:Chinese,Beijing,Chinese  
China:Chinese,Chinese,Shanghai  
China:Chinese,Macao  
Not China:Tokyo,Japan,Chinese
```

Writing NaiveBayes.txt

Challenge build a Multinomial NB

- Being with a word count for each class

Problematic Naïve Bayes Code

<https://www.dropbox.com/s/2qsnmcsxhphkcw/Naive%20Bayes-Not-Scaleable.ipynb?dl=0>

```
1 from __future__ import division
2 RDD_data = sc.textFile("NaiveBayes.txt").cache()
3 RDD_data_China = RDD_data.filter(lambda line: line.split(":")[0]=='China') #filter out lines class=China
4 RDD_data_notChina = RDD_data.filter(lambda line: line.split(":")[0]=='Not China') #filter out lines class=Not China
5 num_docs_China = RDD_data_China.count()
6 num_docs_notChina = RDD_data_notChina.count()
7 China_wordcount = RDD_data_China.flatMap(lambda line: line.split(":")[1].split(',')) \
8     .map(lambda x: (x,1)) \
9     .reduceByKey(lambda a,b:a+b) \
10    .collect()
11 notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(',')) \
12     .map(lambda x: (x,1)) \
13     .reduceByKey(lambda a,b:a+b) \
14    .collect()
15 total_word_China = sum([data[1] for data in China_wordcount])
16 total_word_notChina = sum([data[1] for data in notChina_wordcount])
17 num_distinct_word = RDD_data.flatMap(lambda line: line.split(":")[1].split(',')).distinct().count()
```

**Challenge: identify issues
And then fix them**

Multinomial NaiveBayes: Notebook

```
: 1 from __future__ import division
: 2 RDD_data = sc.textFile("NaiveBayes.txt").cache()
: 3 RDD_data_China = RDD_data.filter(lambda line: line.split(":")[0]=='China') #filter out lines class=China
: 4 RDD_data_notChina = RDD_data.filter(lambda line: line.split(":")[0]=='Not China') #filter out lines class=Not China
: 5 num_docs_China = RDD_data_China.count()
: 6 num_docs_notChina = RDD_data_notChina.count()
: 7 China_wordcount = RDD_data_China.flatMap(lambda line: line.split(":")[1].split(',')) \
: 8     .map(lambda x: (x,1)) \
: 9     .reduceByKey(lambda a,b:a+b) \
:10    .collect()
:11 notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(',')) \
:12     .map(lambda x: (x,1)) \
:13     .reduceByKey(lambda a,b:a+b) \
:14    .collect()
:15 total_word_China = sum([data[1] for data in China_wordcount])
:16 total_word_notChina = sum([data[1] for data in notChina_wordcount])
:17 num_distinct_word = RDD_data.flatMap(lambda line: line.split(":")[1].split(',')).distinct().count()

: num_docs_China / (num_docs_China + num_docs_notChina) #Prior for class China
: 0.75

: num_docs_notChina / (num_docs_China + num_docs_notChina) #Prior for class not China
: 0.25
```

Solution to Multinomial NB

```
In [51]: sc.stop()
```

```
In [4]: from __future__ import division
RDD_data = sc.textFile("NaiveBayes.txt").cache()
RDD_data_China = RDD_data.filter(lambda line: line.startswith("C"))
RDD_data_notChina = RDD_data.filter(lambda line: line.startswith("N"))
num_docs_China = RDD_data_China.count()
num_docs_notChina = RDD_data_notChina.count()
China_wordcount = RDD_data_China.flatMap(lambda line: line.split(":")[1].split(","))
    .map(lambda x: (x,1))
    .reduceByKey(lambda a,b:a+b)
    .collect()
notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(","))
    .map(lambda x: (x,1))
    .reduceByKey(lambda a,b:a+b)
    .collect()
total_word_China = sum([data[1] for data in China_wordcount])
total_word_notChina = sum([data[1] for data in notChina_wordcount])
num_distinct_word = RDD_data.flatMap(lambda line: line.split(":")[1].split(",")).distinct().count()
```

<https://www.dropbox.com/s/9g0bu6m6v5sjb27/Naive%20Bayes-Scalable.ipynb?dl=0>

```
In [5]: num_docs_China / (num_docs_China + num_docs_notChina) #Prior for class China
```

```
Out[5]: 0.75
```

```
In [12]:
```

```
notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(","))
    .map(lambda x: (x,1))
    .reduceByKey(lambda a,b:a+b)
```

```
notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(","))
    .map(lambda x: (x,1))
    .reduceByKey(lambda a,b:a+b)
China_wordcount.map(lambda x: x[1]).reduce(lambda a, b: a+b)
```

```
#Should return 8
```

Solution to Multinomial NB

- ..
<https://www.dropbox.com/s/9g0bu6m6v5sjb27/Naive%20Bayes-Scalable.ipynb?dl=0>

```
from __future__ import division
RDD_data = sc.textFile("NaiveBayes.txt").cache()
RDD_data_China = RDD_data.filter(lambda line: line.split(":")[0]=='China') #filter out lines class=China
RDD_data_notChina = RDD_data.filter(lambda line: line.split(":")[0]=='Not China') #filter out lines class=Not China
num_docs_China = RDD_data_China.count()
num_docs_notChina = RDD_data_notChina.count()

China_wordcount = RDD_data_China.flatMap(lambda line: line.split(":")[1].split(' '))
    .map(lambda x: (x,1)) \
    .reduceByKey(lambda a,b:a+b)
total_word_China = China_wordcount.map(lambda x:x[1]).reduce(lambda a,b: a+b)
notChina_wordcount = RDD_data_notChina.flatMap(lambda line: line.split(":")[1].split(' '))
    .map(lambda x: (x,1)) \
    .reduceByKey(lambda a,b:a+b)
total_word_notChina = notChina_wordcount.map(lambda x:x[1]).reduce(lambda a,b: a+b)

num_distinct_word = RDD_data.flatMap(lambda line: line.split(":")[1].split(',')).distinct().count()
```

Spark MLLib: NaiveBayes Example

- **NaiveBayes implements multinomial naive Bayes.** It takes an RDD of LabeledPoint and an optionally smoothing parameter lambda as input, and output a NaiveBayesModel, which can be used for evaluation and prediction.
- **Note that the Python API does not yet support model save/load but will in the future.**

Data set

-

MLLib Naive Bayes

For more details see: <http://spark.apache.org/docs/latest/mllib-naive-bayes.html>

```
: %%writefile sample_naive_bayes_data.txt
0,1 0 0
0,2 0 0
1,0 1 0
1,0 2 0
2,0 0 1
2,0 0 2
```

Overwriting sample_naive_bayes_data.txt

Sample Notebook

```
1 from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
2 from pyspark.mllib.linalg import Vectors
3 from pyspark.mllib.regression import LabeledPoint
4
5 def parseLine(line):
6     parts = line.split(',')
7     label = float(parts[0])
8     features = Vectors.dense([float(x) for x in parts[1].split(' ')])
9     return LabeledPoint(label, features)
10
11 data = sc.textFile('sample_naive_bayes_data.txt').map(parseLine)
12
13 # Split data approximately into training (60%) and test (40%)
14 training, test = data.randomSplit([0.6, 0.4], seed = 0)
15
16 # Train a naive Bayes model.
17 model = NaiveBayes.train(training, 1.0)
18
19 # Make prediction and test accuracy.
20 predictionAndLabel = test.map(lambda p : (model.predict(p.features), p.label))
21 accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x == v).count() / test.count()
22
23 # Save and load model
24 model.save(sc, "myModelPath")
25 sameModel = NaiveBayesModel.load(sc, "myModelPath")
```

• ..

```
: 1 test.collect()  
:  
: [LabeledPoint(0.0, [1.0,0.0,0.0]),  
 LabeledPoint(1.0, [0.0,2.0,0.0]),  
 LabeledPoint(2.0, [0.0,0.0,2.0])]
```

```
: 1 for record in test.collect():  
 2     print record.label, record.features
```

```
0.0 [1.0,0.0,0.0]  
1.0 [0.0,2.0,0.0]  
2.0 [0.0,0.0,2.0]
```

```
.
```

Sample Notebook

```
1 from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
2 from pyspark.mllib.linalg import Vectors
3 from pyspark.mllib.regression import LabeledPoint
4
5 def parseLine(line):
6     parts = line.split(',')
7     label = float(parts[0])
8     features = Vectors.dense([float(x) for x in parts[1].split(' ')])
9     return LabeledPoint(label, features)
10
11 data = sc.textFile('sample_naive_bayes_data.txt').map(parseLine)
12
13 # Split data approximately into training (60%) and test (40%)
14 training, test = data.randomSplit([0.6, 0.4], seed = 0)
15
16 # Train a naive Bayes model.
17 model = NaiveBayes.train(training, 1.0)
18
19 # Make prediction and test accuracy.
20 predictionAndLabel = test.map(lambda p : (model.predict(p.features), p.label))
21 accuracy = 1.0 * predictionAndLabel.filter(lambda (x, v): x == v).count() / test.count()
22
23 # Save and load model
24 model.save(sc, "myModelPath")
25 sameModel = NaiveBayesModel.load(sc, "myModelPath")
```

Naïve Bayes in MLLib (no need to cache)

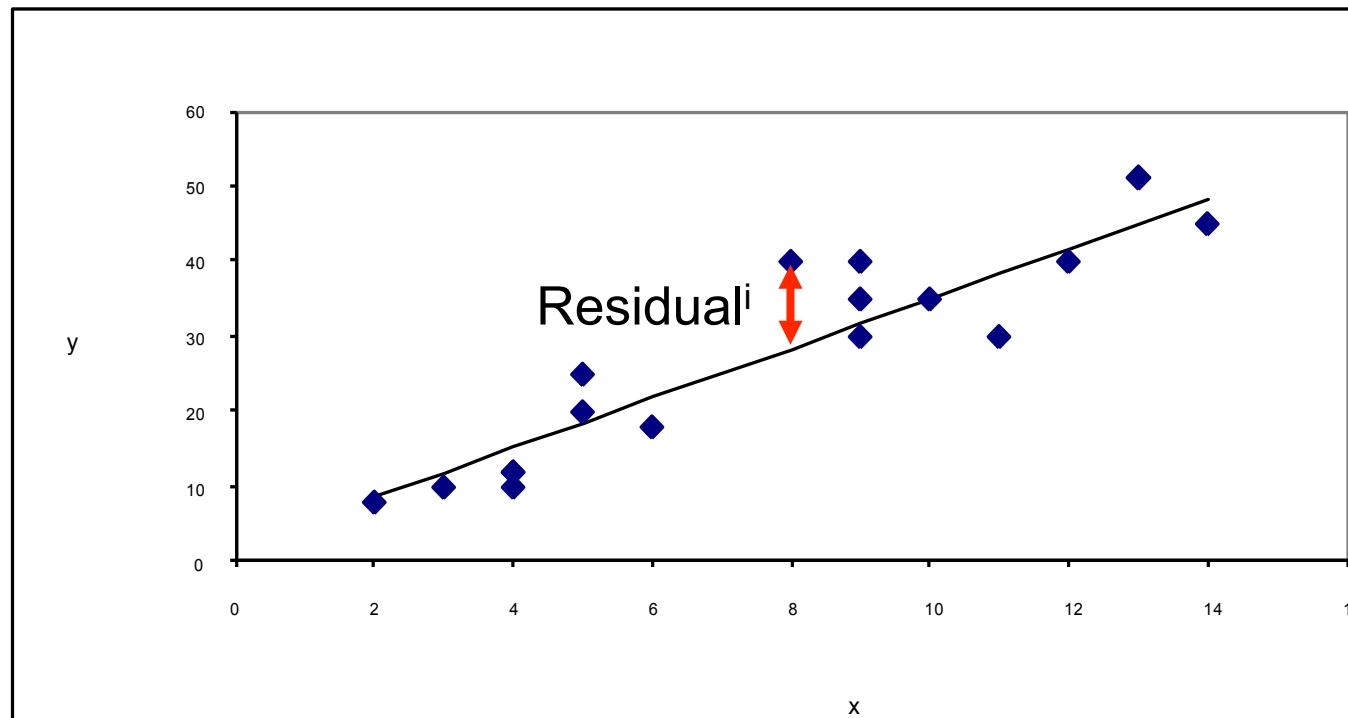
- **MLlib supports multinomial naive Bayes and Bernoulli naive Bayes.**
 - These models are typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes).
 - Feature values must be nonnegative.
 - The model type is selected with an optional parameter “multinomial” or “bernoulli” with “multinomial” as the default.
 - Additive smoothing can be used by setting the parameter λ (default to 1).
 - For document classification, the input feature vectors are usually sparse, and sparse vectors should be supplied as input to take advantage of sparsity.
 - Since the training data is only used once, it is not necessary to cache it.

Equation of a line

$$J_q(W, X_1^m) = \text{Minimize} \sum_{i=1}^m (W^T X_i - y_i)^2$$

$$\text{Residual}^i = (WX^i - y^i)$$

$$\text{Residual}^i = (WX^i - y^i)^2$$



$$y = mx + b$$
$$Y = w_1 * x + w_0$$

Where w_1, w_0 are model parameters

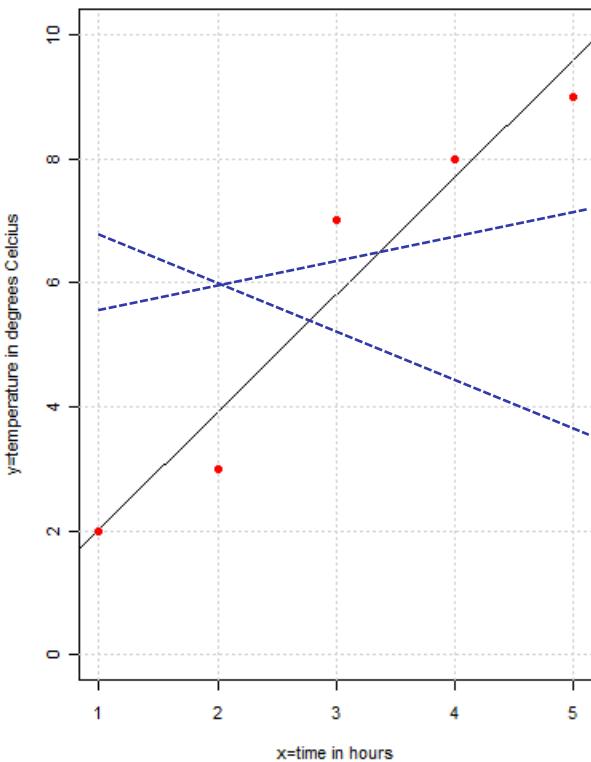
Each pair yields a different sum of squares error

Version Space, Error surface

$$Y = mx + b$$

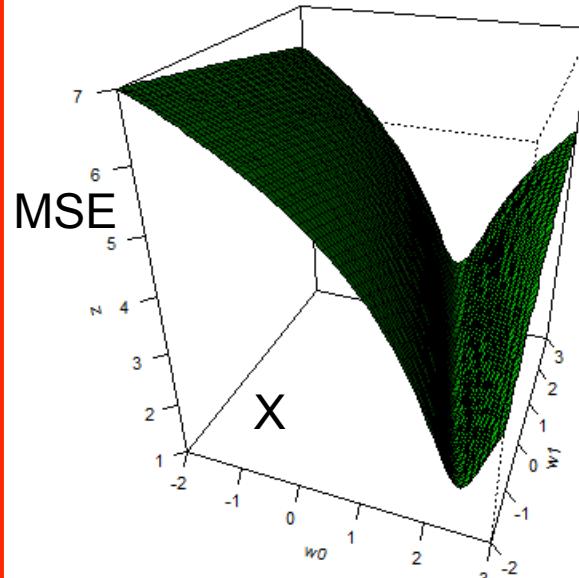
$$Y = w_1x + w_0$$

Temperature Dataset



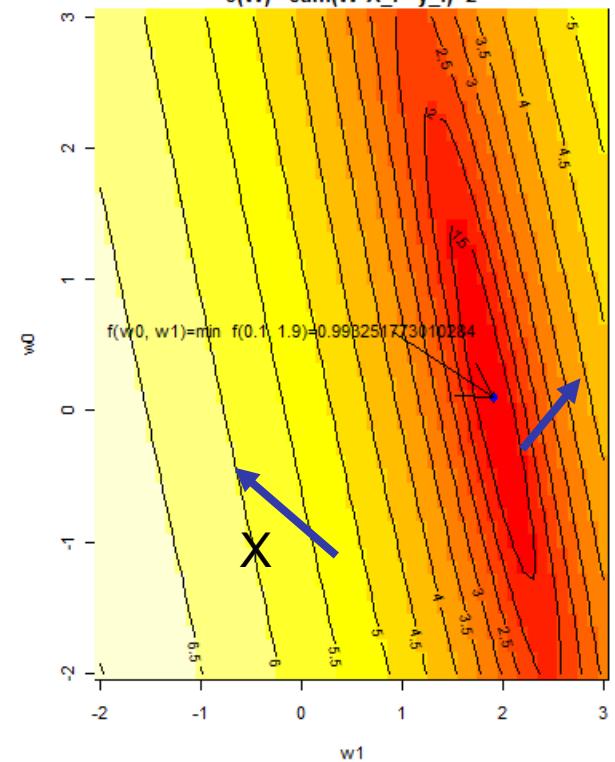
$$MSE = \frac{1}{n} \sum \text{Residual}^i = \frac{1}{n} \sum (WX^i - y^i)^2$$

Error Surface in 3D
 $J(W) = \sum (W^*X_i - y_i)^2$



\$coefficients
[1] 0.09644596, 1.90098441
\$iterations
[1] 658
\$SSE
[1] 2.700011

HeatMap and Contour Plot of
Error Surface
 $J(W) = \sum (W^*X_i - y_i)^2$



Problem Domain vs

Version Space

HeatMap and Contour Plot in R

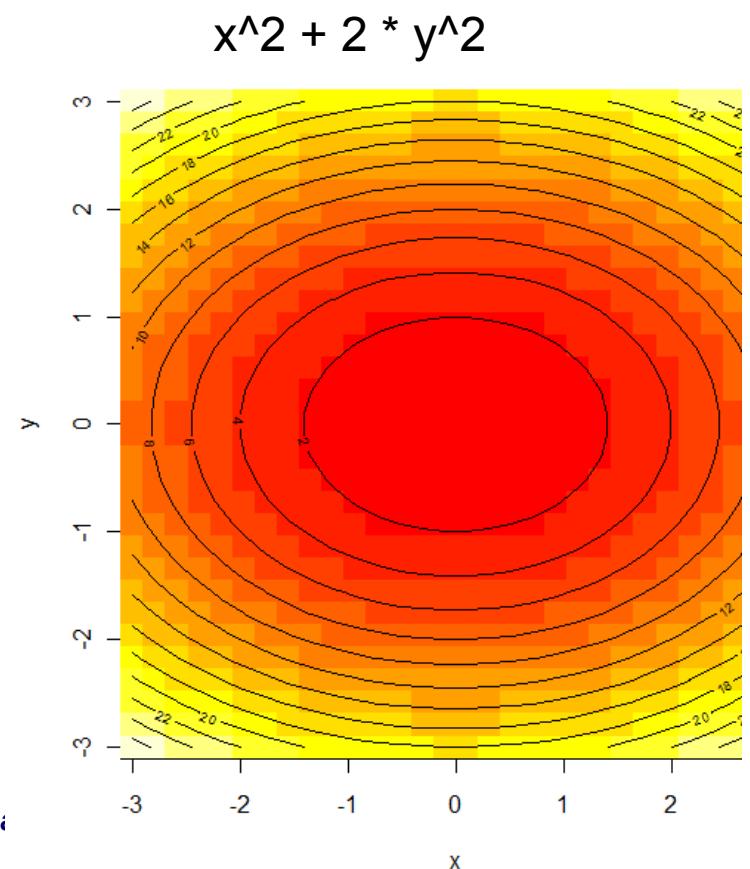
```
x <- seq(-3, 3, length= 30);  y <- x
```

```
f <- function(x,y) { x^2 + 2 * y^2 }
```

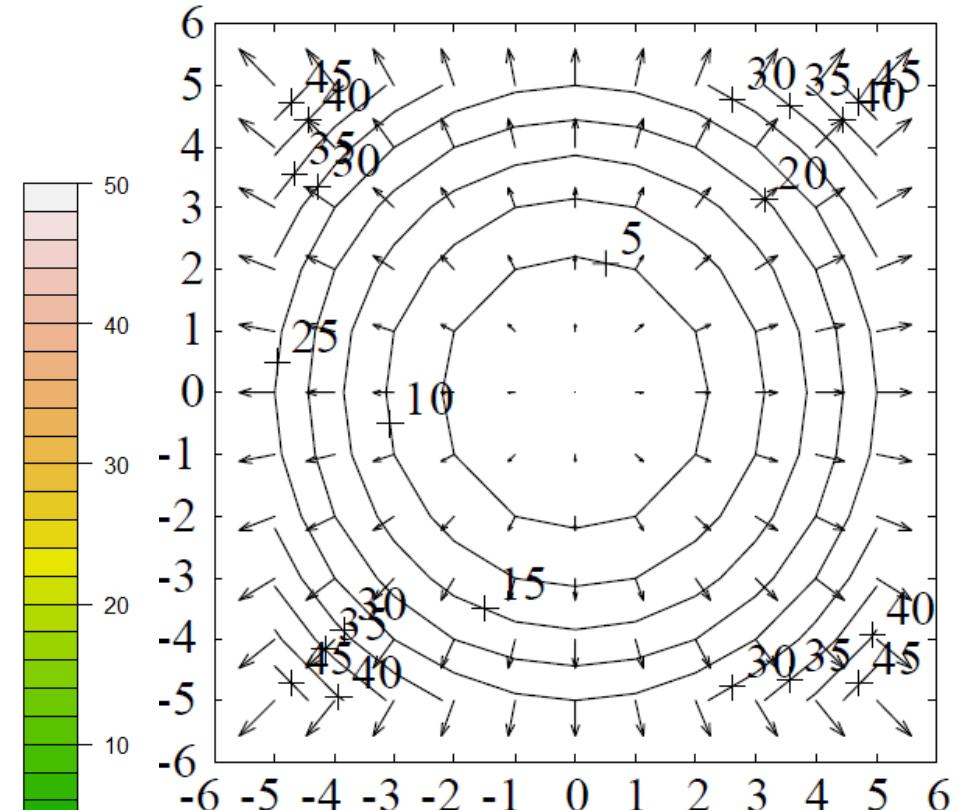
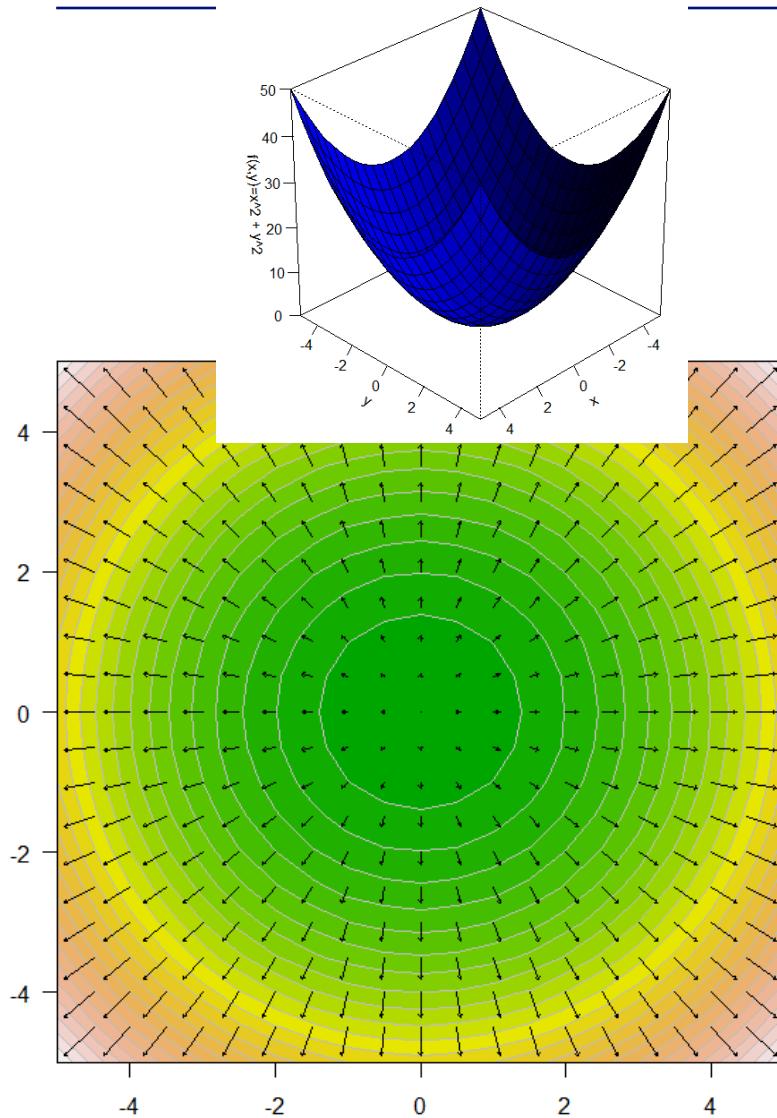
```
z <- outer(x, y, f)
```

- **#new plot of isolines and heatmap**
- **image(x,y,z) #heat image of surface**
- **contour(x,y,z, add=TRUE) #add contours**

See example.gradientPlots()



Gradient Vector Plots of $f(x,y)=x^2+y^2$



3. Superimpose the level curves of $f(x, y) = x^2 + y^2$

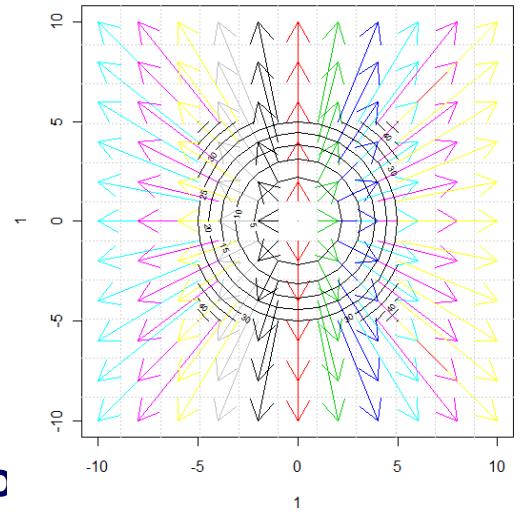
<http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf>

Ugly Unnormalized Gradient Plot in R

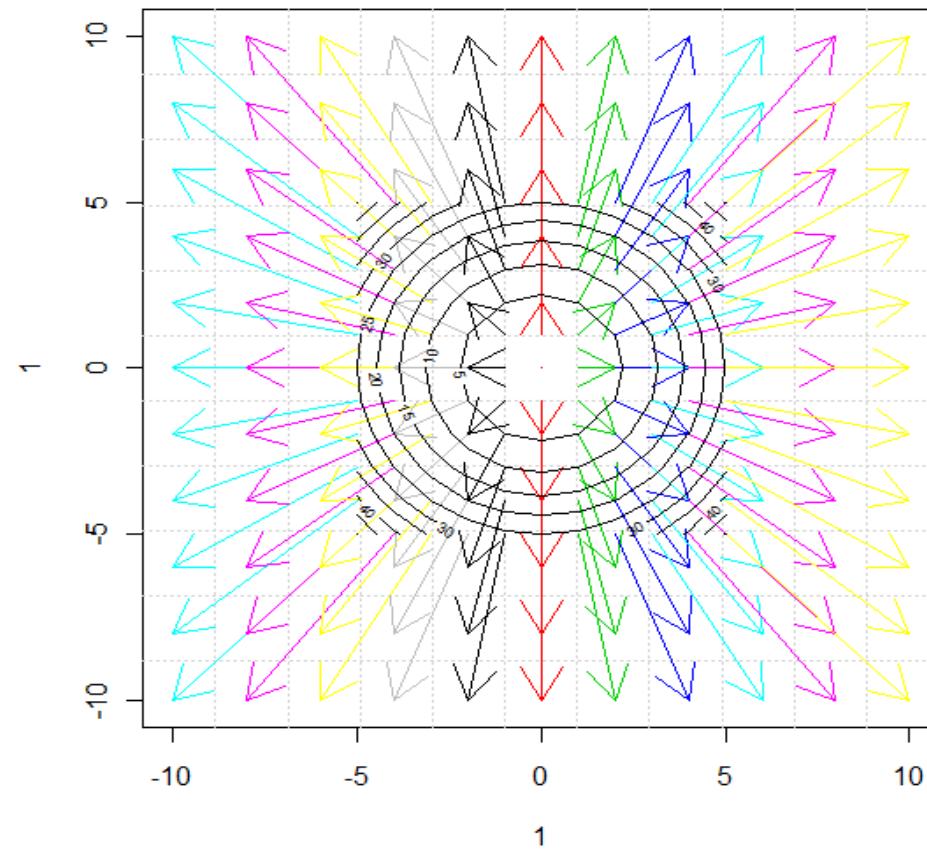
```
#ugly unnormalized gradient vector plot
# plot vector plot and contour plot

x <- -5:5
y <- x
f <- function(x,y) { x^2 + y^2 }
z <- outer(x, y, f)
u=2*x #fprime_x(x,y) = df/dx =d(x^2 + y^2)/dx=2x
v=2*y
plot(1, 1, xlim=c(-10, 10), ylim=c(-10, 10), pch="")
grid(length(x))
for(i in x) {
  for (j in y) {
    arrows(i,j, 2*i, 2*j, col=i+10) #(f'x(x,y), f'y(x,y), col=col)
  }
}
contour(x,y,z, add=TRUE) #add contours
```

In example.gradientPlots()

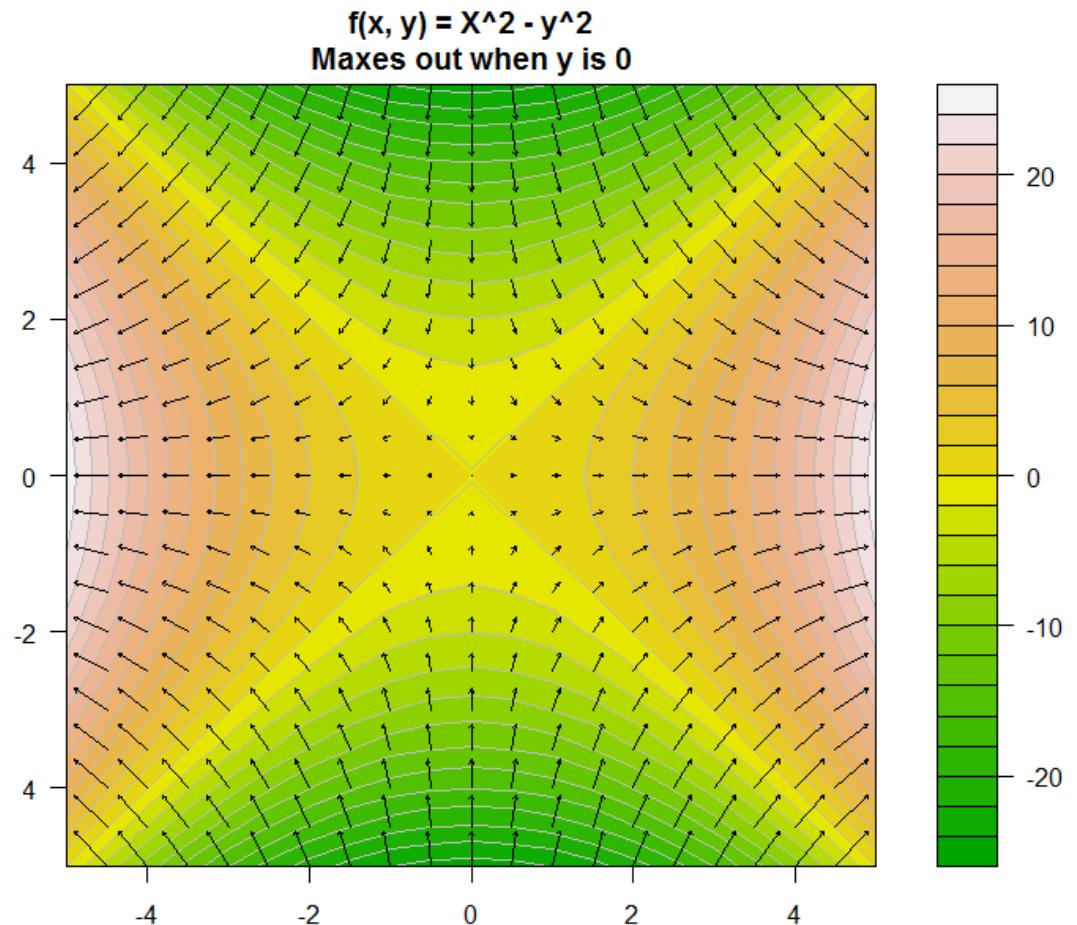
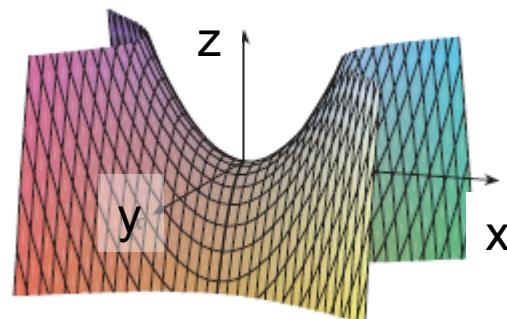


Ugly Unnormalized Gradient Plot



Gradient Vector Field for $f(x,y) = x^2 - y^2$

- Sampled gradient vector plot superimposed on a function heat map of $f(x, y) = x^2 - y^2$
- Each gradient vector is plotted starting at the point
- As expected, the gradient vectors point “uphill” and are perpendicular to the level curves.



Prettified Gradient Plot

(looks like a quiver!)

```
#prettified gradient vector plot using quiver()
```

```
#f <- expression( (3*x^2 + y) * exp(-x^2-y^2))
```

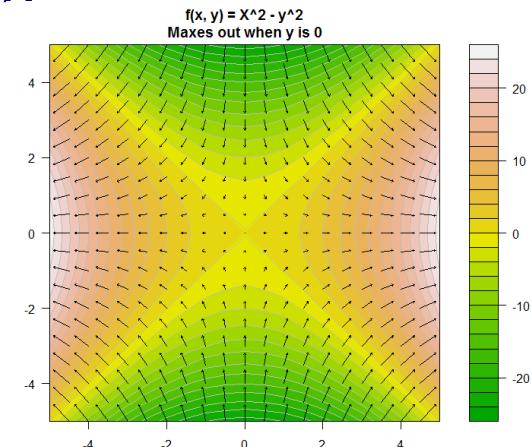
```
f <- expression( (x^2) - (y^2))
```

```
#f <-expression((x^2+x^2))
```

```
x <- y <- seq(-5, 5, by=0.5)
```

```
par(mar=c(3,3,3,3))
```

```
quiver2(f,x,y, color.palette=terrain.colors,  
main="f(x, y) = X^2 - y^2\nMaxes out when y is 0")
```

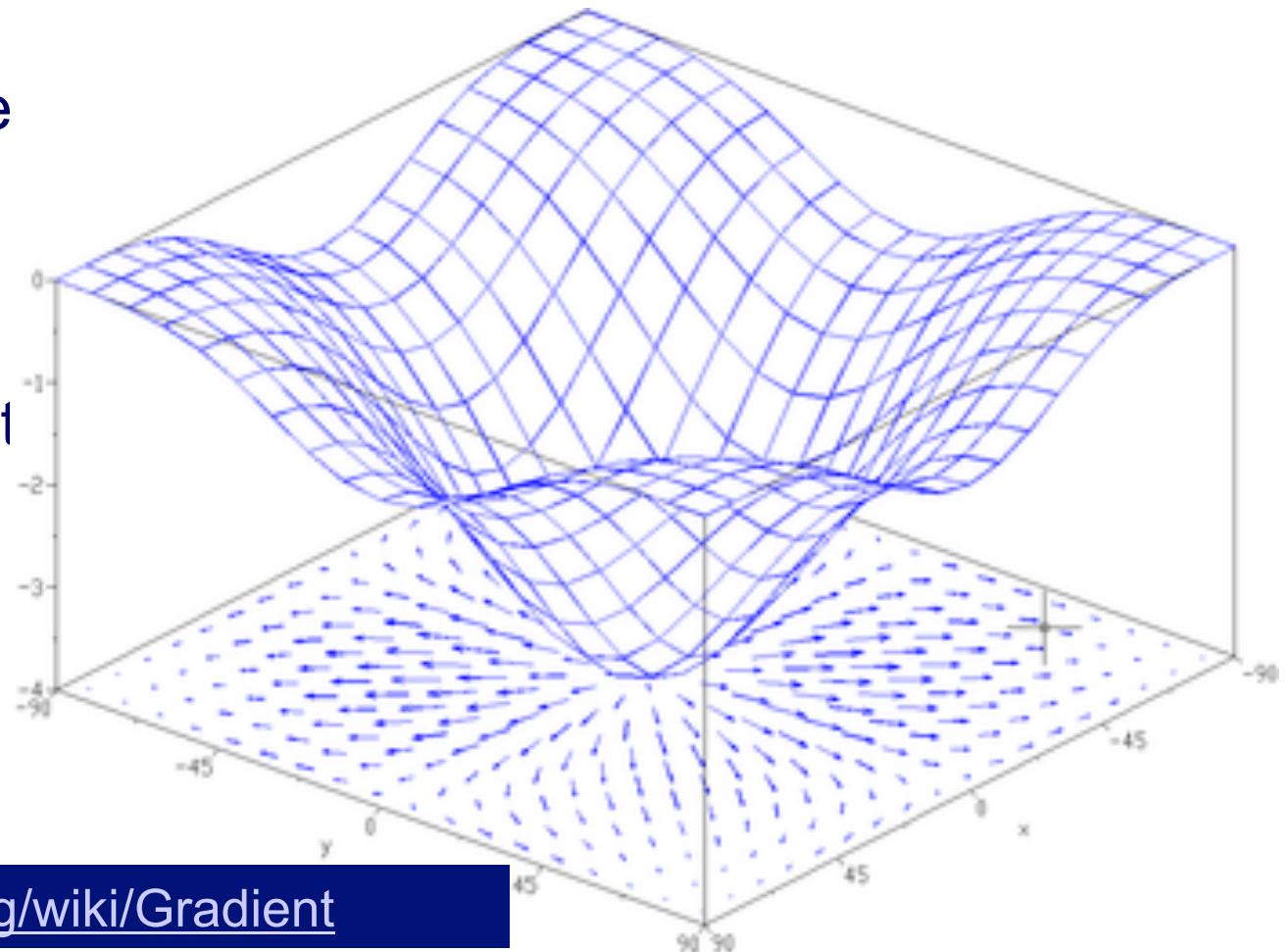


Gradient Vector at Extrema is $<0, 0, \dots>$

- The gradient is a fancy word for derivative, or the rate of change of a function.
- It's a vector (a direction to move) that points in the direction of greatest increase of a function is zero at a local maximum or local minimum (because there is no single direction of increase); the magnitude of the vector is zero.
Gradient at turning points = $<0, 0, 0\dots, 0>$
- The term gradient typically refers to the derivative of vector functions, or functions of more than one variable. Yes, you can say a line has a gradient (its slope), but using the term gradient for single-variable functions is unnecessarily confusing. Keep it simple.
- <http://betterexplained.com/articles/vector-calculus-understanding-the-gradient/>

Gradient Example

- The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a vector field on the bottom plane
- gradient vector plot using quiver()



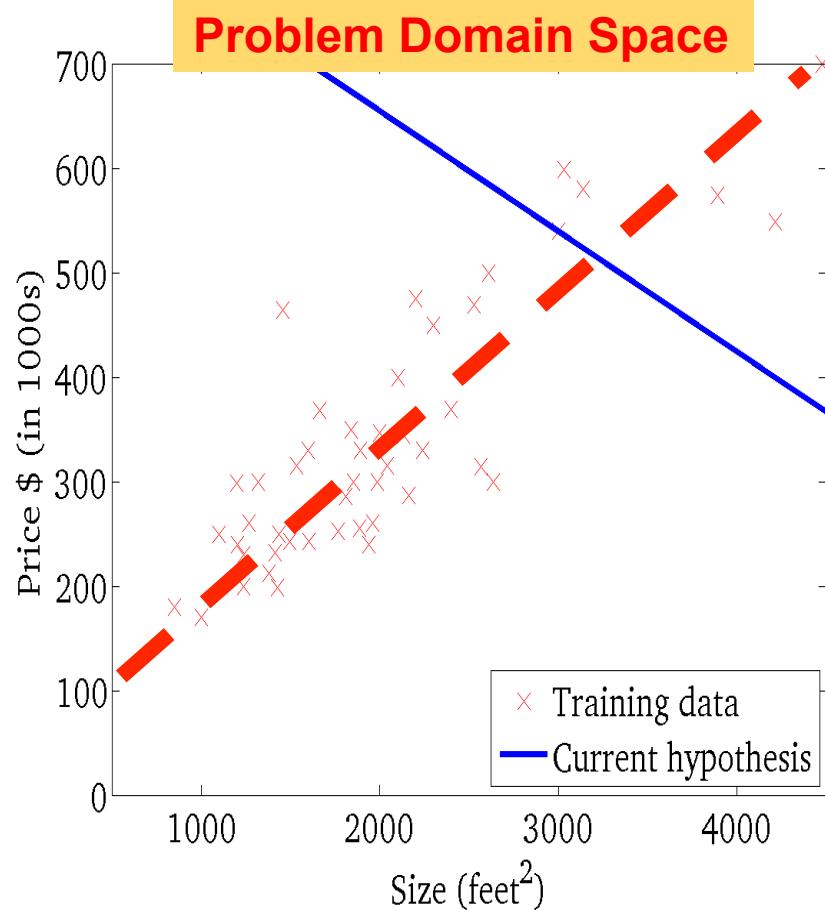
<http://en.wikipedia.org/wiki/Gradient>

Gradient Descent for LR Price~Size Iteration 0 vs Iteration Opt

Eqn Line $y = aX + b$

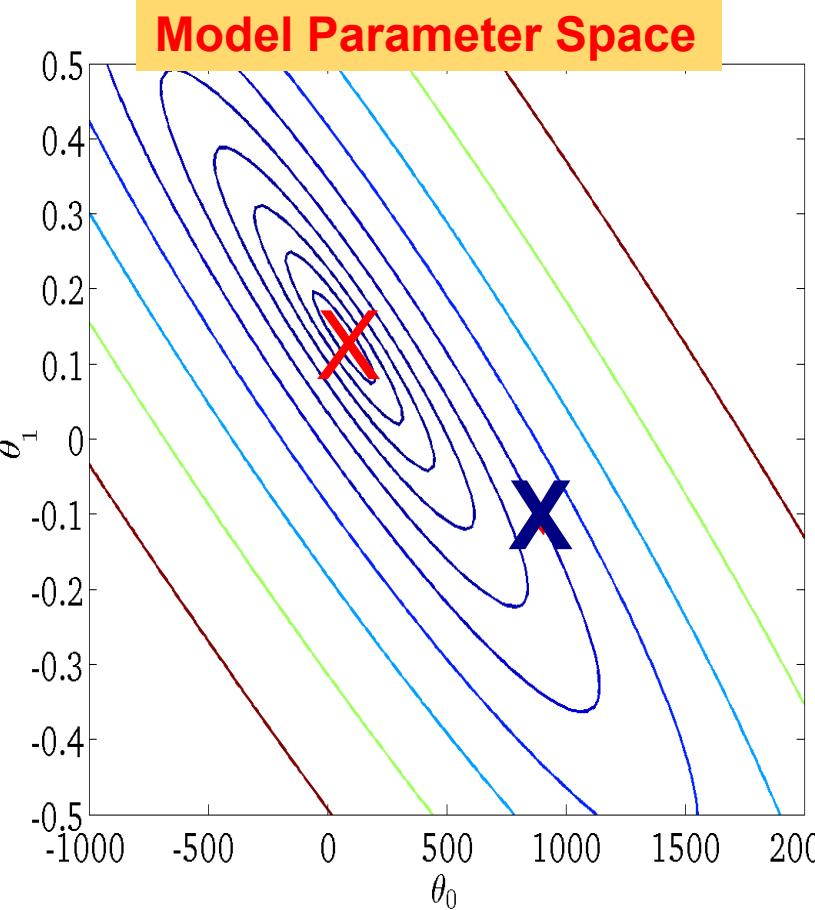
$$Y = w_1 X + W_0$$

(for fixed θ_0, θ_1 , this is a function of x)



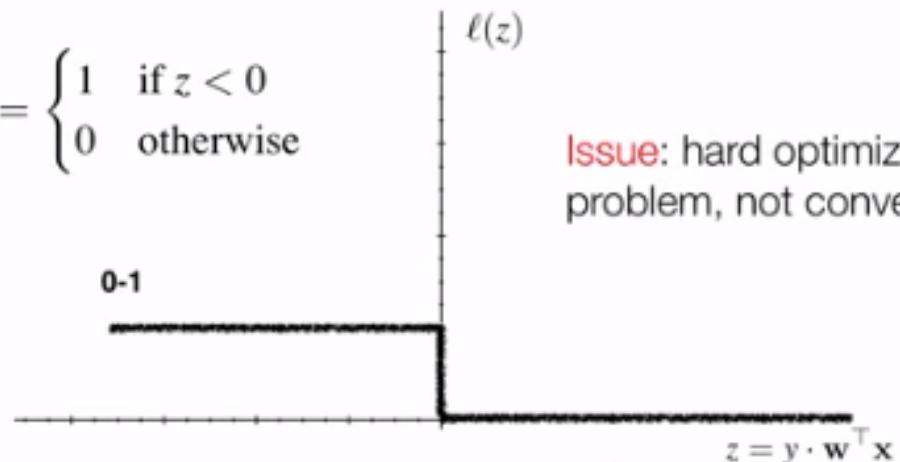
$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

(function of the parameters θ_0, θ_1)



0/1 Loss Minimization

$$\ell_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$



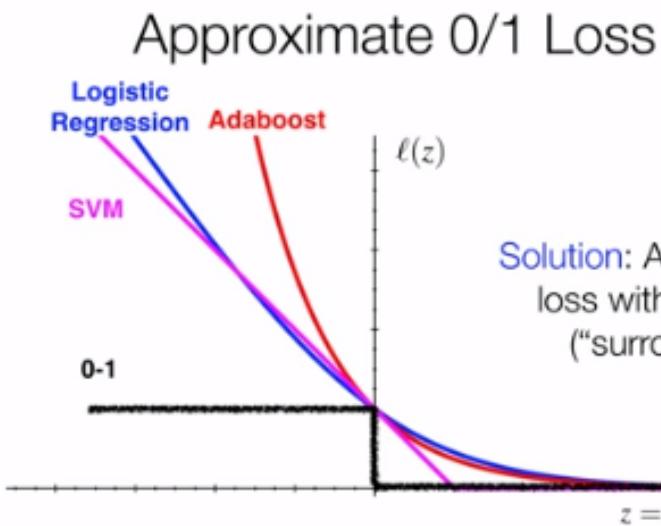
Issue: hard optimization problem, not convex!

0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

Let $y \in \{-1, 1\}$ and define $z = y \cdot w^T x$

z is positive if y and $w^T x$ have same sign, negative otherwise

Approximate 0/1 loss with a convex loss surrogate

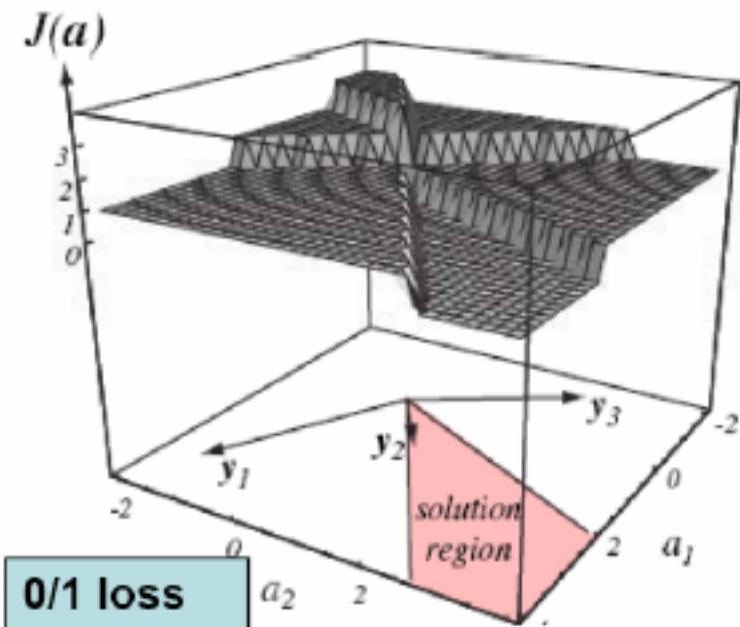


Solution: Approximate 0/1 loss with convex loss ("surrogate" loss)

SVM (hinge), Logistic regression (logistic), Adaboost (exponential)

Loss Functions

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N} \sum_{i=1}^N L(\text{sgn}(w \cdot x_i), y_i)$
 $L(y', y) = 0$ when $y' = y$, otherwise $L(y', y) = 1$
- Does not produce useful gradient since the surface of J is flat

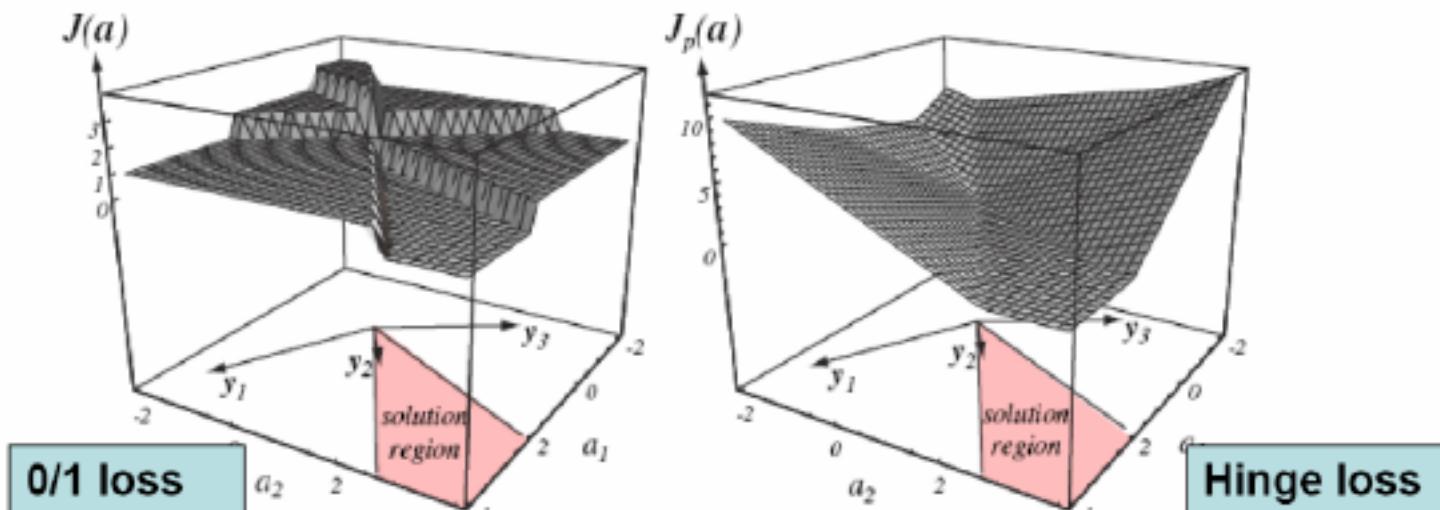


Loss Functions

- Instead we will consider the “**hinge loss**”:

$$J_p(w) = \frac{1}{N} \sum_{i=1}^N \max(0, -y_i w \cdot x_i)$$

- The term $\max(0, -y_i w \cdot x_i)$ is 0 when y_i is predicted correctly otherwise it is equal to the “confidence” in the mis-prediction
- Has a nice gradient leading to the solution region



Es from Xiaoli Fern (OSU)

Optimization Theory Bootcamp

- Optimization Theory background
- Find the optimal value
 - Newton-Rhapson
 - Gradient descent
- Convex optimization

Optimization → Convex Optimization

- Refresh optimisation theory before zeroing in on convex optimization
- In mathematics, computer science, economics, or management science, mathematical optimization is the selection of a best element (with regard to some criteria) from some set of available alternatives

Optimization

- In mathematics, computer science, economics, or management science,

BEST

- mathematical optimization is the selection of a best element (with regard to some criteria) from some set of available alternatives

Mathematical optimization

- ▶ Finding the minimizer of a function subject to constraints:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{s.t.} & f_i(x) \leq 0, \quad i = \{1, \dots, k\} \\ & h_j(x) = 0, \quad j = \{1, \dots, l\} \end{array} \quad \begin{array}{l} \text{an objective function, a loss function or} \\ \text{cost function (minimization)} \\ \text{Inequality constraints} \\ \text{Equality constraints} \end{array}$$

- ▶ Example: Stock market. "Minimize variance of return subject to getting at least \$50."

Such a formulation is called an optimization problem or a mathematical programming problem. Many real-world and theoretical problems may be modeled in this general framework.

The function f_0 is called, variously, an objective function, a loss function or cost function (minimization),[2] a utility function or fitness function (maximization), or, in certain fields, an energy function or energy functional.

- A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an optimal solution.

Why do we care?

Optimization is at the heart of many (most practical?) machine learning algorithms.

- ▶ Linear regression:

$$\underset{w}{\text{minimize}} \ \|Xw - y\|^2$$

- ▶ Classification (logistic regression or SVM):

$$\underset{w}{\text{minimize}} \sum_{i=1}^n \log(1 + \exp(-y_i x_i^T w))$$

$$\text{or } \|w\|^2 + C \sum_{i=1}^n \xi_i \text{ s.t. } \xi_i \geq 1 - y_i x_i^T w, \xi_i \geq 0.$$

But wait there is more...

- ▶ Maximum likelihood estimation:

$$\underset{\theta}{\text{maximize}} \sum_{i=1}^n \log p_{\theta}(x_i)$$

- ▶ Collaborative filtering:

$$\underset{w}{\text{minimize}} \sum_{i < j} \log (1 + \exp(w^T x_i - w^T x_j))$$

- ▶ k -means:

$$\underset{\mu_1, \dots, \mu_k}{\text{minimize}} J(\mu) = \sum_{j=1}^k \sum_{i \in C_j} \|x_i - \mu_j\|^2$$

- ▶ And more (graphical models, feature selection, active learning, control)



-
- Back to basics first

Maximum vs Minimum

$$f(x) = x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

$f'(x=x^*) = 0$ at maximum and minimum

These zero points are the roots!

Second derivative $f''(x) = 6x$

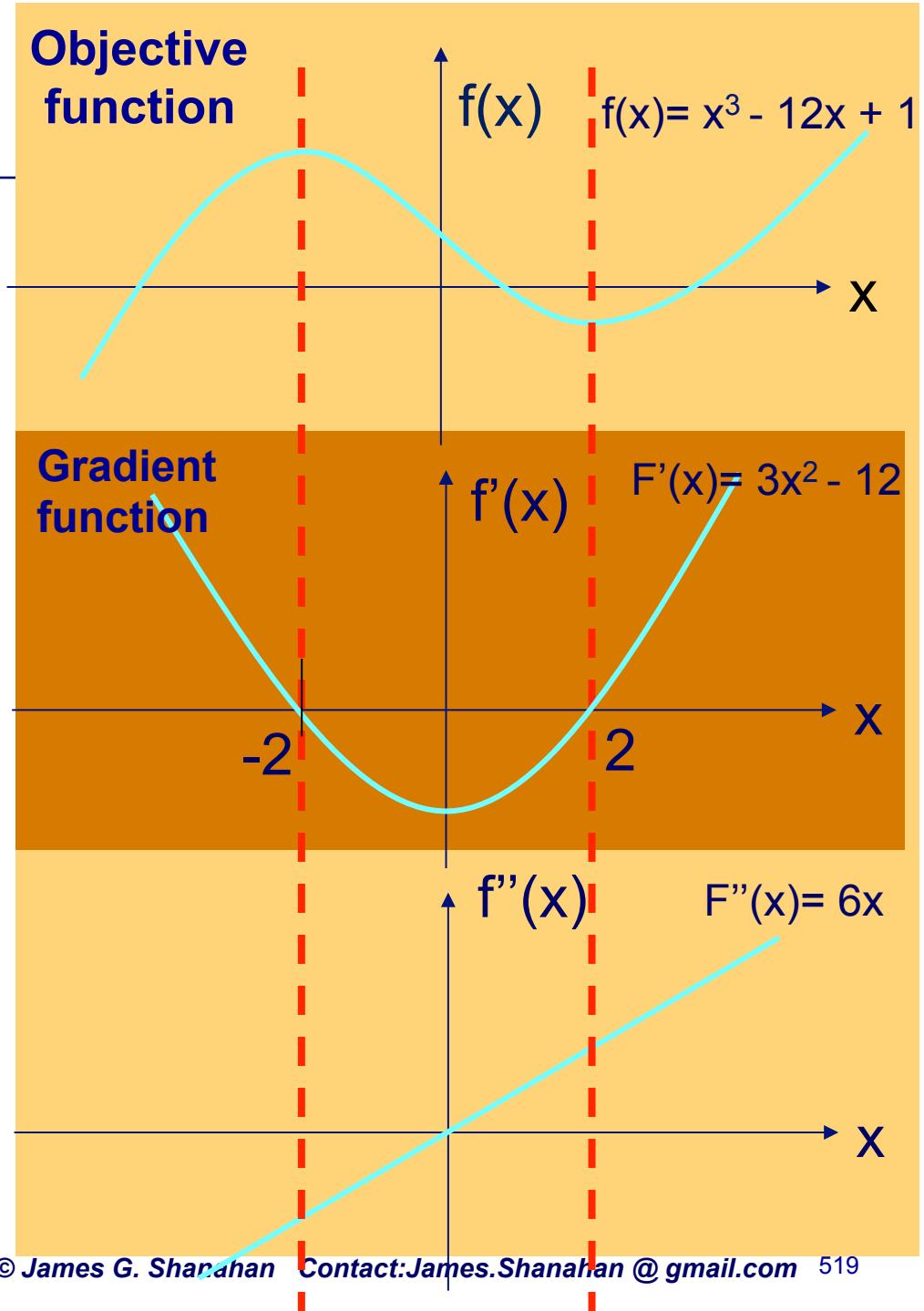
Gives the “rate of change of gradient”

SOC

If $f''(x=x^*) < 0$ then maximum

If $f''(x=x^*) > 0$ then minimum

If $f''(x=x^*) == 0$ then ???

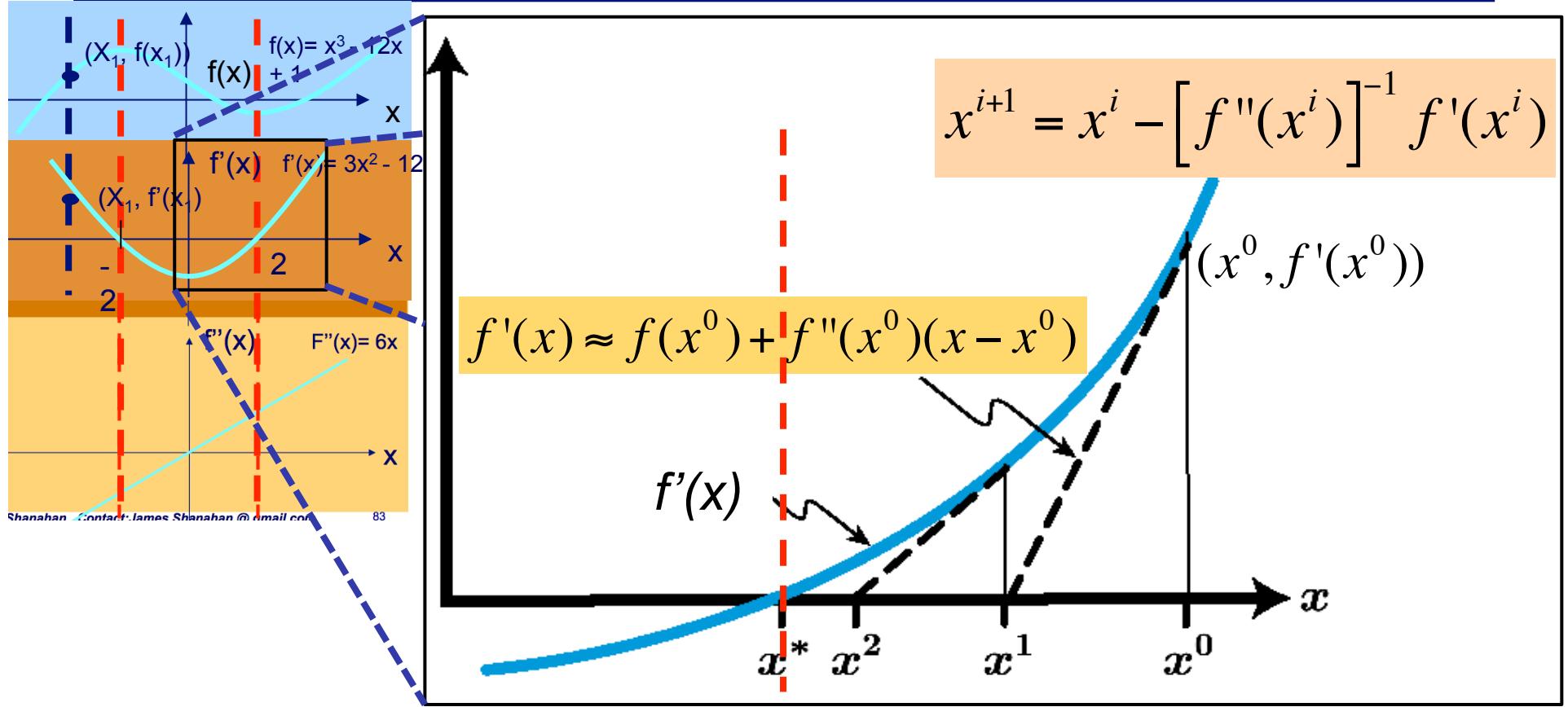


Finding the optimum

- **Finding the roots of the gradient function**
 - Closed form
 - Newton Raphson
 - Gradient descent
 - Using bisection method
- **Are you a root finder?**

Focus on Newton-Rhapson → Gradient Descent

Newton-Raphson Method – Root Finding of derivative/gradient function $f'(x)$



1. Initial guess: x^0 Letting $i=0$ $x^i=x^0$
2. Approximate $f(x)$ by tangent at $(x^i, f(x^i)) \# (x^0, f(x^0))$ for the first iteration
3. Find where $f_{\text{Tangent}, x_0}(x) = 0$, i.e., x^{i+1} ; better approx. of the root (x^*)
4. Repeat until convergence E.g., X does not change

Deriving Newton-Raphson Method (univariate case)

- Solving a nonlinear equation of the form $f(x)=0$
- Generate a sequence of iterate x^{n-1}, x^n, x^{n+1} which hopefully converges to the solution x^* (the root of $f(x)$)

$$f(x) \approx f(x^0) + f'(x^0)(x - x^0) \quad \forall x \text{ surrounding } x^0$$

$$f(x^{i+1}) \approx f(x^i) + f'(x^i)(x^{i+1} - x^i) \quad \forall x \text{ surrounding } x^i$$

$$0 = f(x^i) + f'(x^i)(x^{i+1} - x^i)$$

We desire a root (i.e., $f(x^{i+1}) = 0$)

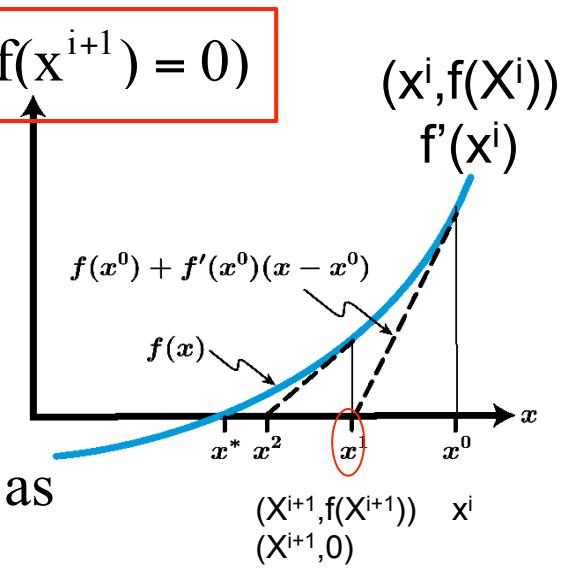
$$f'(x^i)(x^{i+1} - x^i) = -f(x^i)$$

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i$$

$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i) \right]^{-1} f(x^i)$$

Iteration function

sometimes written as



Gradient Descent (a simpler root finder) in 1D

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)}$$

Iteration function Newton-Raphson
 In 1-Dimension

$$x^{i+1} = x^i - [f''(x^i)]^{-1} f'(x^i)$$

- Calculating $f''(x)$, the Hessian H , and inverting it is complex so simpler algorithms have been developed such gradient descent

$$x^{i+1} = x^i - a^i f'(x^i)$$

Gradient Descent
Iterate until X does not change

- How large should I step in the positive gradient direction (gradient ascent)
 - or in the negative gradient direction (gradient descent)
 - Convergence criteria. E.g., decision vector X does not change that much

Gradient: a vector-valued function of partial derivatives

- In mathematics, the gradient is a generalization of the usual concept of derivative of a function in one dimension to a function in several dimensions.
- If $f(x_1, \dots, x_n)$ is a differentiable, scalar-valued function of standard Cartesian coordinates in Euclidean space, its gradient is the vector whose components are the n partial derivatives of f . It is thus a vector-valued function.
- Similarly to the usual derivative, the gradient represents the slope of the tangent of the graph of the function.
 - More precisely, the gradient points in the direction of the greatest rate of increase of the function and its magnitude is the slope of the graph in that direction.
 - The components of the gradient in coordinates are the coefficients of the variables in the equation of the tangent space to the graph.
 - This characterizing property of the gradient allows it to be defined independently of a choice of coordinate system, as a vector field whose components in a coordinate system will transform when going from one coordinate system to another.

<http://en.wikipedia.org/wiki/Gradient>

Gradient: Nabla is the symbol (∇): $\nabla=0$ FOC; points in the direction of greatest increase.

- The gradient at a specific point $x = x'$ is the vector whose elements are the respective partial derivatives evaluated at $X = x'$, so that
 - $\nabla f(X=x') = (df/dx_1, df/dx_2, \dots, df/dx_n)$
 - $\nabla(X=x')=0; \nabla f(X=x') = (0, 0, \dots, 0)$ at a candidate extremum (FOC)
- The significance of the gradient is that the (infinitesimal) change in x that maximizes the rate at which $f(x)$ increases is the change that is proportional to $\nabla f(x)$.

Gradient

Because the objective function $f(\mathbf{x})$ is assumed to be differentiable, it possesses a gradient, denoted by $\nabla f(\mathbf{x})$, at each point \mathbf{x} . In particular, the **gradient** at a specific point $\mathbf{x} = \mathbf{x}'$ is the *vector* whose elements are the respective *partial derivatives* evaluated at $\mathbf{x} = \mathbf{x}'$, so that

$$\nabla f(\mathbf{x}') = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right) \quad \text{at } \mathbf{x} = \mathbf{x}'.$$

The significance of the gradient is that the (infinitesimal) change in \mathbf{x} that *maximizes* the rate at which $f(\mathbf{x})$ increases is the change that is *proportional* to $\nabla f(\mathbf{x})$. To express this idea geometrically, the “direction” of the gradient $\nabla f(\mathbf{x}')$ is interpreted as the *direction* of the directed line segment (arrow) from the origin $(0, 0, \dots, 0)$ to the point $(\partial f / \partial x_1, \partial f / \partial x_2, \dots, \partial f / \partial x_n)$, where $\partial f / \partial x_j$ is evaluated at $x_j = x'_j$. Therefore, it may be said that the rate at which $f(\mathbf{x})$ increases is maximized if (infinitesimal) changes in \mathbf{x} are in the *direction* of the gradient $\nabla f(\mathbf{x})$. Because the objective is to find the feasible solution maximizing $f(\mathbf{x})$, it would seem expedient to attempt to move in the direction of the gradient as much as possible.

<http://en.wikipedia.org/wiki/Gradient>

Gradient descent is based on the observation that if the multivariable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point \mathbf{a} , then $F(\mathbf{x})$ decreases fastest if one goes from \mathbf{a} in the direction of the negative gradient of F at $\mathbf{a}, -\nabla F(\mathbf{a})$. It follows that, if

- $\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$

for γ small enough, then $F(\mathbf{a}) \geq F(\mathbf{b})$. With this observation in mind, one starts with a guess \mathbf{x}_0 for a local minimum of F , and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots,$$

so hopefully the sequence (\mathbf{x}_n) converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen via a line search that satisfies the Wolfe conditions), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

Hessian: Matrix of second partial derivatives of f

Jacobian versus Hessian

In mathematics, the **Hessian matrix** or **Hessian** is a square matrix of second-order **partial derivatives** of a scalar-valued **function**, or **scalar field**. It describes the local curvature of a function of many variables. The Hessian matrix was developed in the 19th century by the German mathematician [Ludwig Otto Hesse](#) and later named after him. Hesse originally used the term "functional determinants".

Specifically, suppose $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is a function taking as input a vector $\mathbf{x} \in \mathbb{R}^n$ and outputting a scalar $f(\mathbf{x}) \in \mathbb{R}$; if all second **partial derivatives** of f exist and are continuous over the domain of the function, then the Hessian matrix \mathbf{H} of f is a square $n \times n$ matrix, usually defined and arranged as follows:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

$$\nabla f (\mathbf{X}=\mathbf{x}') = (df/dx_1, df/dx_2, \dots, df/dx_n) = \mathbf{J}(\mathbf{X}=\mathbf{x}')$$

or, component-wise:

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

The **determinant** of the above matrix is also sometimes referred to as the **Hessian**.^[1]

The Hessian matrix can be considered related to the **Jacobian matrix** by $\mathbf{H}(f)(\mathbf{x}) = \mathbf{J}(\nabla f)(\mathbf{x})$.

Hessian matrices are used in large-scale **optimization** problems within **Newton**-type methods because they are the coefficient of the quadratic term of a local **Taylor expansion** of a function. That is,

$$y = f(\mathbf{x} + \Delta \mathbf{x}) \approx f(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta \mathbf{x} + \frac{1}{2}\Delta \mathbf{x}^T \mathbf{H}(\mathbf{x})\Delta \mathbf{x}$$

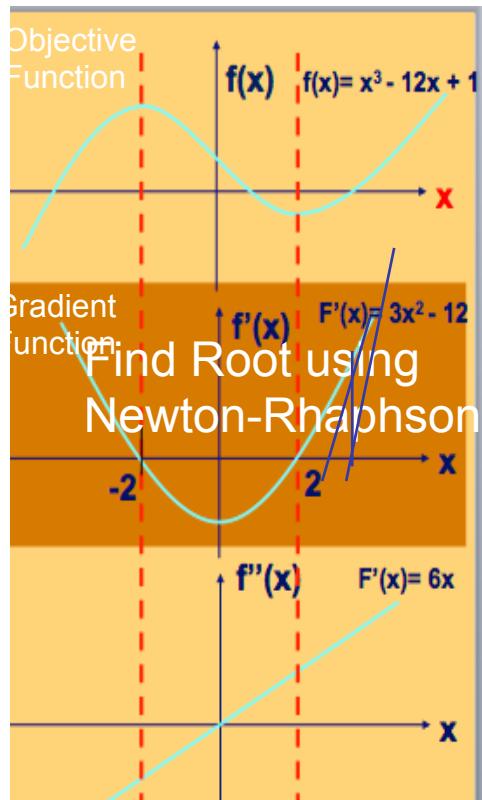
where **J** is the **Jacobian matrix**. The full Hessian matrix can be difficult to compute in practice; in such situations, **quasi-Newton** algorithms have been developed that use approximations to the Hessian. One of the most popular quasi-Newton algorithms is **BFGS**.^[2]

Gradient Descent (a simpler root finder) versus Newton-Raphson

GIVEN: Minimize $f(x)$

STEP 1: Find the zeros of the gradient function $f'(x)$

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$x^{i+1} = x^i - [f''(x^i)]^{-1} f'(x^i) \quad \text{Univariate case}$$

$$x^{i+1} = x^i - [H(x^i)]^{-1} J(x^i) \quad \text{Multivariate Case}$$

Newton-Raphson

Calculating $f''(x)$, the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \quad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \quad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)
- **Convergence criteria.** E.g., decision vector X does not change that much or error does not change

Determine the learning rate

- Fixed
 - Decay over time
-
- Linesearch (one-dimensional line search)
 - Lipschitz constant

Setting $\alpha = 1/L$ #Lipschitz constant

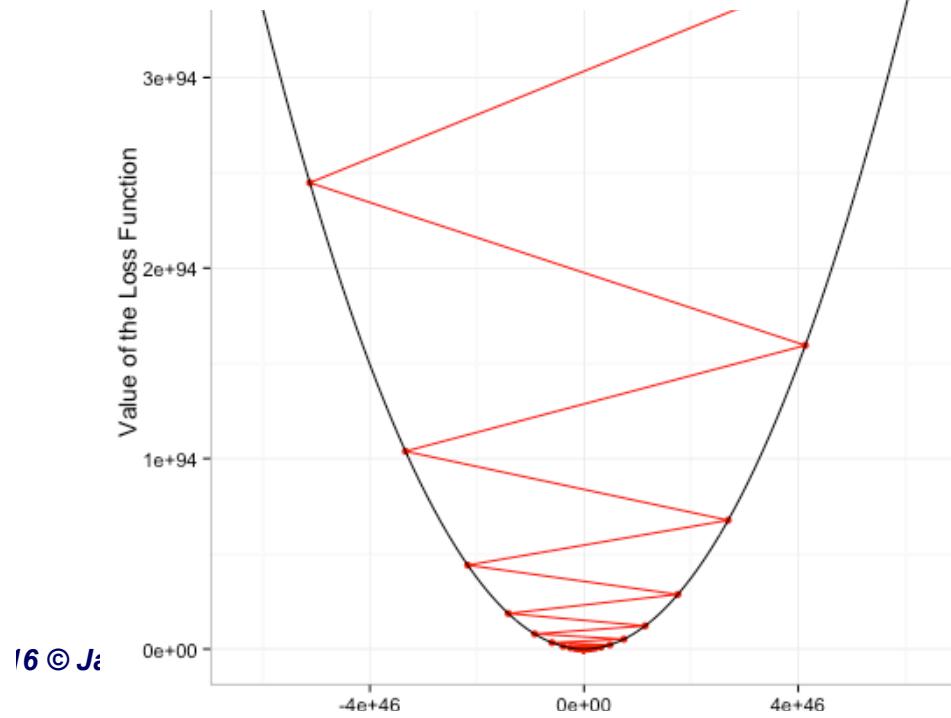
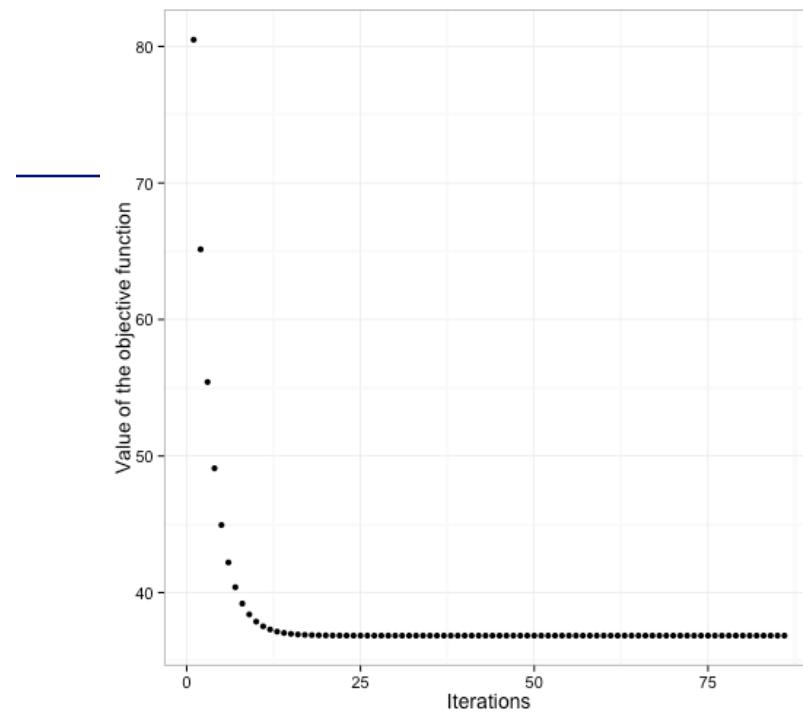
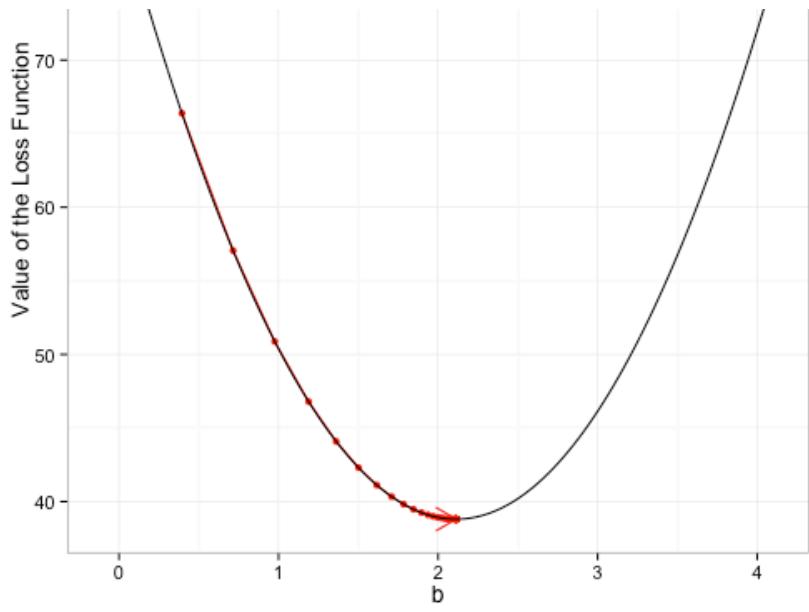
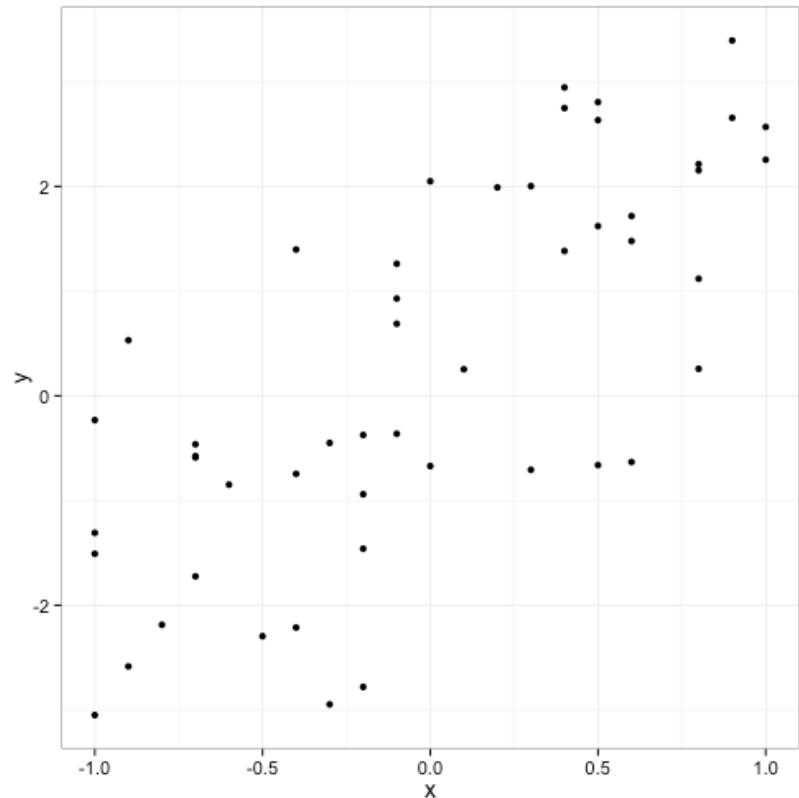
- In OLS, $f(\mathbf{b})$ is twice differentiable and its Hessian is $\mathbf{X}^t \mathbf{X}$, which does not depend on \mathbf{b} . Therefore, the smallest Lipschitz constant of ∇f is the largest eigenvalue of $\mathbf{X}^t \mathbf{X}$. Naturally, we want to take the biggest steps possible, so if we can compute the Lipschitz constant L we set $\alpha=1/L$.
- <http://www.statisticsviews.com/details/feature/5722691/Getting-to-the-Bottom-of-Regression-with-Gradient-Descent.html>

In OLS, $f(\mathbf{b})$ is twice differentiable and its Hessian is $\mathbf{X}^t \mathbf{X}$, which does not depend on \mathbf{b} . Therefore, the smallest Lipschitz constant of ∇f is the largest eigenvalue of $\mathbf{X}^t \mathbf{X}$. Naturally, we want to take the biggest steps possible, so if we can compute the Lipschitz constant L we set $\alpha = 1/L$.

Using the simple example we employed previously, we observe that when our choice of α is not small enough, the norm of the gradient will diverge towards infinity and the algorithm will not converge. (Note that the code for this example is provided below but in the interest of space, we do not include the output in this article.)

```
simple_ex2 <- gdescent(f, grad_f, x, y, alpha=0.05, liveupdates=TRUE)
```

The live updates in this example show the norm of the gradient in each iteration and we can see that the norm of the gradient diverges when α is not sufficiently small. The following two figures illustrate why this might occur. In the first figure, α is sufficiently small so each iteration in the algorithm results in a step towards the minimum, resulting in convergence of the algorithm.



Operational Issues: Quasi-Newton

Suppose that the objective f is a function of multiple arguments, $f(w_1, w_2, \dots, w_p)$. Let's bundle the parameters into a single vector, \vec{w} . Then the Newton update is

$$\vec{w}_{n+1} = \vec{w}_n - H^{-1}(w_n) \nabla f(\vec{w}_n) \quad (16)$$

Calculating gradient and Hessian not very time-consuming but calculating the inverse of H is!

where ∇f is the **gradient** of f , its vector of partial derivatives $[\partial f / \partial w_1, \partial f / \partial w_2, \dots, \partial f / \partial w_p]$, and H is the **Hessian** of f , its matrix of second partial derivatives, $H_{ij} = \partial^2 f / \partial w_i \partial w_j$.

Calculating H and ∇f isn't usually very time-consuming, but taking the inverse of H is, unless it happens to be a diagonal matrix. This leads to various **quasi-Newton** methods, which either approximate H by a diagonal matrix, or take a proper inverse of H only rarely (maybe just once), and then try to update an estimate of $H^{-1}(w_n)$ as w_n changes. (See section 8.3 in the textbook for more.)

In R, have a look at

```
?optim #method=BFGS
```

[\[http://www.stat.cmu.edu/~cshalizi/350/2008/lectures/29/lecture-29.pdf\]](http://www.stat.cmu.edu/~cshalizi/350/2008/lectures/29/lecture-29.pdf)

[Hand, Manilla, Smith, Data Mining, Section 8.3]

Characterizing Extrema in Multi-Dims

- **FOC**
 - Gradient function is zero (i.e., the objective function is at an extremum). In the mult
- **Is $F(x^*)$ a minimum?**
- **SOC**
 - Can be established by checking if the Hessian is positive definite (second partial derivative $f''(x^*)$ i.e., $\delta^2 F / \delta x_i \delta x_j$ evaluated at x^*)
- **A sufficient condition for an extreme point x^* (i.e., $F(x^*) = 0$) to be a minimum is to have a positive definite Hessian at x^* (i.e., has positive (nonzero) eigenvalues)**

Maximum vs Minimum

$$f(x) = x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

$f'(x=x^*) = 0$ at maximum and minimum

These zero points are the roots!

Second derivative $f''(x) = 6x$

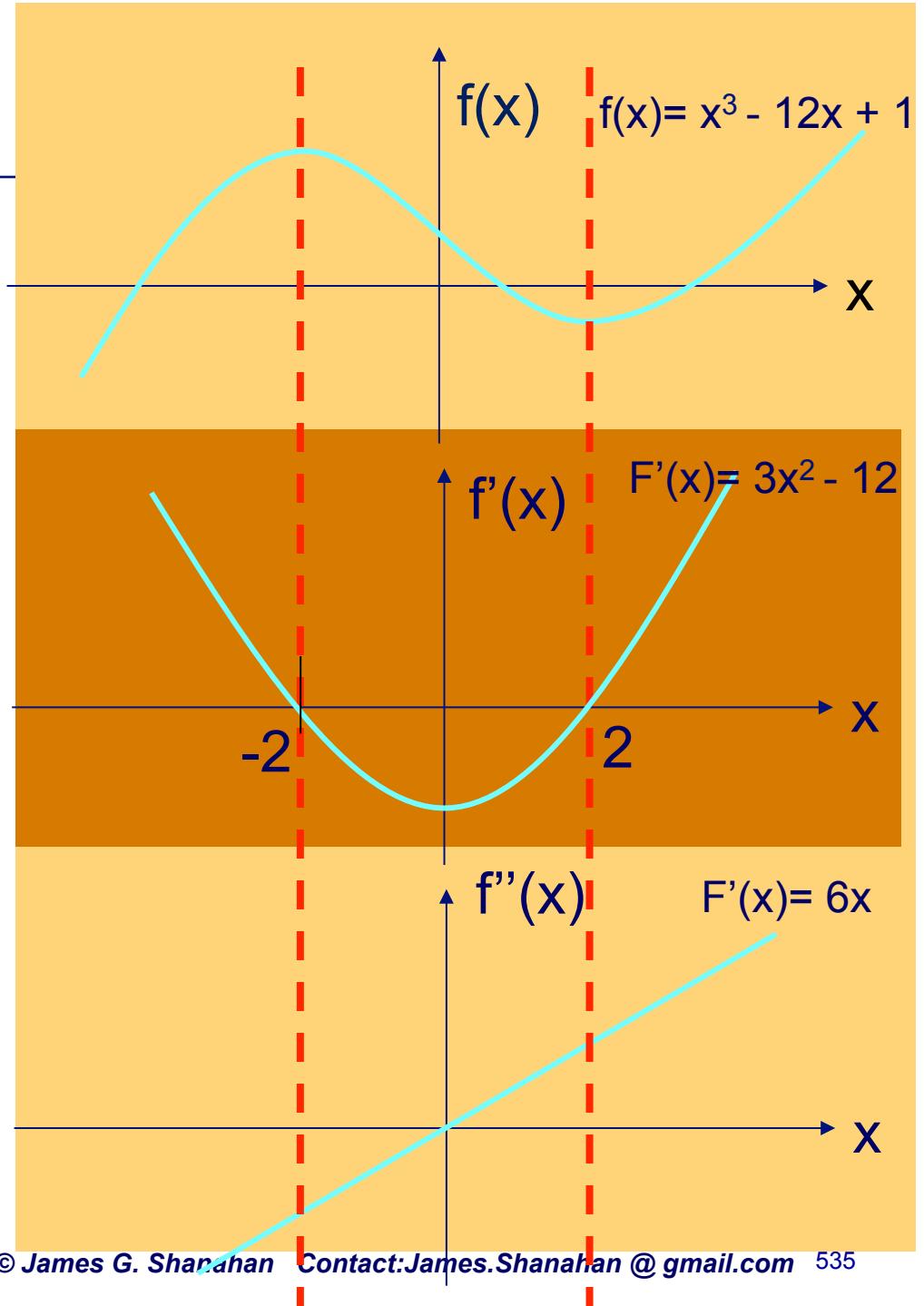
Gives the “rate of change of gradient”

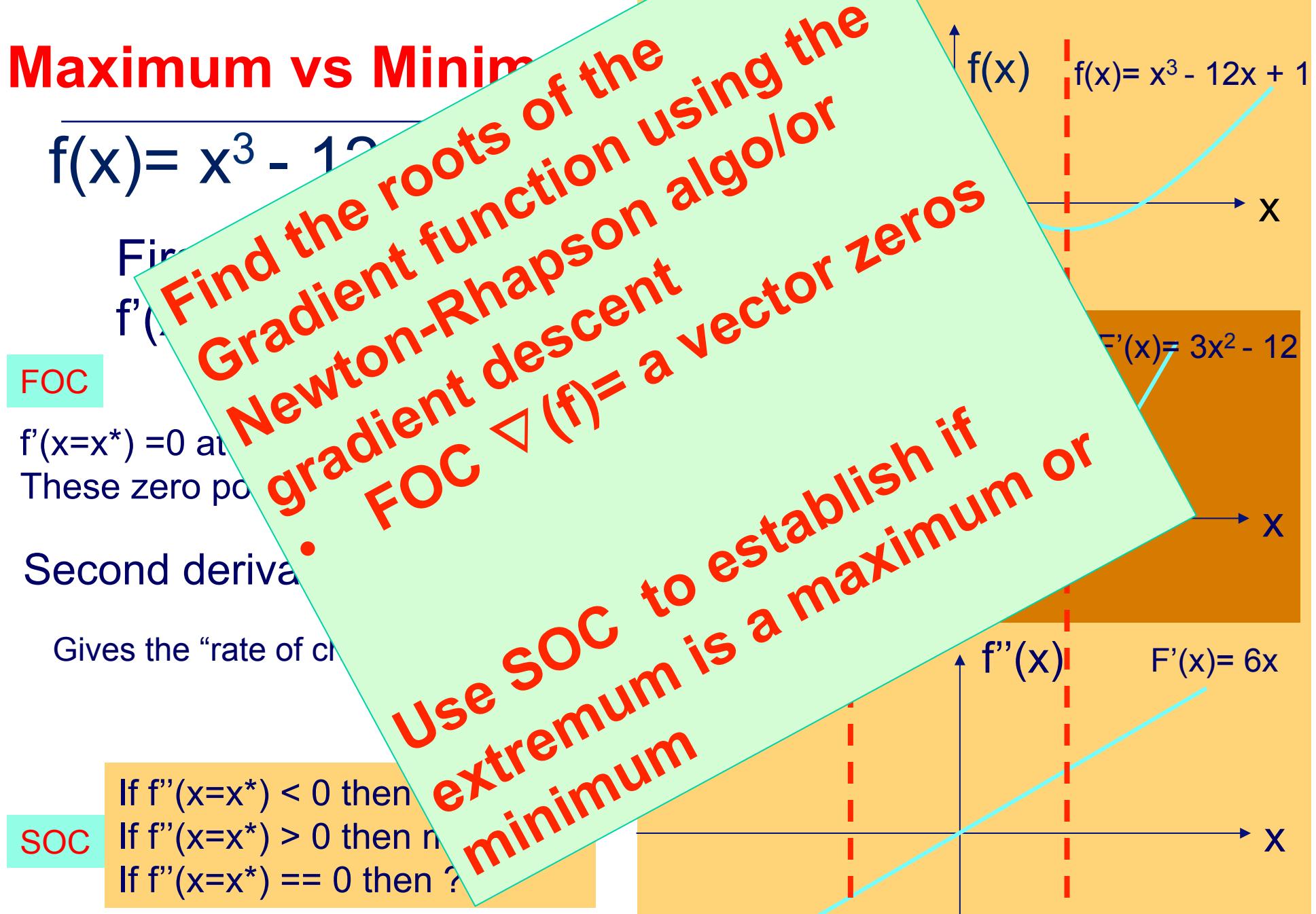
SOC

If $f''(x=x^*) < 0$ then maximum

If $f''(x=x^*) > 0$ then minimum

If $f''(x=x^*) == 0$ then ???





The unconstrained multivariate case

(m=0, n>1)

- A similar argument applies to obtain FOC and SOC in the multivariate case

FOC

- The gradient vector of partial first derivatives is equal to zero

soc

- The second condition implies that the second partial derivatives of f represented by the Hessian matrix is **positive definitive or positive semi definitive for a minimum**

$\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*}$ is the Hessian of $F(\mathbf{x})$ evaluated at \mathbf{x}^*

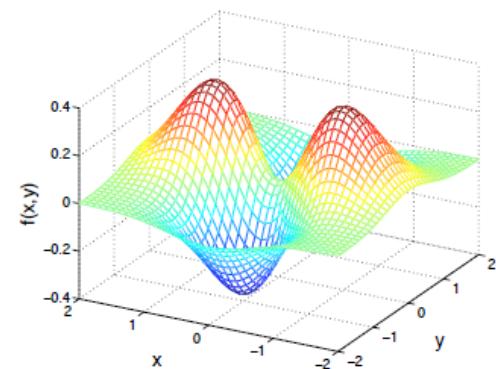
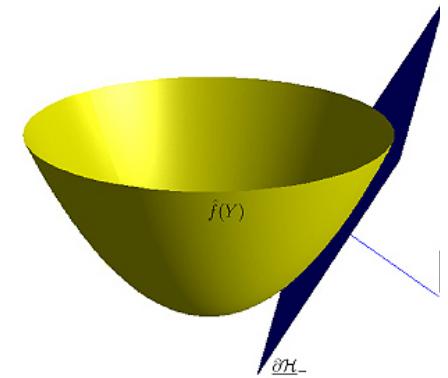
$\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*}$ is the gradient of $F(\mathbf{x})$ evaluated at \mathbf{x}^*

$$\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1} F(\mathbf{x}) \frac{\partial}{\partial x_2} F(\mathbf{x}) \dots \frac{\partial}{\partial x_n} F(\mathbf{x}) \right]^T$$

$$H = \nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_2 \partial x_n} F(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_n^2} F(\mathbf{x}) \end{bmatrix}$$

General Approach to Finding Fxtrema

- **Well-behaved version spaces**
 - Convex or concave function (\pm definiteness)
 - Algorithms seek a local extrema knowing that it will be global
 - If $f()$ is a concave function then local maximum is a global maximum
 - If $f()$ is a convex function then local minimum is a global minimum
 - Newton-Raphson, Gradient Descent, Conjugate Gradient Descent
- **Otherwise**
 - We resort to local approximations
 - Hill-Climbing
 - Simulated annealing
 - Commonly used in Neural Networks

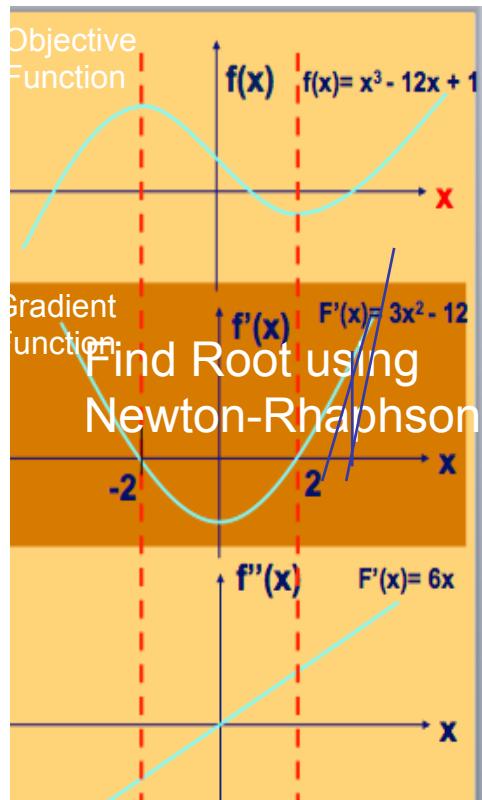


Gradient Descent (a simpler root finder) versus Newton-Raphson

GIVEN: Minimize $f(x)$

STEP 1: Find the zeros of the gradient function $f'(x)$

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$x^{i+1} = x^i - [f''(x^i)]^{-1} f'(x^i) \quad \text{Univariate case}$$

$$x^{i+1} = x^i - [H(x^i)]^{-1} J(x^i) \quad \text{Multivariate Case}$$

Newton-Raphson

Calculating $f''(x)$, the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \quad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \quad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)
- **Convergence criteria.** E.g., decision vector X does not change that much or error does not change

-
- **Next let's look at the world of convex optimization and how it relates to machine learning**

L6: Supervised Machine Learning Outline (Part 1)

- Supervised machine learning (parametric vs non) (5)
- Core supporting concepts: Matrices and Optimization Theory
 - Matrices
 - Applications of matrices at scale (5)
 - Distributed Matrix by vector multiplication (15)
 - Distributed matrix by matrix multiplication (5)
 - Optimization Theory
 - Gradient descent (15)
 - Convex optimization (5) (50)
- Linear Regression via closed form
 - Linear Regression Introduction (10)
 - Distributed Closed form Linear Regression (3)
 - Notebook for Distributed CF Linear Regression (5) [SCREENFLOW?]
 - Complexity, Scaling out, and Case study (5)
 - BLT question: what is the minimum number of MRJobs? (Multiple choice) (10)
- Linear Regression via Gradient Descent
 - Distributed Linear regression via gradient descent (15)
 - Notebook for Distributed SGD Linear Regression (5) [SCREENFLOW?]

V4

- Convexity

-
- **In this section we review convex optimization**
 - **convex optimization turns out to one of the most important pillars of math underlying machine learning**

Convex minimization: Just need FOC

- Convex minimization, a subfield of optimization, studies the problem of minimizing convex functions over convex sets.
- The convexity property can make optimization in some sense "easier" than the general case - for example, any local minimum must be a global minimum.
- We do NOT need to look at the SOC to determine min/max

Given a real vector space X together with a convex, real-valued function

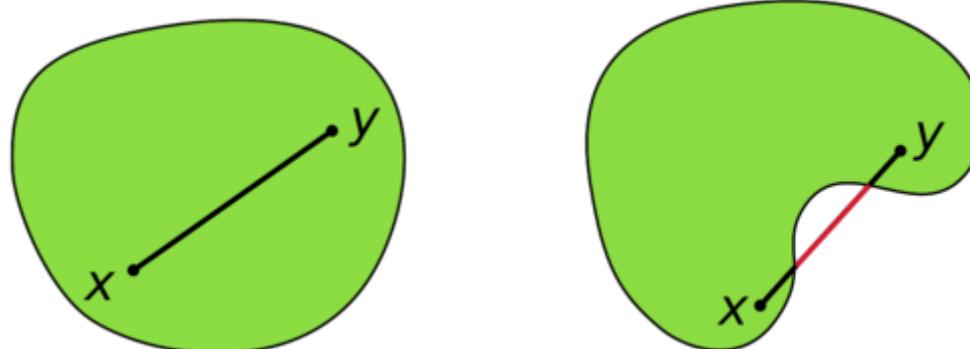
$$f : \mathcal{X} \rightarrow \mathbb{R}$$

defined on a convex subset \mathcal{X} of X , the problem is to find any point x^* in \mathcal{X} for which the number $f(x)$ is smallest, i.e., a point x^* such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{X}.$$

Convex minimization over Convex Sets

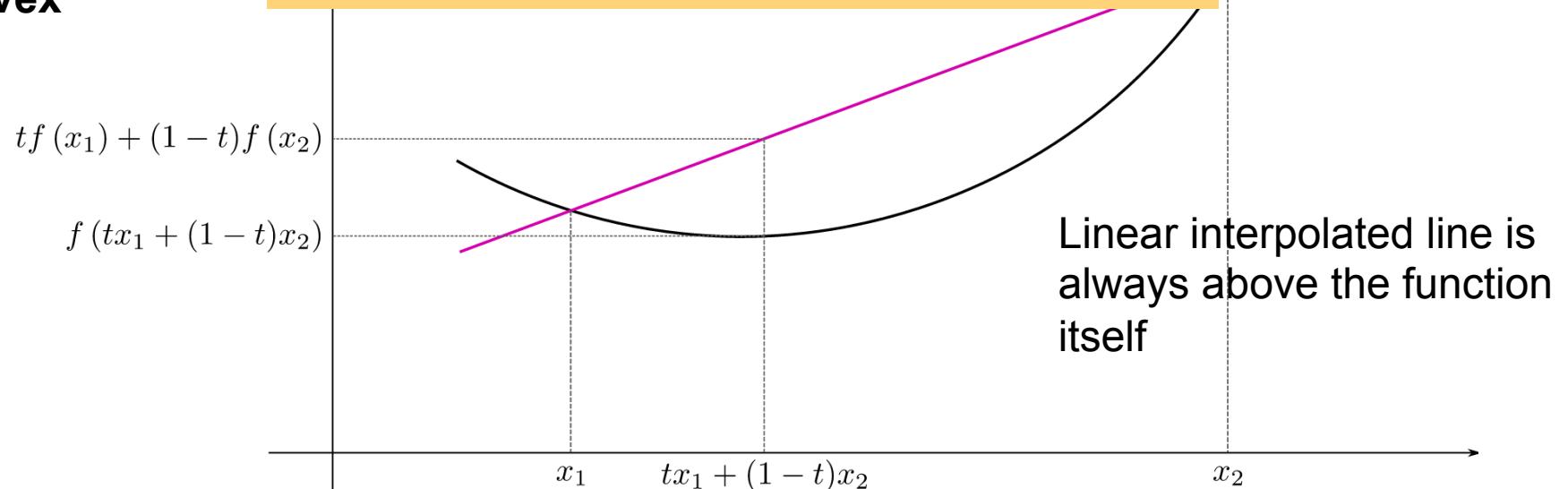
- In Euclidean space, an object is **convex** if for every pair of points within the object, every point on the straight line segment that joins them is also within the object.
- *Intuitively, this means that the set is connected (so that you can pass between any two points without leaving the set) and has no dents in its perimeter.*
- For example, a solid cube is convex, but anything that is hollow or has a dent in it, for example, a crescent shape, is not convex.



Minimization: Convex Functions in 1D

Objective functions for convex optimization need to be convex

Linear interpolation of values of the function evaluated at two points X_1 and X_2 should be greater than the function evaluated at the sum of the values $X_1 + X_2$ across the universe of interest



Let X be a convex set in a real vector space and let $f: X \rightarrow \mathbf{R}$ be a function.

- f is called **convex** if:

$$\forall x_1, x_2 \in X, \forall t \in [0, 1] : f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2).$$

- f is called **strictly convex** if:

$$\forall x_1 \neq x_2 \in X, \forall t \in (0, 1) : f(tx_1 + (1 - t)x_2) < tf(x_1) + (1 - t)f(x_2).$$

Convexity: Zero-order condition

A real-valued function is **convex** if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all $x, y \in \mathbb{R}^n$ and all $0 \leq \theta \leq 1$.

- Function is *below the chord* from x to y .
- Show that all local minima are global minima.

Exercise: Zero- to First-Order Condition

Show that zero-order condition,

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

implies first-order condition,

$$f(x) \geq f(y) + \nabla f(y)^T (x - y).$$

① Use: $\theta x + (1 - \theta)y = y + \theta(x - y)$.

② Use:

$$\nabla f(y)^T d = \lim_{\theta \rightarrow 0} \frac{f(y + \theta d) - f(y)}{\theta}$$

First order condition of convexity

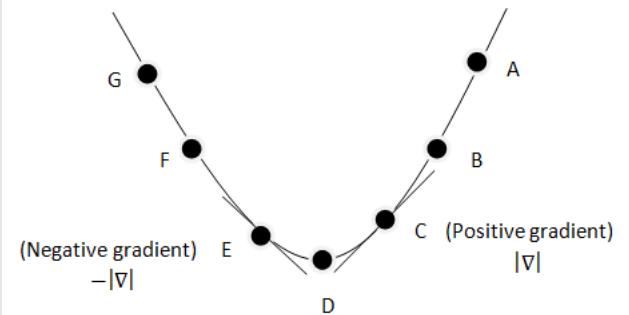
Convexity: First order condition

Suppose f is differentiable; i.e., gradient is defined over the domain of f . Then f is convex iff:

A real-valued *differentiable* function is **convex** iff

$$f(y) \geq f(x) + \nabla f(y)^T \cdot (y - x)$$

for all $x, y \in \mathbb{R}^n$.



- The function is globally above the tangent at y .

<http://www.cs.ubc.ca/~schmidtm/MLSS/convex.pdf>

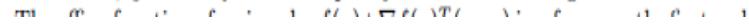
-
- This is called First order condition of convexity

3.1.3 First-order conditions

Suppose f is differentiable (i.e., its gradient ∇f exists at each point in $\text{dom } f$, which is open). Then f is convex if and only if $\text{dom } f$ is convex and

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) \quad (3.2)$$

holds for all $x, y \in \text{dom } f$. This inequality is illustrated in figure 3.2.



Convexity: First-order condition

A real-valued *differentiable* function is convex iff

$$f(x) \geq f(y) + \nabla f(y)^T(x - y),$$

for all $x, y \in \mathbb{R}^n$.

- The function is globally *above* the tangent at y .

https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

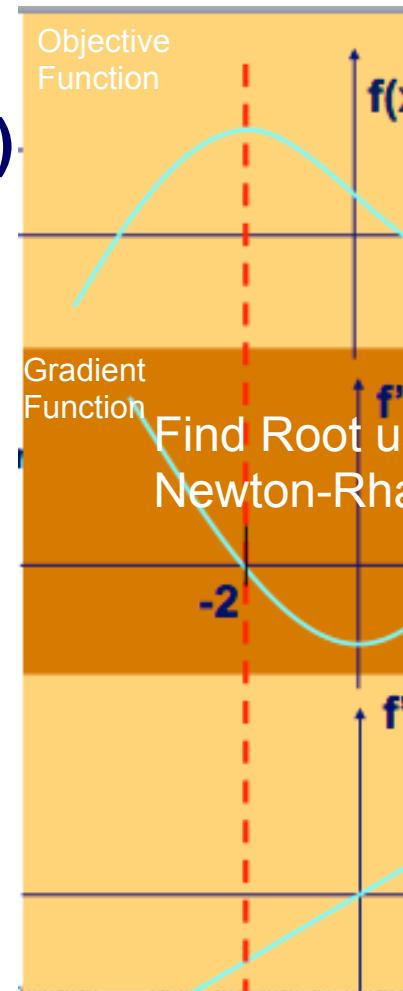
<http://www.cs.ubc.ca/~schmidtm/MLSS/convex.pdf>

rigorous proof

- <http://mathgotchas.blogspot.com/2011/10/proof-for-first-order-condition-of.html>

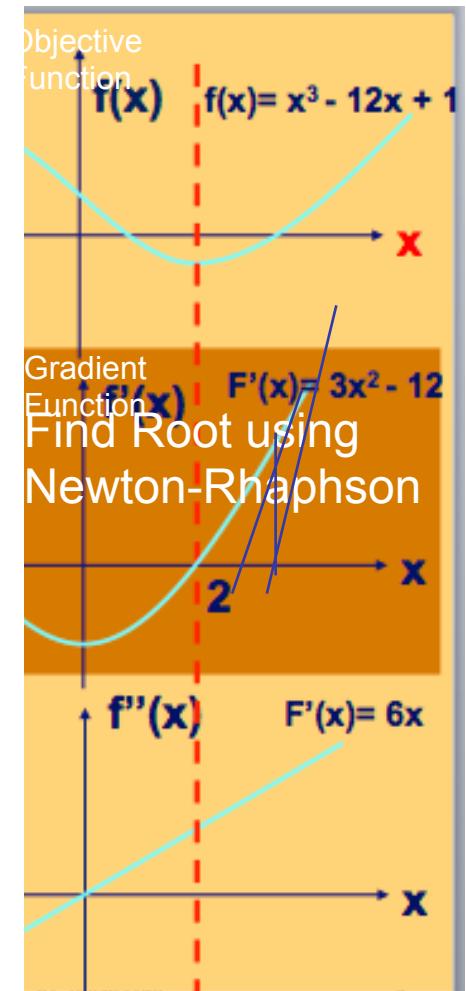
Convex opt: Maximization Problems

- Max, concave, negative $f''(x)$
- Establish where $f'(x)=0$
- And we have found the max



Convex opt: Minimization Problems

- Min, convex, $f''(x) < 0$
- Establish where $f'(x)=0$
- And we have found the min



Prove Convexity thru SOC

- In the case of multidimensional functions:
 - For minimization we look at Hessian matrix;
 - It needs to be positive (semi) definite

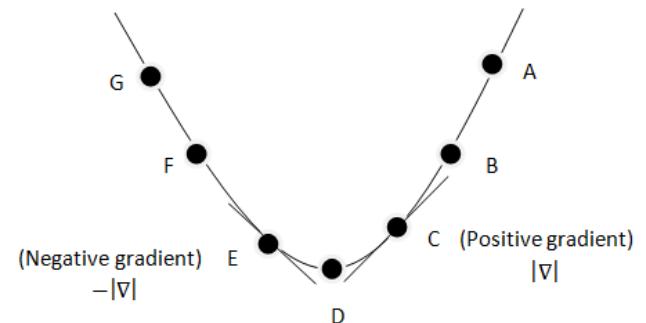
Convex Optimization Problems

More formally

Definition

An optimization problem is *convex* if its objective is a convex function, the inequality constraints f_j are convex, and the equality constraints h_j are affine

$$\begin{aligned} & \underset{x}{\text{minimize}} \quad f_0(x) \quad (\text{Convex function}) \\ & \text{s.t. } f_i(x) \leq 0 \quad (\text{Convex sets}) \\ & \quad h_j(x) = 0 \quad (\text{Affine}) \end{aligned}$$



Theorem

$\nabla f(x) = 0$ if and only if x is a global minimizer of $f(x)$.

Proof.

- $\nabla f(x) = 0$. We have

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) = f(x).$$

- $\nabla f(x) \neq 0$. There is a direction of descent.

First order condition of convexity
Around a minimum, everywhere
 $f(y) \geq f(x) + \nabla f(x)^T (y - x)$
For it to be minimum if $\nabla f(x)$ is zero otherwise we violate the condition

A not rigorous proof

If y is close to x , then by Taylor series

$$f(y) \approx f(x) + \nabla f(x) \cdot (y - x) + \frac{1}{2} \nabla^2 f(x) \cdot (y - x)^2 \geq f(x) + \nabla f(x) \cdot (y - x) = f(x)$$

$$f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x)$$

$$\nabla^2 f(x) \cdot (y - x)^2 \geq 0 \quad \text{because} \quad \nabla^2 f(x) \geq 0 \\ (y - x)^2 \geq 0$$

**First order condition
of convexity**
Around a minimum, everywhere
 $f(y) \geq f(x) + \nabla f(x)^T (y - x)$
For it to be minimum if $\nabla f(x)$ is zero otherwise we violate the condition

• Loss Functions

Loss Functions Succinct Expose

- [http://www.ics.uci.edu/~dramanan/teaching/
ics273a_winter08/lectures/lecture14.pdf](http://www.ics.uci.edu/~dramanan/teaching/ics273a_winter08/lectures/lecture14.pdf)
- **Local copy on Jimi computer at Lecturer 5 MIDS
Loss-Functions-Explained.pdf**

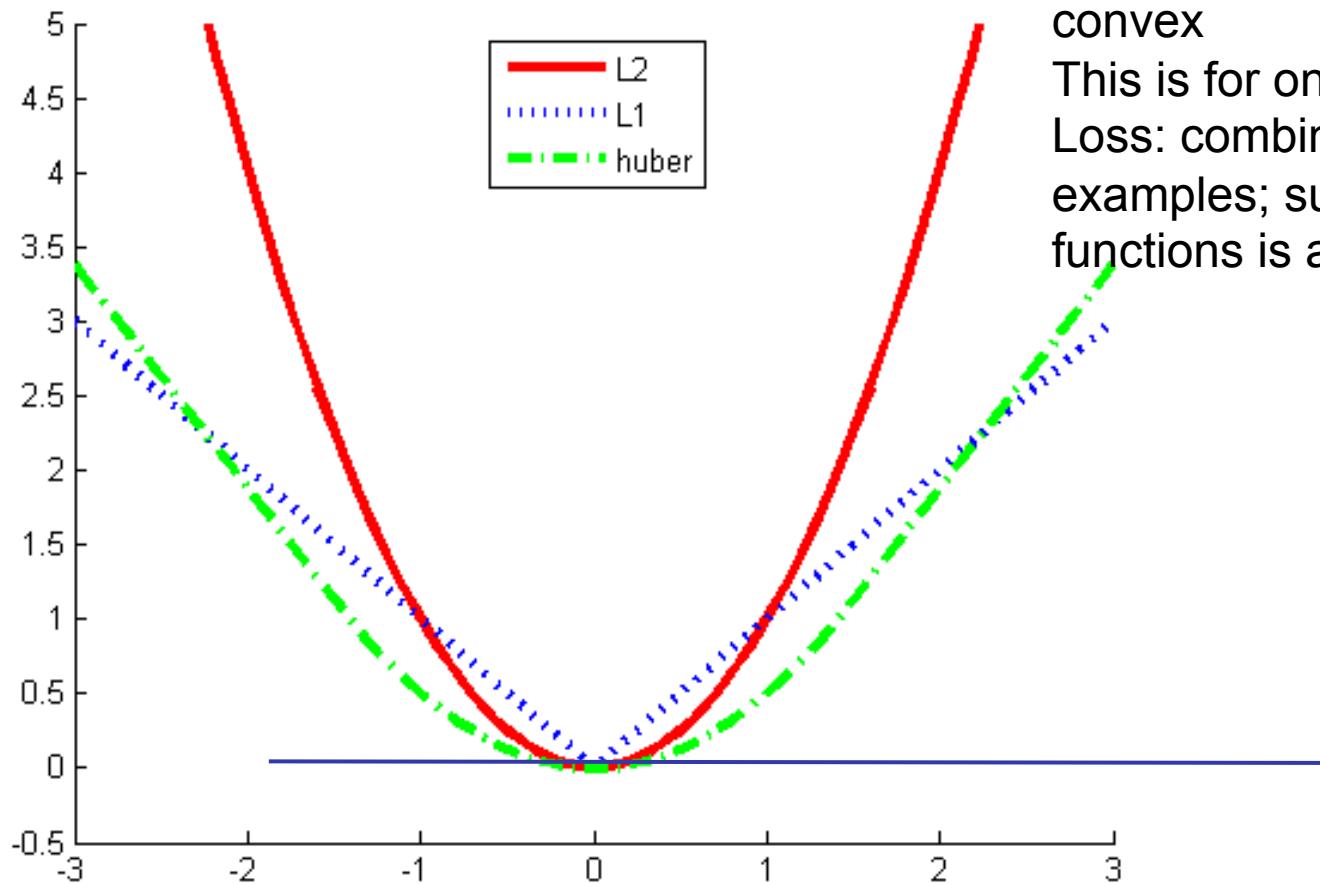
Convex optimization in ML

- **Many methods in machine learning are based on finding parameters that minimize some objective function.**
- **Very often, the objective function is a weighted sum of two terms:**
 - a cost function and regularization term.
 - In statistics terms the (log-)likelihood and (log-)prior.
 - If both of these components are convex, then their sum is also convex.
 - Loss functions are summed over examples so the sum of a convex functions is a convex function

Machine Learning Objective Function: regression loss function

Model: $y = \theta^T x$

Goal: $\theta^* = \min_{\theta} \sum_{i=1}^m (\theta^T x_i - y_i)^2$

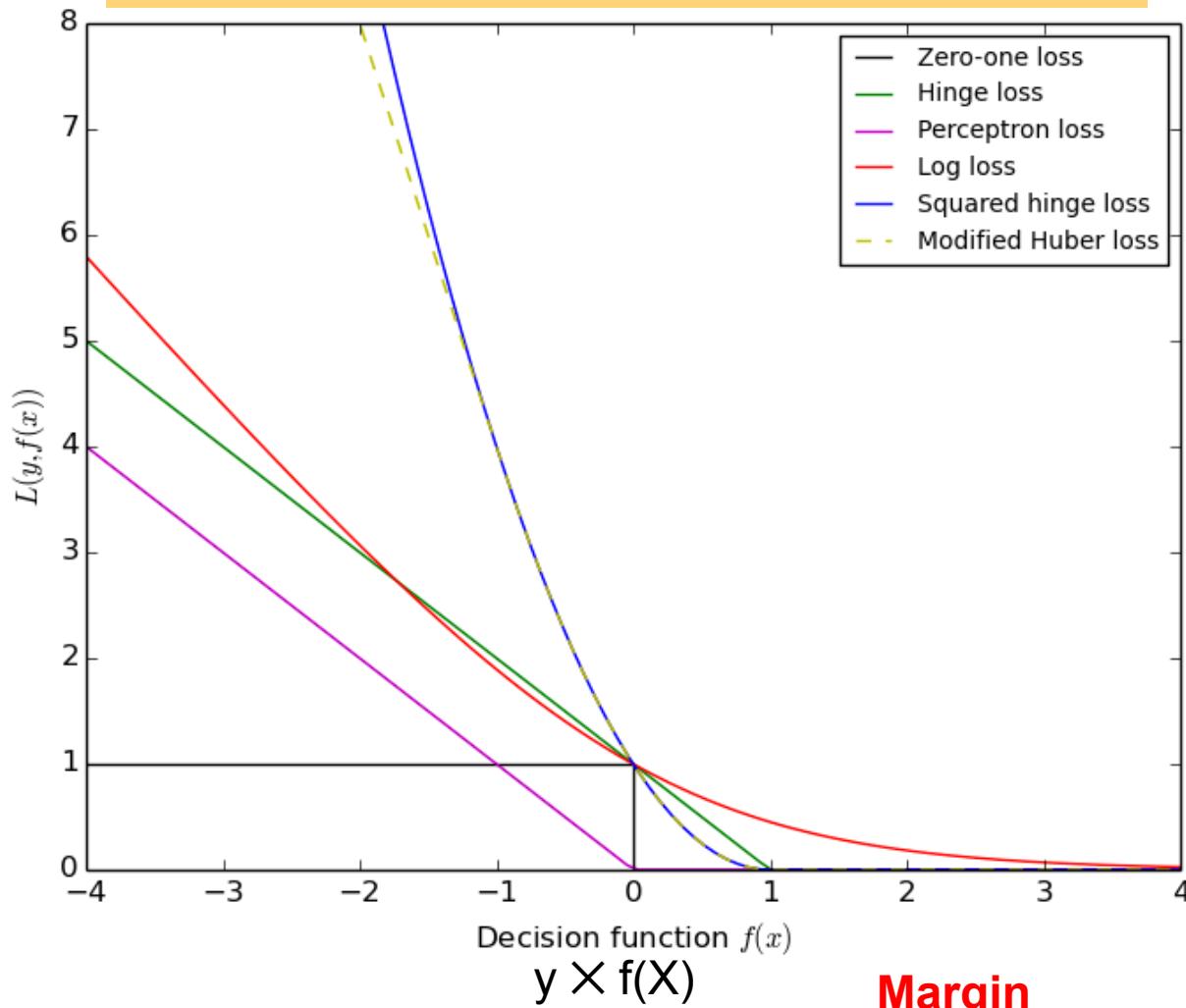


Regression loss functions: all are convex

This is for one example in 1Dim;
Loss: combine over all training
examples; sum of such convex
functions is also convex

Machine Learning Objective Function: classification

Classification loss functions



Loss functions; a unifying view

- Loss function consists of:
 - loss term ($L(m_i(w))$, expressed in terms of the margin of each training example) and
 - regularization term ($R(w)$ expressed as a function of the model complexity)

$$J(w) = \sum_i \text{Loss term } L(m_i(w)) + \text{regularization term } \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

Empirical Risk Minimization

- Provides a criterion to decide on h :

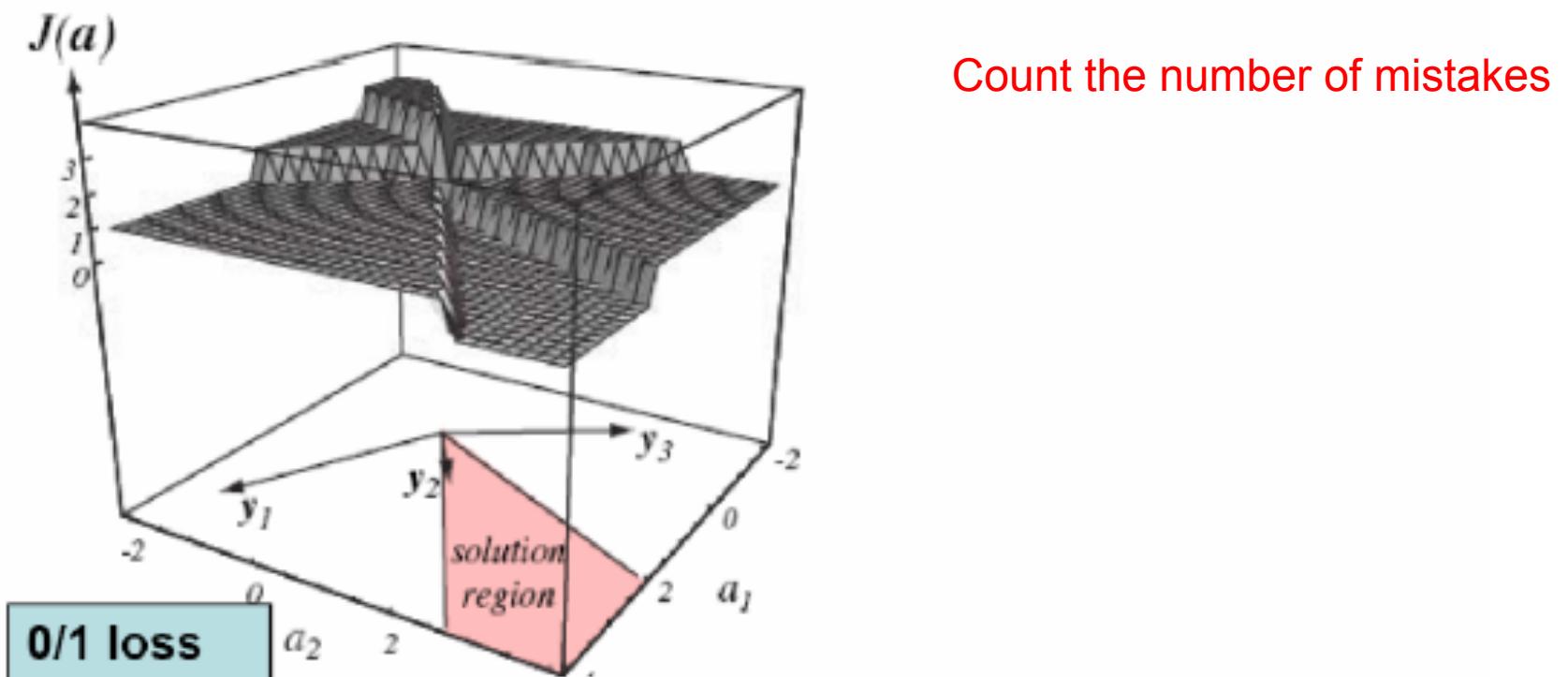
$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N loss(\mathbf{x}_i, \mathbf{y}_i; h)$$

- Background preferences over h can be included in **regularized empirical risk minimization**:

$$\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N loss(\mathbf{x}_i, \mathbf{y}_i; h) + R(h)$$

0-1 Loss function for Classification

- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N} \sum_{i=1}^N L(\text{sgn}(w \cdot x_i), y_i)$
 $L(y', y) = 0$ when $y' = y$, otherwise $L(y', y) = 1$
- Does not produce useful gradient since the surface of J is flat

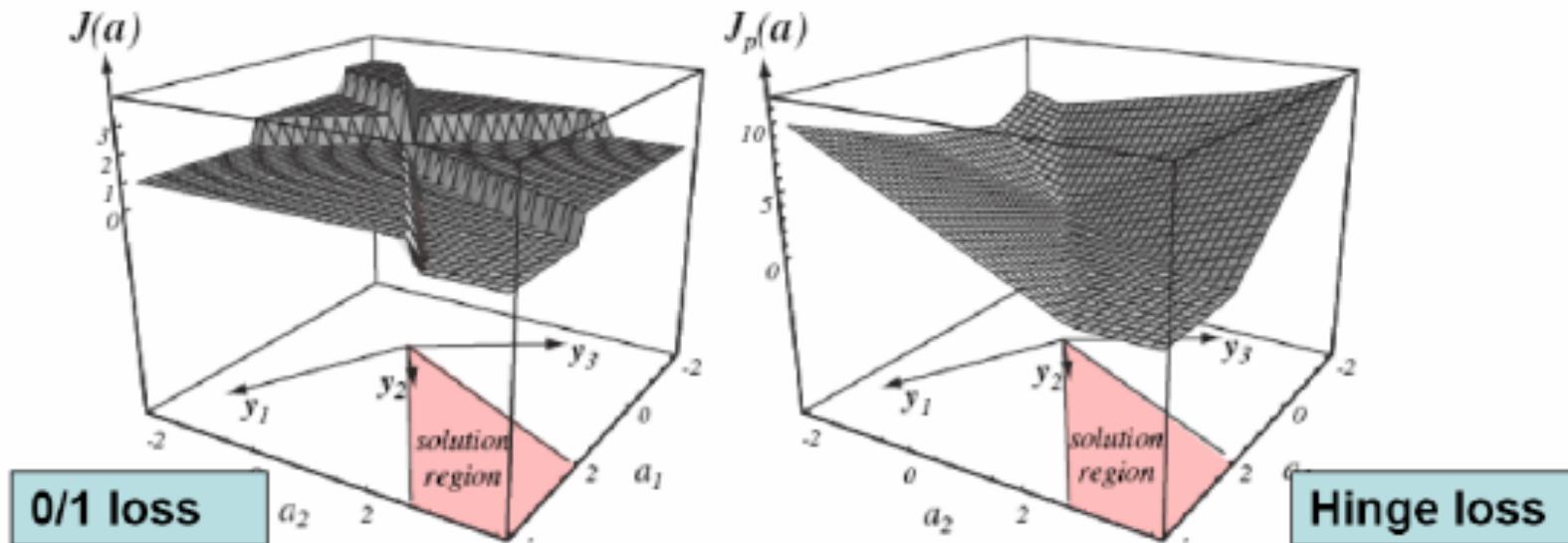


Hinge Loss

- Instead we will consider the “**hinge loss**”:

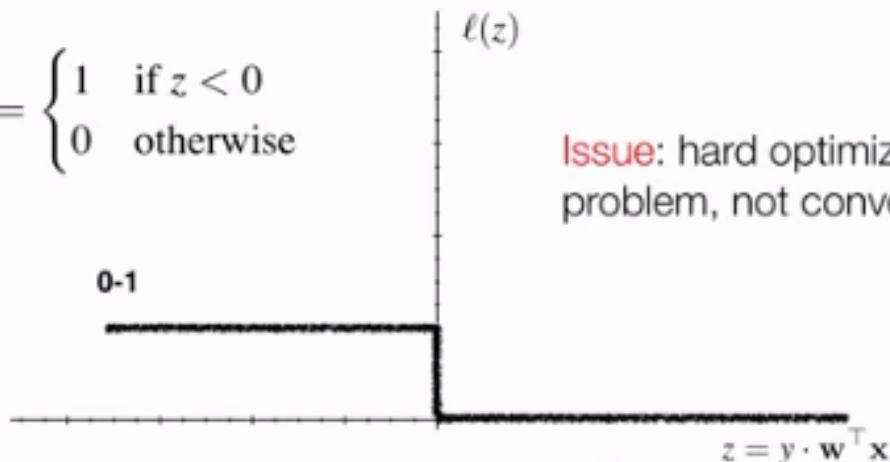
$$J_p(w) = \frac{1}{N} \sum_{i=1}^N \max(0, -y_i w \cdot x_i)$$

- The term $\max(0, -y_i w \cdot x_i)$ is 0 when y_i is predicted correctly otherwise it is equal to the “confidence” in the mis-prediction
- Has a nice gradient leading to the solution region



0/1 Loss Minimization

$$\ell_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$



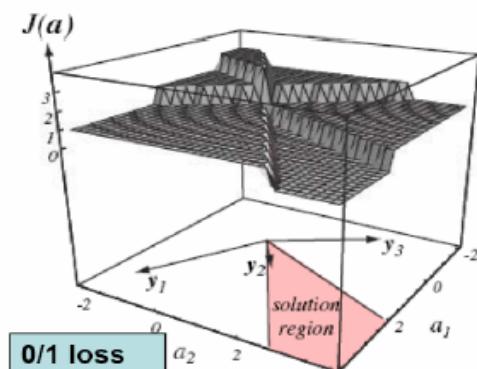
Issue: hard optimization problem, not convex!

0/1 loss is Not Convex
Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables);

Let $y \in \{-1, 1\}$ and define $z = y \cdot w^\top x$

z is positive if y and $w^\top x$ have same sign, negative otherwise

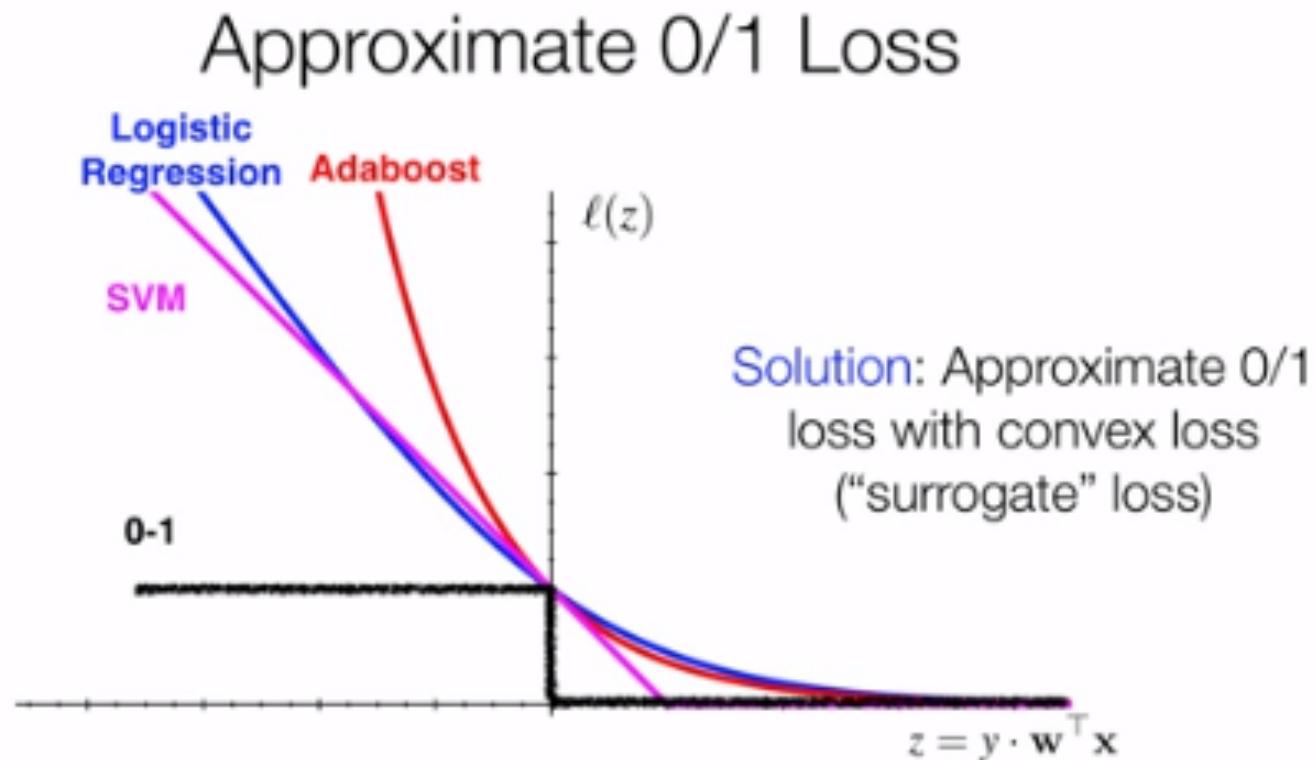
- 0/1 Loss function: $J_{0/1}(w) = \frac{1}{N} \sum_{i=1}^N L(\text{sgn}(w \cdot x_i), y_i)$
 $L(y',y) = 0$ when $y' = y$, otherwise $L(y',y) = 1$
- Does not produce useful gradient since the surface of J is flat



What can we do?

- Approximate 0/1 loss with a convex loss surrogate
- E.g., Hinge Loss Function

Zero-one loss versus Hinge Loss



SVM (hinge), Logistic regression (logistic), Adaboost (exponential)

Loss Terms: 5 examples

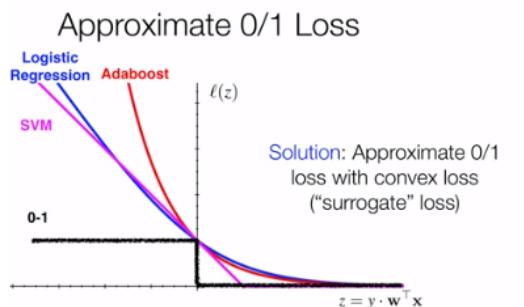
First: 0-1 loss and Hinge loss

In this section, we use 5 examples to illustrate the loss term. The five example are the Gold Standard(ideal case),Hinge(for soft margin SVM), log(for logistic regression,cross entropy error), squared loss(for linear regression) and Boosting.

First, let's see the “gold standard” loss function. We've implicitly been using it to evaluate our classifiers - we count the number of mistakes that are made. This is often called “0-1” loss, or L_{01}

$$L_{01}(m) = \begin{cases} 0 & \text{if } m \geq 0 \\ 1 & \text{if } m < 0 \end{cases} \quad \text{Count the number of mistakes}$$

Second, let's look at loss term for soft margin SVM. We use L_{hinge} to represent the hinge loss.



$$J(w) = \frac{1}{2} \|w\|^2 + \sum_i \max(0, 1 - y^i w^T x^i) \quad (14.5)$$

$$= \frac{1}{2} \|w\|^2 + \sum_i \max(0, 1 - m_i(w)) \quad (14.6)$$

$$= R_2(w) + \sum_i L_{hinge}(m_i) \quad (14.7)$$

Logistic Regression assumes a parametric form for $P(Y|X)$

Logistic Regression assumes a parametric form for the distribution $P(Y|X)$, then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is boolean is:

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (16)$$

and

$$P(Y = 0|X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (17)$$

Equivalently, we can work with the log of the conditional likelihood:

$$W \leftarrow \arg \max_W \sum_l \ln P(Y^l | X^l, W)$$

[https://www.cs.cmu.edu/~tom/mlbook/
NBayesLogReg.pdf](https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf)

This conditional data log likelihood, which we will denote $l(W)$, can be written as

$$l(W) = \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W)$$

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

Maximize

Loss term $L(M)$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Maximum Likelihood Estimator for Logistic Regression

Cost Function

- Model output

$$\hat{y}_k = P(y_k = 1 | \varphi_k)$$

- Likelihood contribution

$$l_{\theta,k} = \hat{y}_k^{y_k} (1 - \hat{y}_k)^{1-y_k}$$

- Likelihood function

$$L_\theta = \prod_{k=1}^N l_{\theta,k}$$

- Log-likelihood function

$$\ln L_\theta = \sum_{k=1}^N (y_k \ln \hat{y}_k + (1 - y_k) \ln(1 - \hat{y}_k))$$

Maximum Likelihood Criterion

$$\max_{\theta} \ln L_\theta \Leftrightarrow \min_{\theta} -2 \ln L_\theta$$

Mistake in Charles's note

Writing the regularized optimization problem as a minimization gives

Minimization

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{i=1}^n -\log p(y_i|x_i; \beta) + \mu \sum_{j=0}^d \beta_j^2.$$

Missing label in this formulation

The expression $-\log p(y_i|x_i; \beta)$ is called the “loss” for training example i . If the predicted probability, using β , of the true label y_i is close to 1, then the loss is small. But if the predicted probability of y_i is close to 0, then the loss is large. Losses are always non-negative; we want to minimize them. We also want to minimize the numerical magnitude of the trained parameters.

If the predicted probability, using β , of the true label y_i is close to 1, then the loss is small. But if the predicted probability of y_i is close to 0, then the loss is large. Losses are always non-negative; we want to minimize them

<http://cseweb.ucsd.edu/~elkan/250B/logreg.pdf>

<http://cseweb.ucsd.edu/~elkan/250B/logreg.pdf>

$$J(w) = \lambda \|w\|^2 - \sum_i \log(1 + e^{-y^{(i)} f_w(x^{(i)})})$$

Third, let's see log loss, which is equivalent to the cross entropy loss function used to train a logistic regression model

$$J(w) = \lambda \|w\|^2 - \sum_i y^i \log g_w(x^{(i)}) + (1 - y^i)(\log 1 - g(x^{(i)})), y^i \in (0, 1) \quad (14.8)$$

Simplify log conditional likelihood to get log a more succinct loss component

$$f_w(x^{(i)}) = w^T x^{(i)}$$

$$\begin{aligned} 1 - g(x^{(i)}) &= 1 - \frac{1}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{e^{-f_w(x^{(i)})}}{1 + e^{-f_w(x^{(i)})}} \\ &= \frac{1}{1 + e^{f_w(x^{(i)})}} \end{aligned}$$

So, we can transform the equation into the following form,

$$J(w) = \lambda \|w\|^2 - \sum_i \log(1 + e^{-y^{(i)} f_w(x^{(i)})})$$

Loss L(M)

$$L(m) = \log(1 + e^{-m})$$

Where m is defined

$$m^i = y^{(i)} f_w(x^{(i)})$$

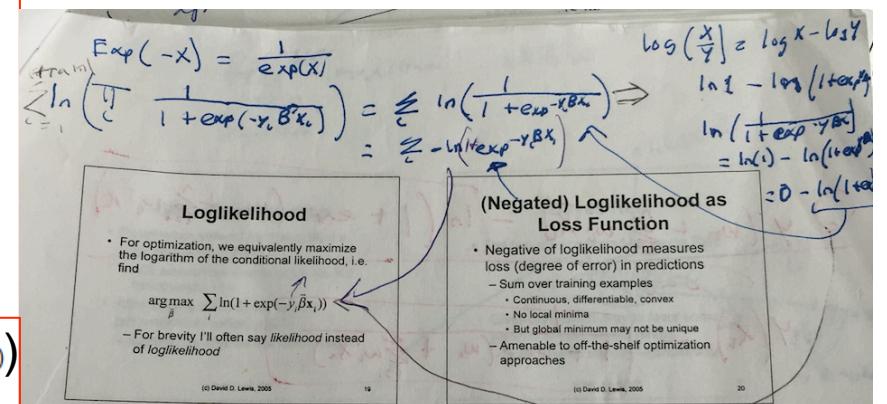
$$y^{(i)} = \begin{cases} -1 & \text{if } y^{(i)} = 0 \\ 1 & \text{if } y^{(i)} = 1 \end{cases}$$

Minimize log loss

Minimize the **NEG** log joint conditional likelihood
The conditional likelihood of θ given data x and y is $L(\theta; y|x) = p(y|x) = f(y|x; \theta)$.

$$g_w(x^{(i)}) = \frac{1}{1 + e^{-f_w(x^{(i)})}}$$

$$1 - g(x^{(i)}) = \frac{1}{1 + e^{f_w(x^{(i)})}}$$



Maximize Log loss

This is a maximize version of the log conditional likelihood

$$\hat{\beta} = \operatorname{argmax}_{\beta} LCL - \mu \|\beta\|_2^2$$

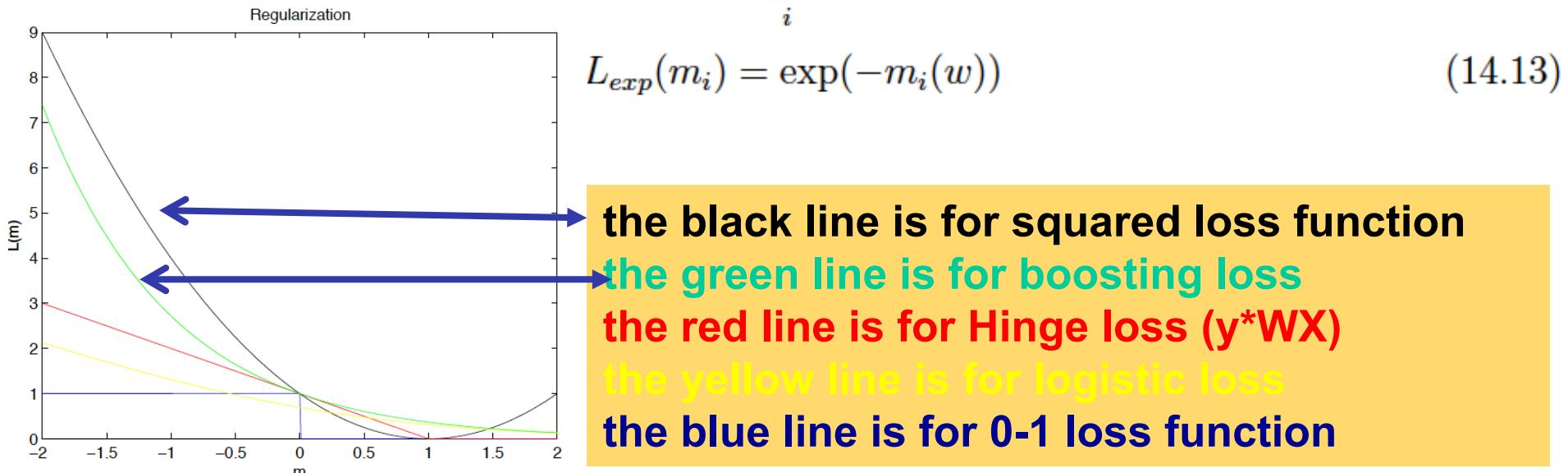
4: Squared Error loss term; 5: Exponential Loss term

Fourth, let's see Linear Regression, which have a squared loss term. We will use L_2 to scribe the squared loss term. We can see from fig(1) the squared term is much higher than the Gold Stand, so it is a bad function.

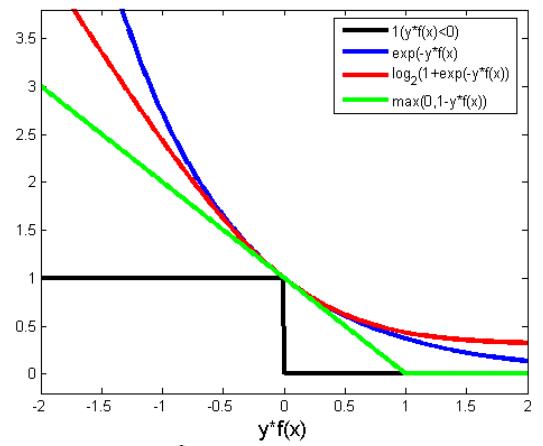
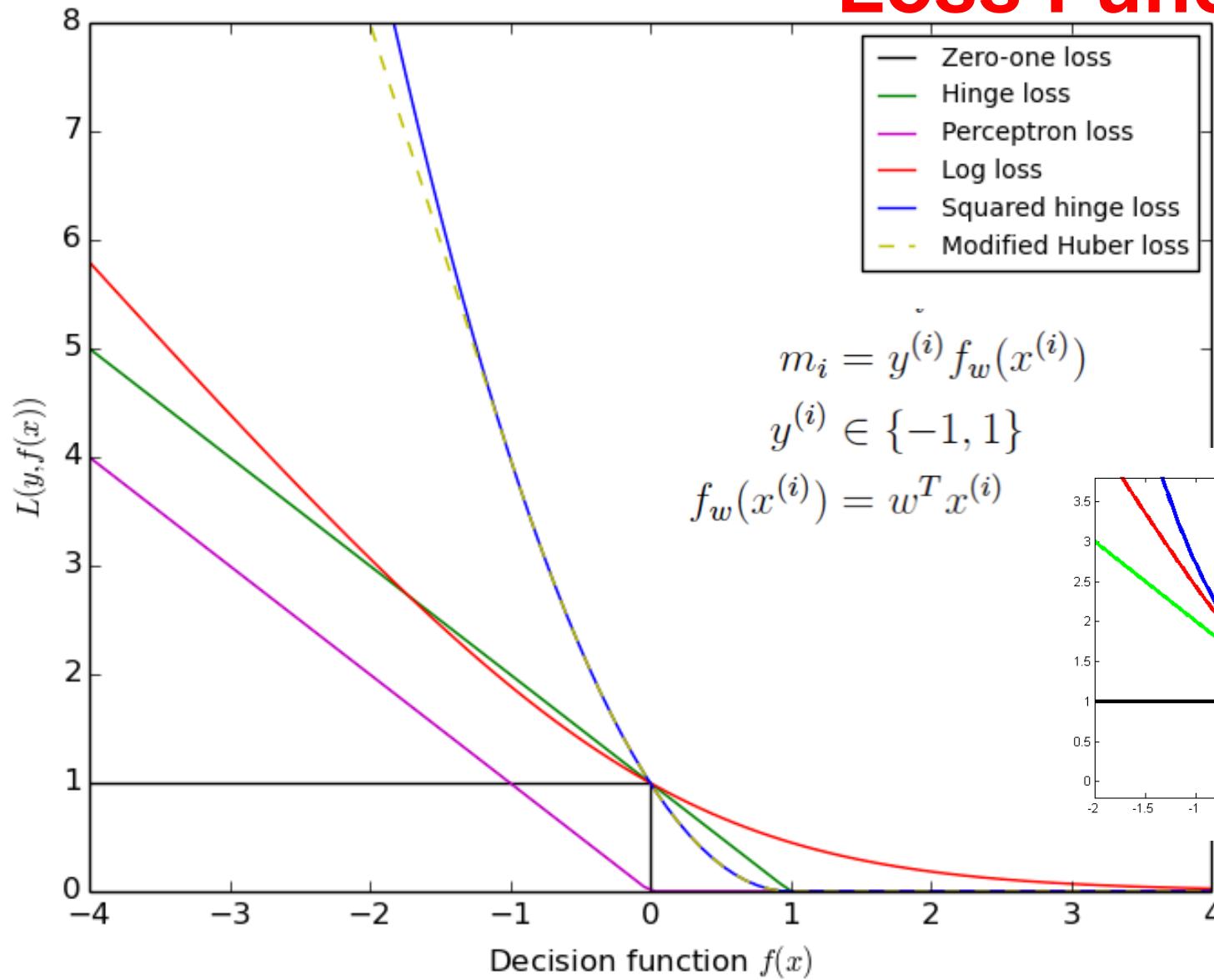
$$L_2(m) = (f_w(x) - y)^2 = (m - 1)^2 \quad (14.11)$$

Later on, we'll look at a learning algorithm called boosting. It can be seen as a greedy optimization of the exponential loss term,

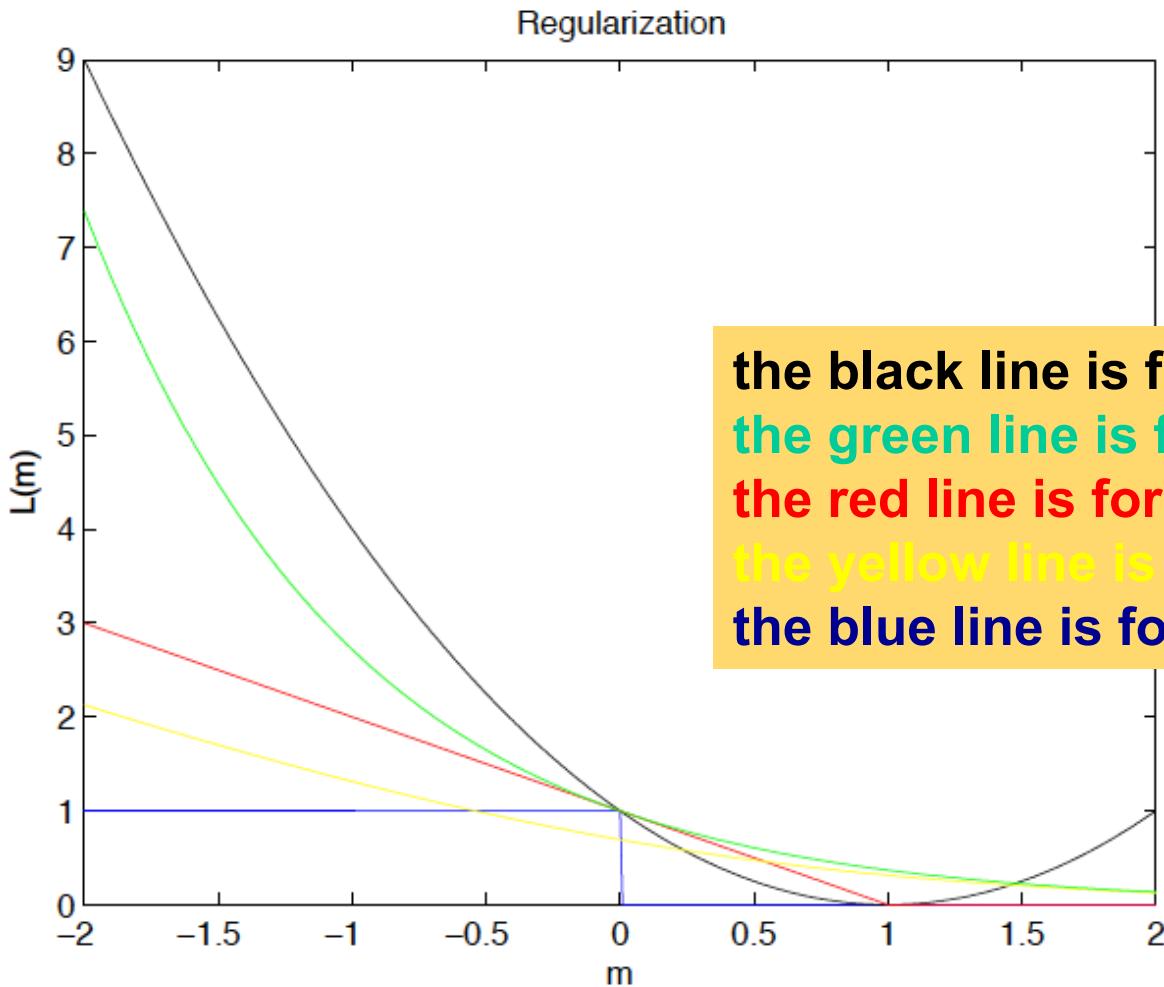
$$J(w) = \lambda R(w) + \sum_i \exp(-y^{(i)} f_w(x^{(i)})) \quad (14.12)$$



Loss Functions



Loss Functions



the black line is for squared loss function
the green line is for boosting loss
the red line is for Hinge loss (y^*WX)
the yellow line is for logistic loss
the blue line is for 0-1 loss function

Let $W = (0,0,...)$

Repeat

For j in $0..n$ #each variable

For i in $1..m$ #each example

$$W_{j,t+1} = W_{j,t} + \alpha * (y - p)X_j$$

until convergence (i.e., no big changes in W or error)

Loss functions; a unifying view

- Loss function consists of:
 - loss term ($L(m_i(w))$, expressed in terms of the margin of each training example) and
 - regularization term ($R(w)$ expressed as a function of the model complexity)

$$J(w) = \sum_i \text{Loss term } L(m_i(w)) + \text{regularization term } \lambda R(w) \quad (14.1)$$

$$m_i = y^{(i)} f_w(x^{(i)}) \quad (14.2)$$

$$y^{(i)} \in \{-1, 1\} \quad (14.3)$$

$$f_w(x^{(i)}) = w^T x^{(i)} \quad (14.4)$$

ML Objectives

$$J(w) = \sum_i \text{Loss term } L(m_i(w)) + \text{regularization term } \lambda R(w)$$

Objective Function

Almost all machine learning objectives are optimized using this update

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^n g(J(w) x_i, y_i)$$

Update weight vector with gradient of the objective function

w is a vector of dimension d
we're trying to find the best w via optimization

Gradient Descent in ML Algorithms

- OLS
- Logistic Regression
- Bayesian logistic regression
- Probit regression
- Perceptron??
- SVMs and its many learning algorithms
 - Linear SVMs
- Neural Networks
- Ensemble models (e.g., gradient boosted decision trees, ADABoost)

OLS using Gradient Descent

$$J_q(W, X_1^m) = \text{Minimize} \frac{1}{2m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

**OLS Objective Function
With decision variables W**

$$g(W; X_i, Y_i) = (y^i - W_i X^i) X^i$$

- Initialize $W = \text{vector or zeros}$
- Repeat until Convergence

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

OLS Batch Update Rule

- End Repeat

True gradient is approximated by the gradient of the cost function only evaluated at all examples; adjust parameters proportional to this approx. gradient.

Intuitively, drag weight vector closer to the incorrectly predicted examples

Distributed Gradient Descent: E.g., Linear Regression

Master/Slave Process

- Initialize model parameters, assume a weight vector $W=(0, 0, \dots, 0)$; Gradient = $(0, 0, \dots, 0)$
- While not converged
 - MASTER: Broadcast model (i.e., weight vector) to the worker nodes
 - MASTER launches map-reduce jobs
 - Mappers (MANY mappers) to compute partial gradients over the respective training data subsets (chunks)
 - Init $g=(0, 0, \dots, 0)$
 - For each training example:
 - » Compute partial gradient for each training example
 - » Combine in memory; $g = \sum_i(g(W; X_i, Y_i))$
 - Finally Yield the partial gradient g
 - Reducer (single Reducer)
 - Initialize full gradient: $G=(0,0,\dots,0)$
 - For each partial gradient g
 - » Aggregate partial gradients: $G = \sum_m g_m$
 - Yield full gradient G
 - MASTER $W = W - \alpha G$ //update the weight vector
 - End-While

Linear regression

Goal: learn a weight vector W

Gradient is defined here:

$$g(W; X_i, Y_i) = (y^i - W_i X^i) X^i$$

Quiz 11.3.1 Loss Functions

- Loss function consists of:
 - A: loss term ($L(m_i(w))$, expressed in terms of the margin of each testing example) and
 - B: loss term ($L(m_i(w))$, expressed in terms of the margin of each support vector) and
 - C: regularization term ($R(w)$ expressed as a function of the model complexity)
 - D: 0/1 loss is Not Convex: Since the surface of the loss function (sum over all training data is flat for various intervals of weight values (decision variables))

Which of the above statements are true?

- A, B, C, D
- B, C, D
- C, D CORRECT
- None of the above

Maximum vs Minimum

$f(x) = x^3 - 12x + 1$

Find the roots of the gradient function using the Newton-Raphson algorithm

FOC

$f'(x=x^*) = 0$ and
These zero points are called stationary points.

Second derivative

Gives the “rate of change” of the first derivative

SOC

- If $f''(x=x^*) < 0$ then $x=x^*$ is a maximum
- If $f''(x=x^*) > 0$ then $x=x^*$ is a minimum
- If $f''(x=x^*) == 0$ then the test is inconclusive

Find the roots of the gradient function using the Newton-Raphson algorithm

FOC △ $f(x)$ = a vector of zeros

Use SOC to establish if extremum is a maximum or minimum

$f(x) = x^3 - 12x + 1$

$F'(x) = 3x^2 - 12$

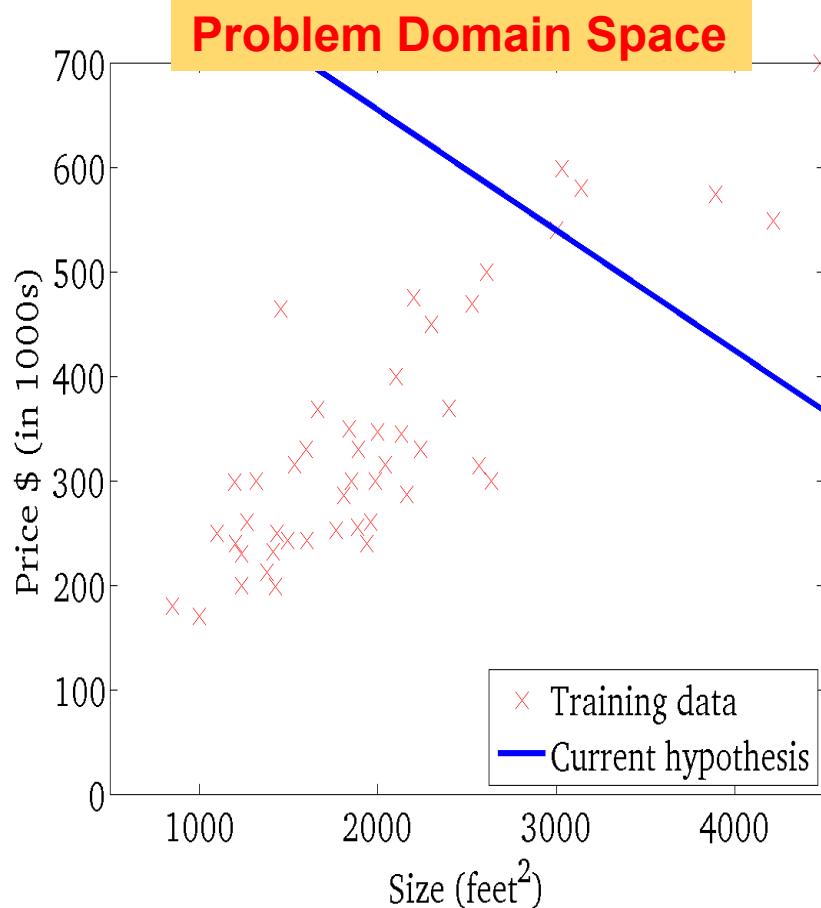
$F'(x) = 6x$

Gradient Descent for LR Price~Size Iteration 0

Eqn Line $y = aX + b$

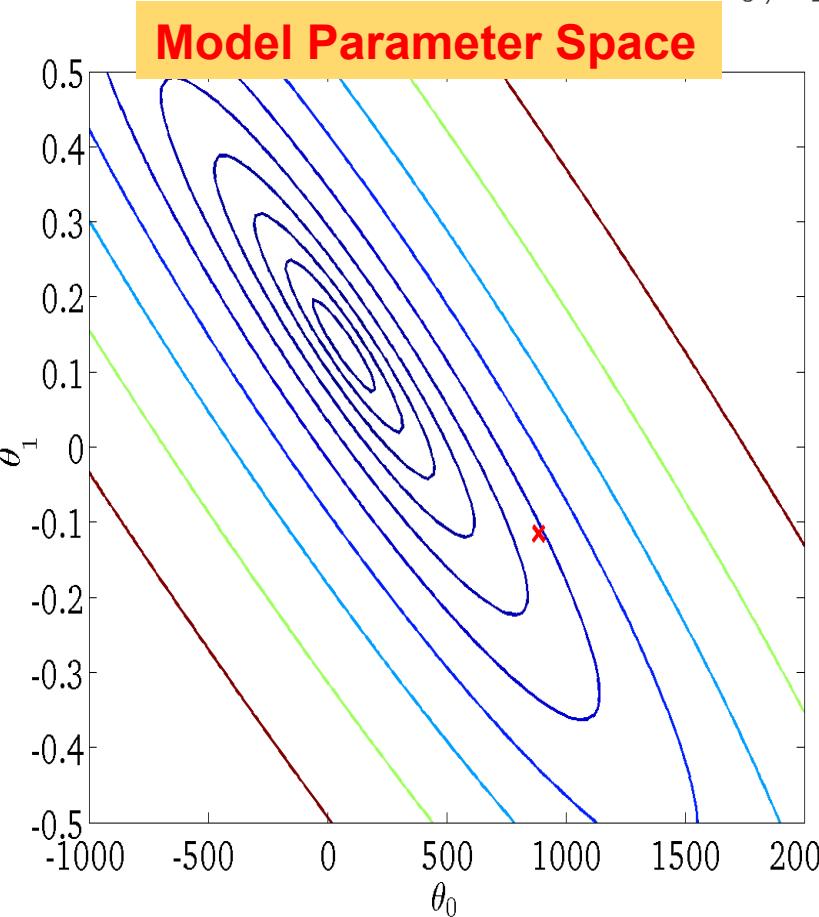
$$Y = w_1 X + W_0$$

(for fixed θ_0, θ_1 , this is a function of x)



$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

(function of the parameters θ_0, θ_1)

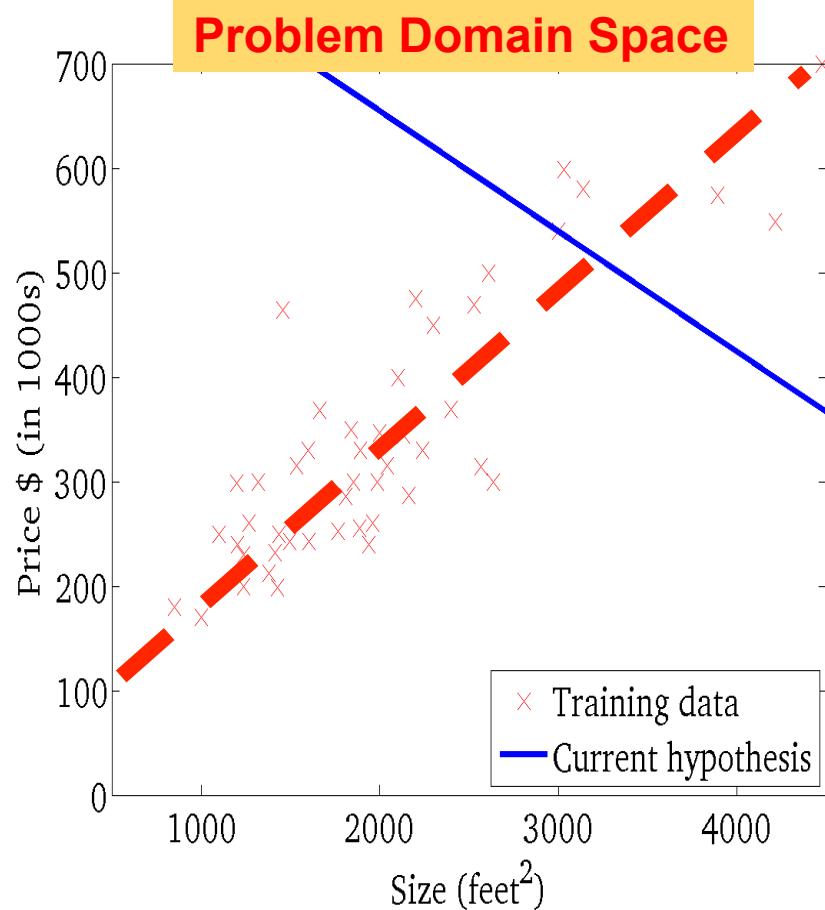


Gradient Descent for LR Price~Size Iteration 0 vs Iteration Opt

Eqn Line $y = aX + b$

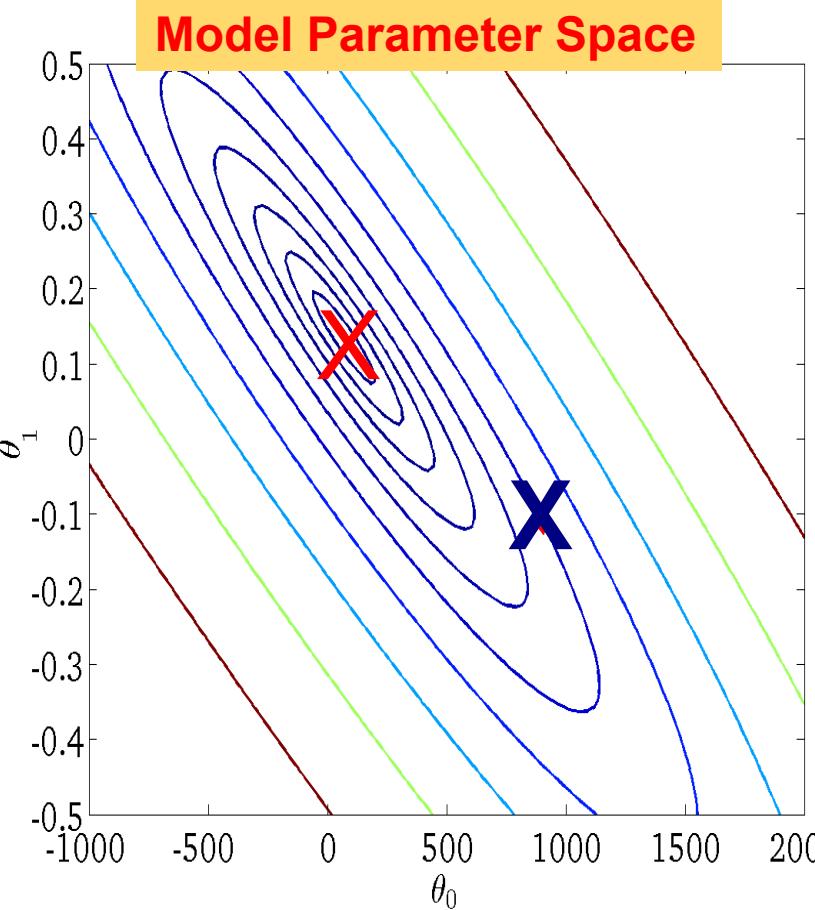
$$Y = w_1 X + W_0$$

(for fixed θ_0, θ_1 , this is a function of x)



$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

(function of the parameters θ_0, θ_1)



Closed form solution to OLS

- Convex optimization tells us that the first order condition
 - $\nabla F(X) = 0$ at the global minimum and we are going to exploit that here to get a closed form solution for linear regression

Closed form solution to OLS

SSE

$$S(\theta) = \sum_i [y(i) - \sum_j \theta_j x_{ij}]^2$$

$$= \sum_i e_i^2$$

$$= \underline{e}' \underline{e}$$

$$= (y - X\theta)' (y - X\theta)$$

$y = N \times 1$ vector
of target values

$(p+1) \times 1$ vector
of parameter values

$N \times (p+1)$ vector
of input values

$$\text{where } \underline{e} = y - X\theta$$

$$S(\theta) = \sum e^2 = \underline{e}' \underline{e} = (y - X\theta)' (y - X\theta)$$

$$= y' y - \theta' X' y - y' X \theta + \theta' X' X \theta$$

$$= y' y - 2\theta' X' y + \theta' X' X \theta$$

Minimize objective function
and find root vector of the
gradient function

Taking derivative of $S(\theta)$ with respect to the components of θ gives....

$$\frac{dS}{d\theta} = -2X'y + 2X'X\theta$$

Set this to 0 to find the extremum (minimum) of S as a function of θ ...

Normal Equations and solving for θ

Set to 0 to find the extremum (minimum) of S as a function of θ ...

$$\Rightarrow -2 \mathbf{X}' \mathbf{y} + 2 \mathbf{X}' \mathbf{X} \theta = 0$$

$$\Rightarrow \mathbf{X}' \mathbf{X} \theta = \mathbf{X}' \mathbf{y} \quad (\text{known in statistics as the Normal Equations})$$

Letting $\mathbf{X}' \mathbf{X} = \mathbf{C}$, and $\mathbf{X}' \mathbf{y} = \mathbf{b}$,
we have $\mathbf{C} \theta = \mathbf{b}$, i.e., a set of linear equations

We could solve this directly, e.g., by matrix inversion

$$\theta = \mathbf{C}^{-1} \mathbf{b} = (\mathbf{X}' \mathbf{X})^{-1} \mathbf{X}' \mathbf{y}$$

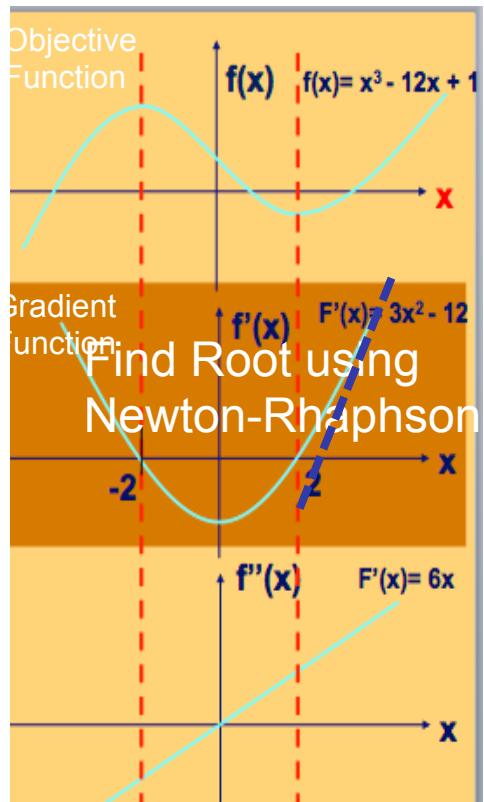
Closed form solution to OLS

Gradient Descent (a simpler root finder)

Given: Minimize $f(x)$

STEP 1: Find the zeros of the gradient function $f'(x)$

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$x^{i+1} = x^i - [f''(x^i)]^{-1} f'(x^i) \quad \text{Univariate case}$$

Newton-Raphson

$$x^{i+1} = x^i - [H(x^i)]^{-1} J(x^i) \quad \text{Multivariate Case}$$

Calculating $f''(x)$, the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

$$x^{i+1} = x^i - a^i f'(x^i) \quad \text{Univariate case}$$

Gradient Descent

$$x^{i+1} = x^i - a^i J(x^i) \quad \text{Multivariate Case}$$

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)

- Convergence criteria. E.g., decision vector X does not change that much

Linear Regression via Gradient Descent (a simpler root finder)

Given: Minimize $f(x)$

STEP 1: Find the zeros of the gradient function $f'(x)$

- Using Bisection method; OR Newton-Raphson; OR Gradient Descent



$$J_q(W, X_1^m) = \text{Minimize} \sum_{i=1}^m (W^T X_i - y_i)^2$$

RSS = Variance of ε

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^n (X_j W - y_i)^2 \right)}{\partial W}$$

Gradient vector of partial derivatives

$$\nabla J(W) = \left(\sum_{j=1}^n (X_j W - y_i) X_j \right)$$

Pull model closer to examples with biggest residual

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

Gradient Descent

For another derivation see:

<http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf>

OLS using Gradient Descent

$$J_q(W, X_1^m) = \text{Minimize} \frac{1}{2m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

**OLS Objective Function
With decision variables W**

- Initialize W = vector or zeros
- Repeat until Convergence

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

OLS Batch Update Rule

- End Repeat

True gradient is approximated by the gradient of the cost function only evaluated at all examples; adjust parameters proportional to this approx. gradient.

Intuitively, drag weight vector closer to the incorrectly predicted examples

OLS via Distributed Gradient Descent

- Master/Driver Process
- Initialize model parameters, $W = \text{vector of zeros}$
 $W = (0, 0, \dots)$
- While not converged
 - Broadcast model (e.g., weight vector) to the worker nodes
 - Mapper (MANY mappers)
 - Compute partial gradient for each training example
 - Combine in memory $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Finally Yield the partial gradient
 - Reducer (single Reducer)
 - Aggregate partial gradients
 - Yield full gradient $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$
 - Check for convergence

RSS = Variance of ϵ

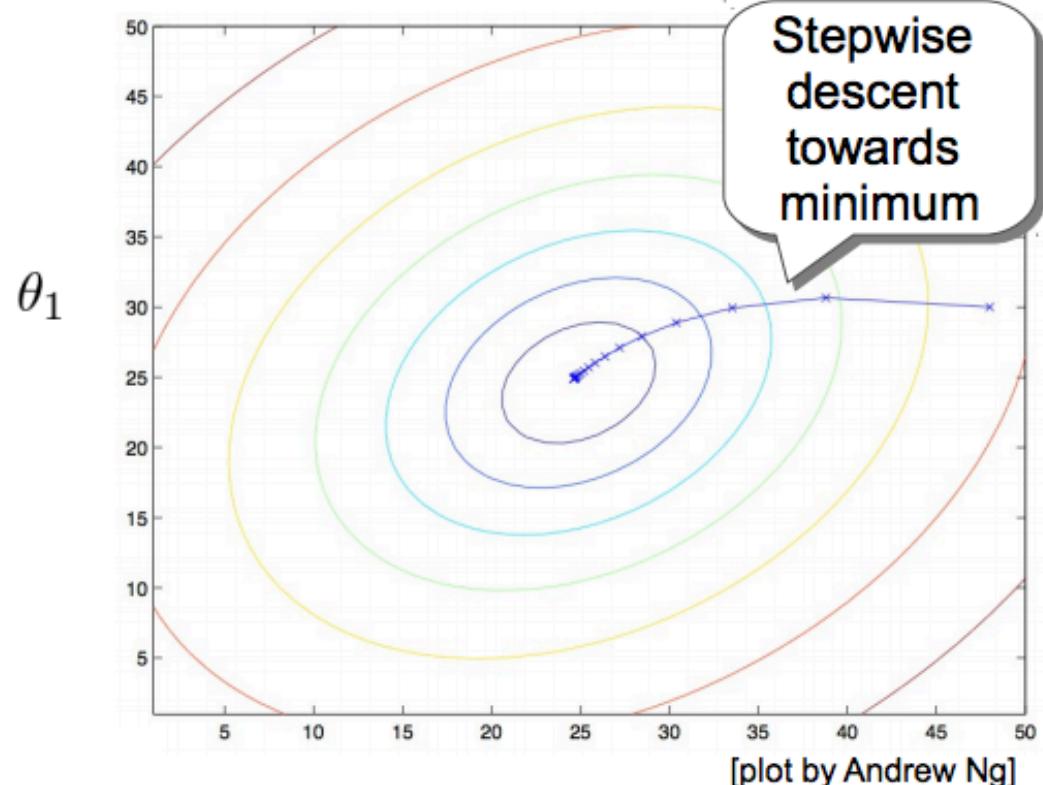
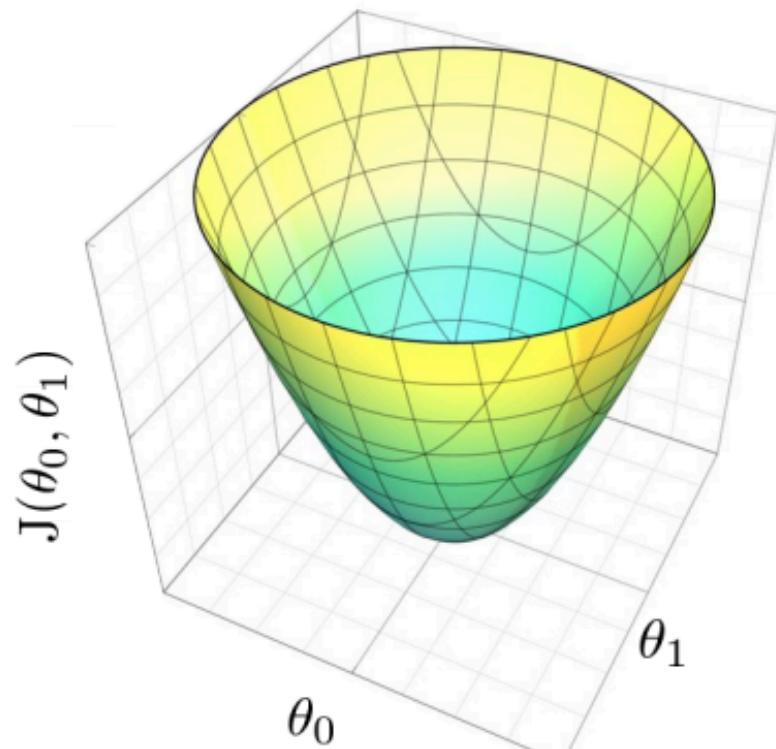
$$0 = \frac{\partial \sum \hat{\epsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^n (X_j W - y_j)^2 \right)}{\partial W}$$

$$\nabla J(W) = \left(\sum_{j=1}^n (X_j W - y_j) X_j \right)$$

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_j) X_j \right)$$

3D plots and contour plots

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$



$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1) = \arg \min_{\theta} J(\theta)$$

OLS via Gradient Descent

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

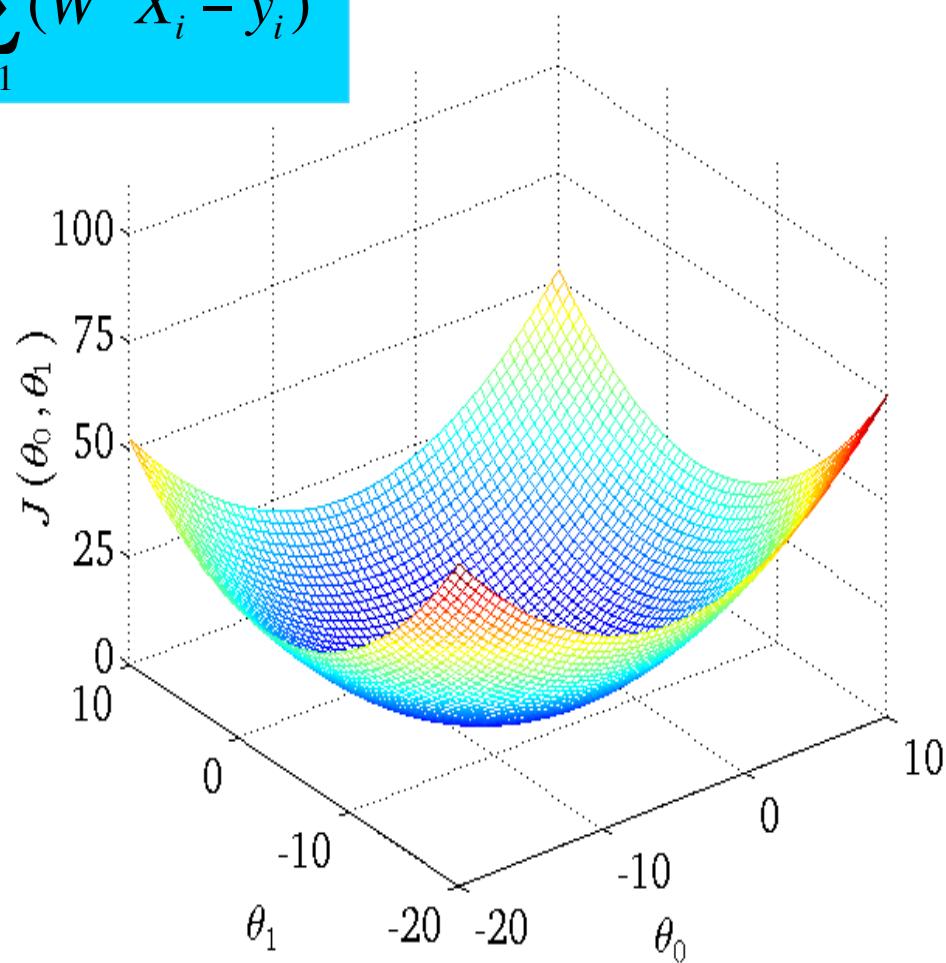
Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

Error Surface for OLS: Price~Size

$$J_q(W, X_1^L) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

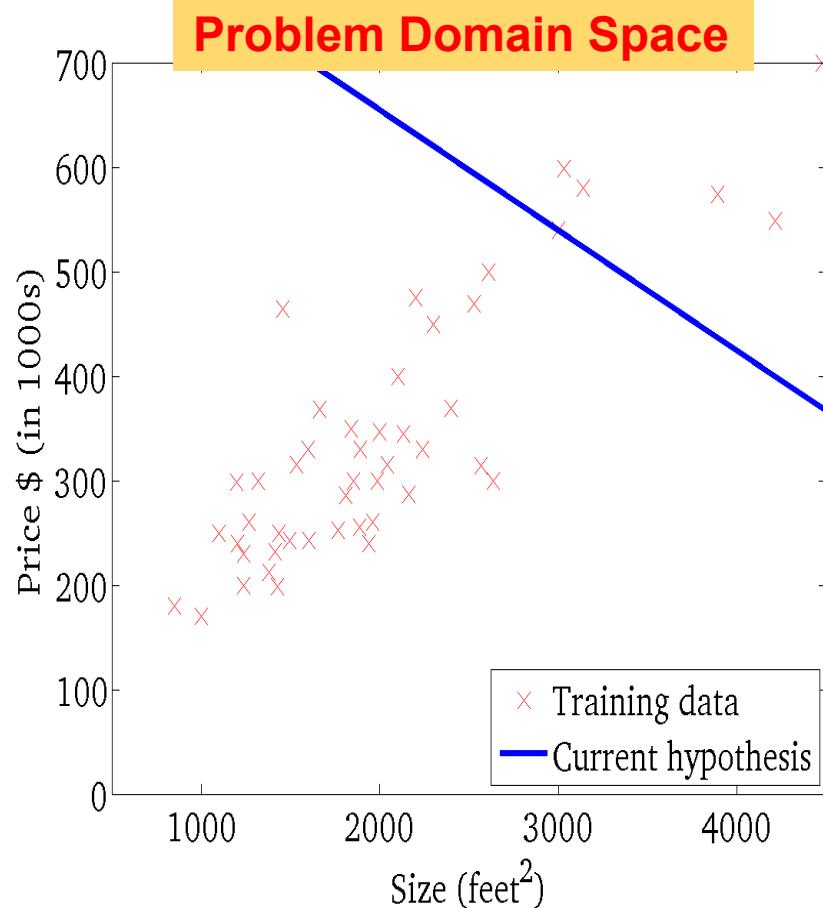


Gradient Descent for LR Price~Size Iteration 0

Eqn Line $y = aX + b$

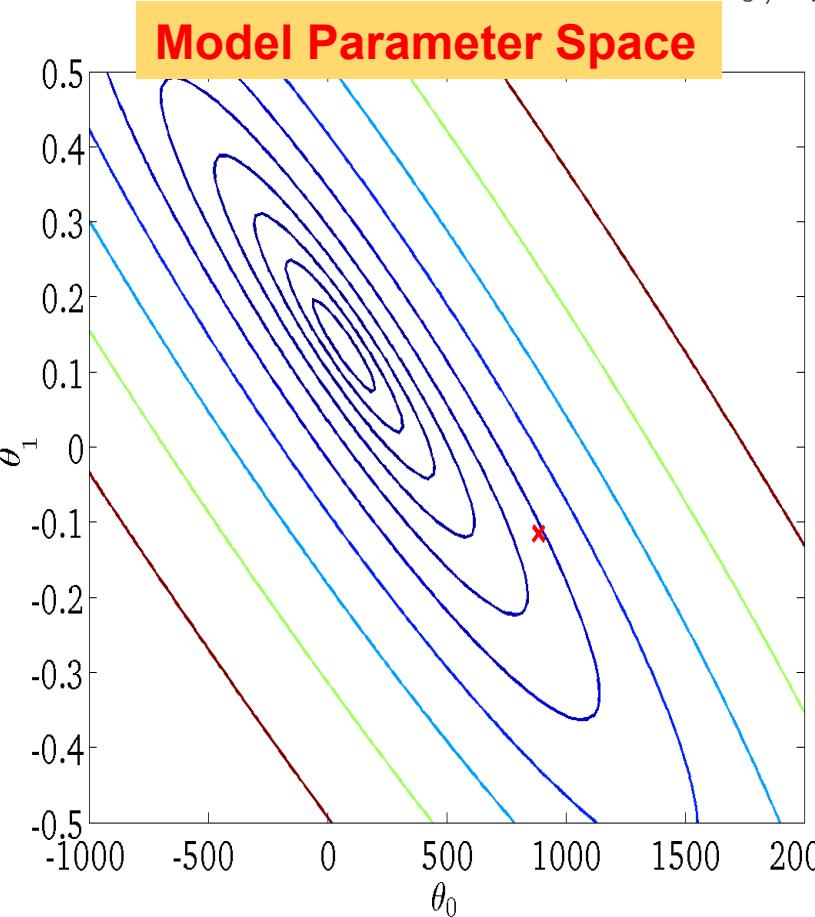
$$Y = w_1 X + w_0 \quad 1$$

(for fixed θ_0, θ_1 , this is a function of x)



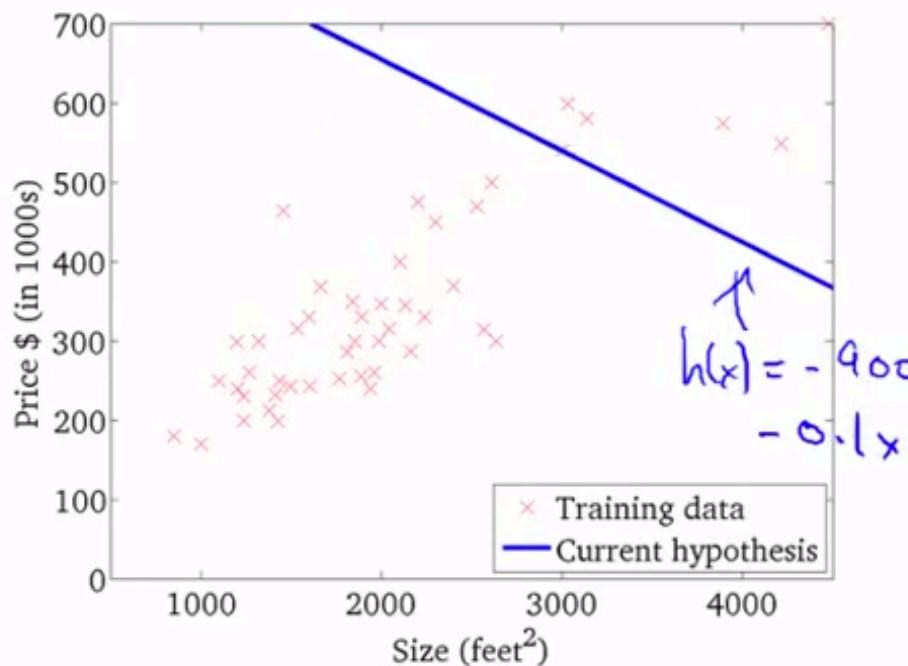
$$J(\theta_0, \theta_1) \quad J_q(W, X^L) = \frac{1}{m} \sum_{i=1}^m (W^T X_i - y_i)^2$$

(function of the parameters θ_0, θ_1)

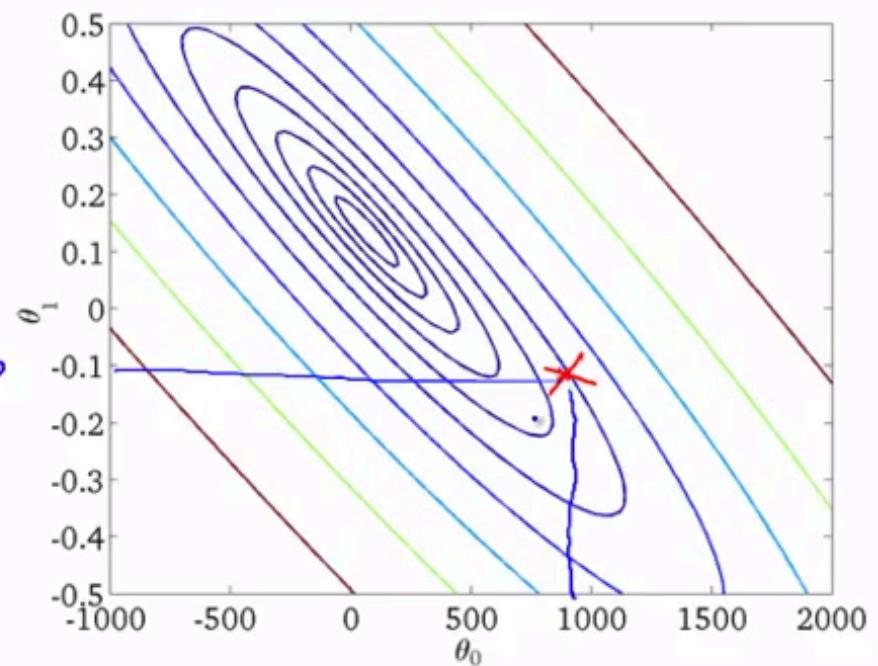


Gradient Descent for LR Price~Size – Iteration 0

$\underline{h_\theta(x)}$
(for fixed θ_0, θ_1 , this is a function of x)



$\underline{J(\theta_0, \theta_1)}$
(function of the parameters θ_0, θ_1)



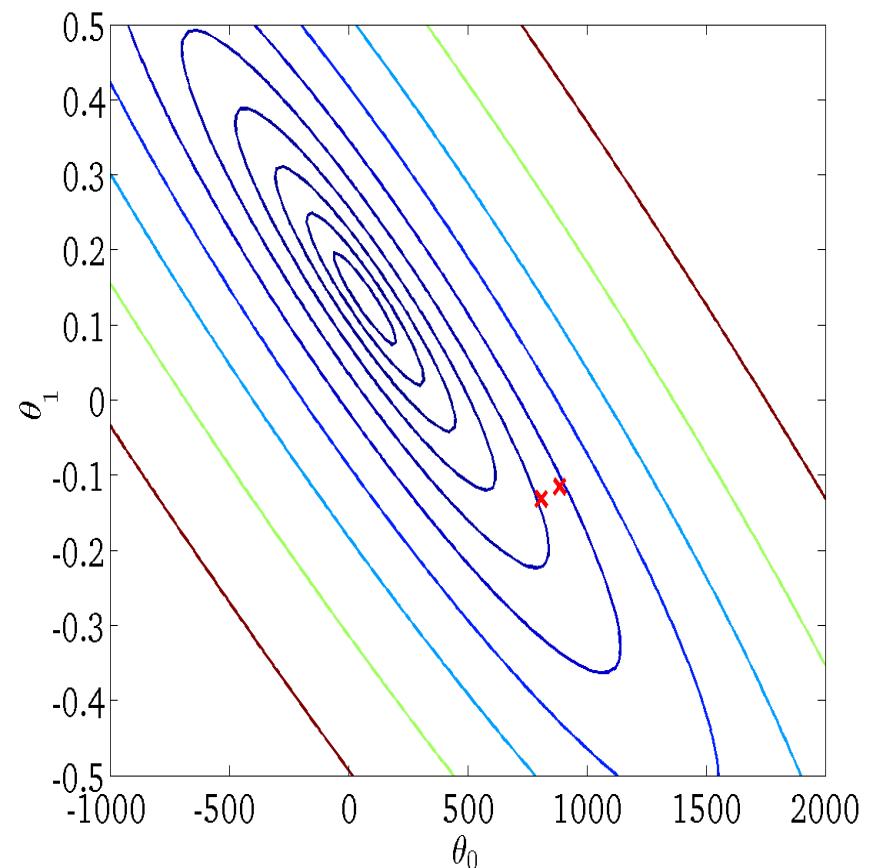
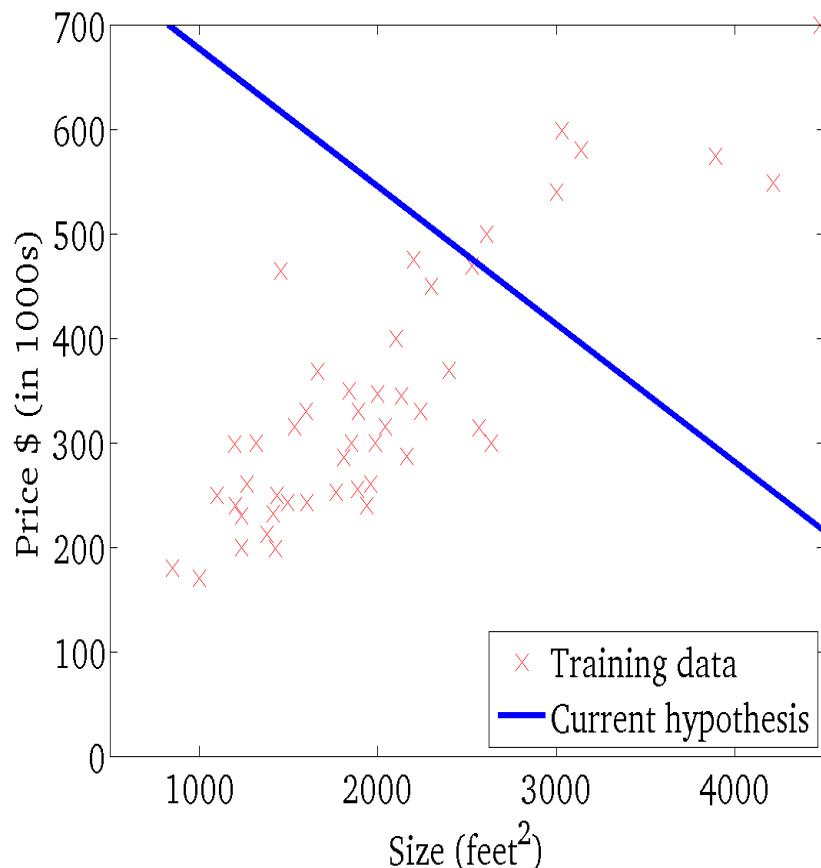
Gradient Descent for LR Price~Size – Iteration 1

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



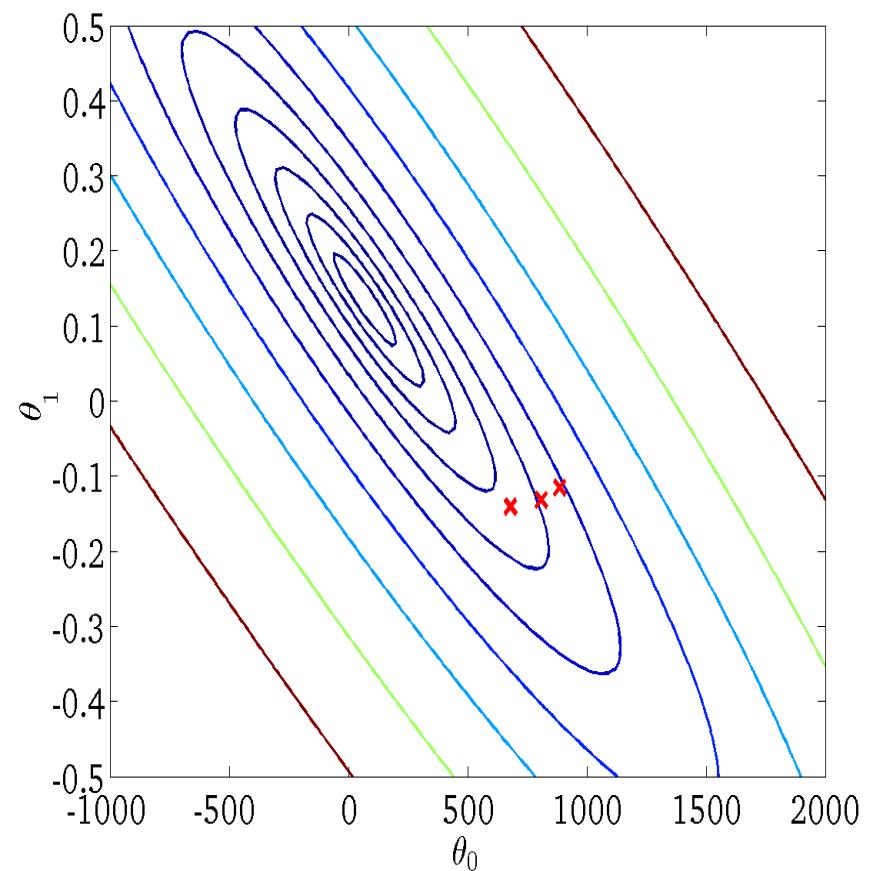
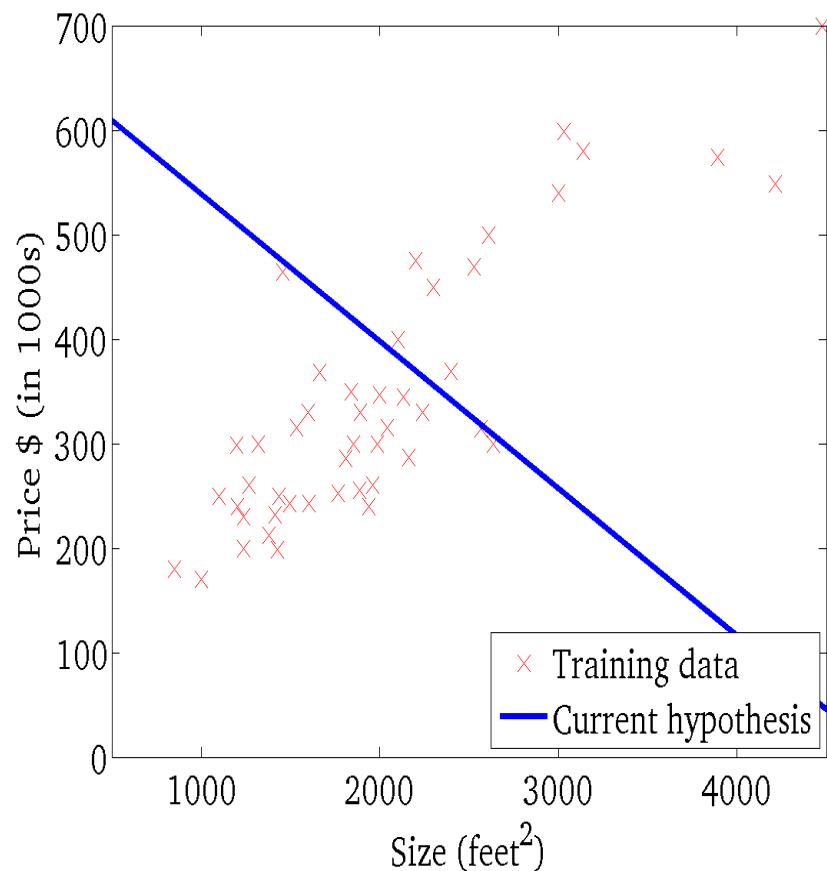
Gradient Descent for LR Price~Size – Iteration 2

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



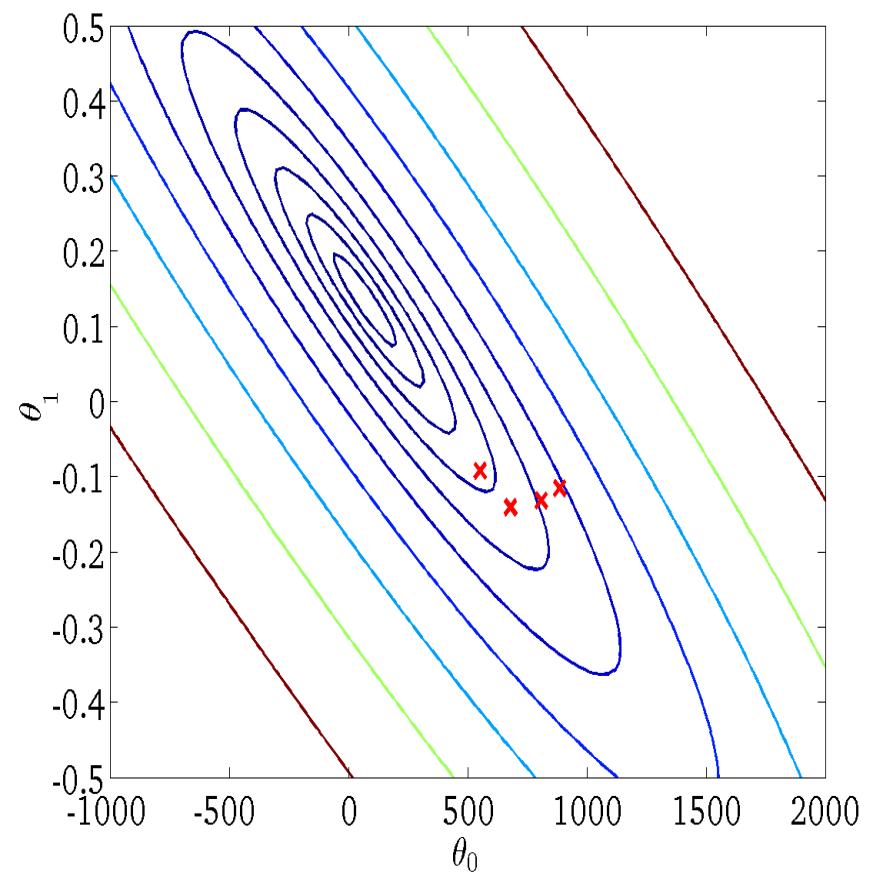
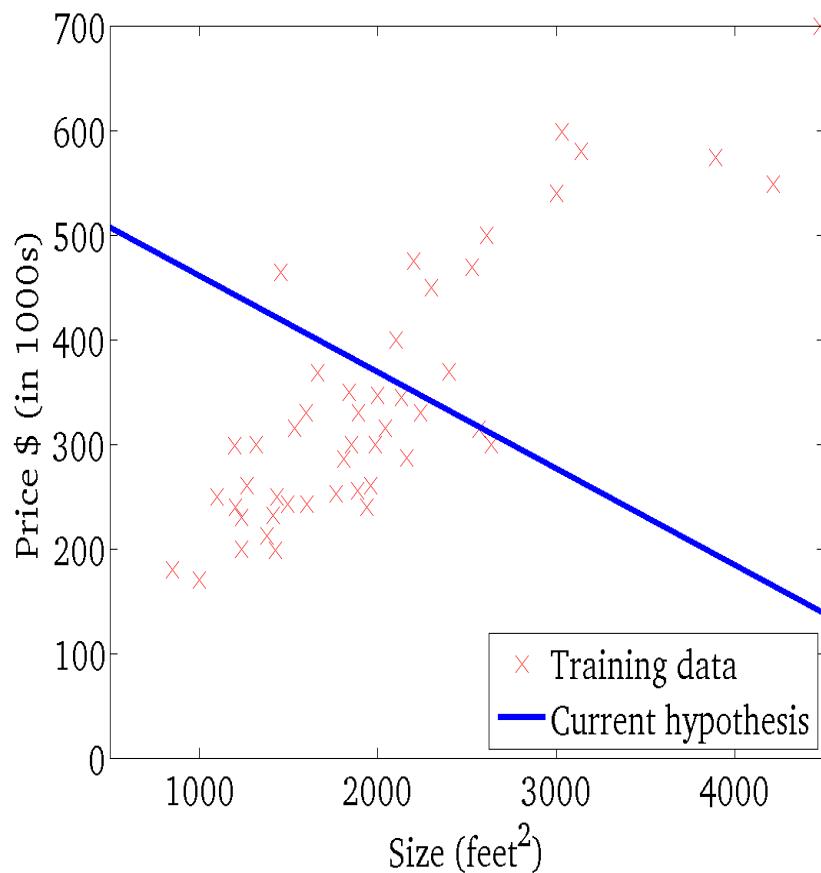
Gradient Descent for LR Price~Size – Iteration 3

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

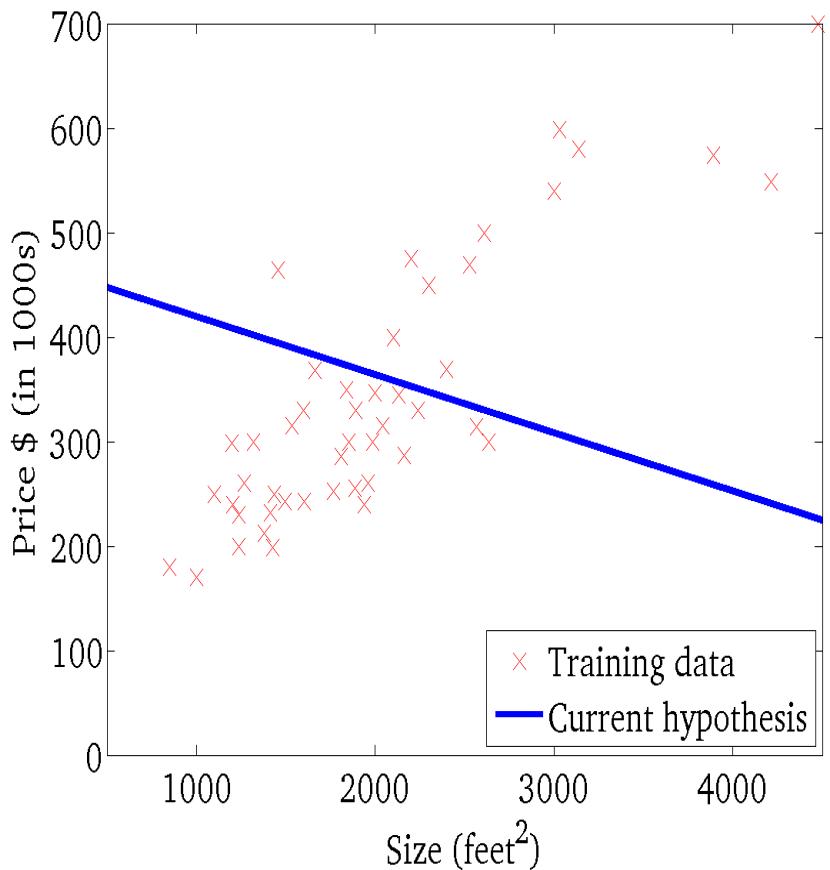
(function of the parameters θ_0, θ_1)



Gradient Descent for LR Price~Size – Iteration 4

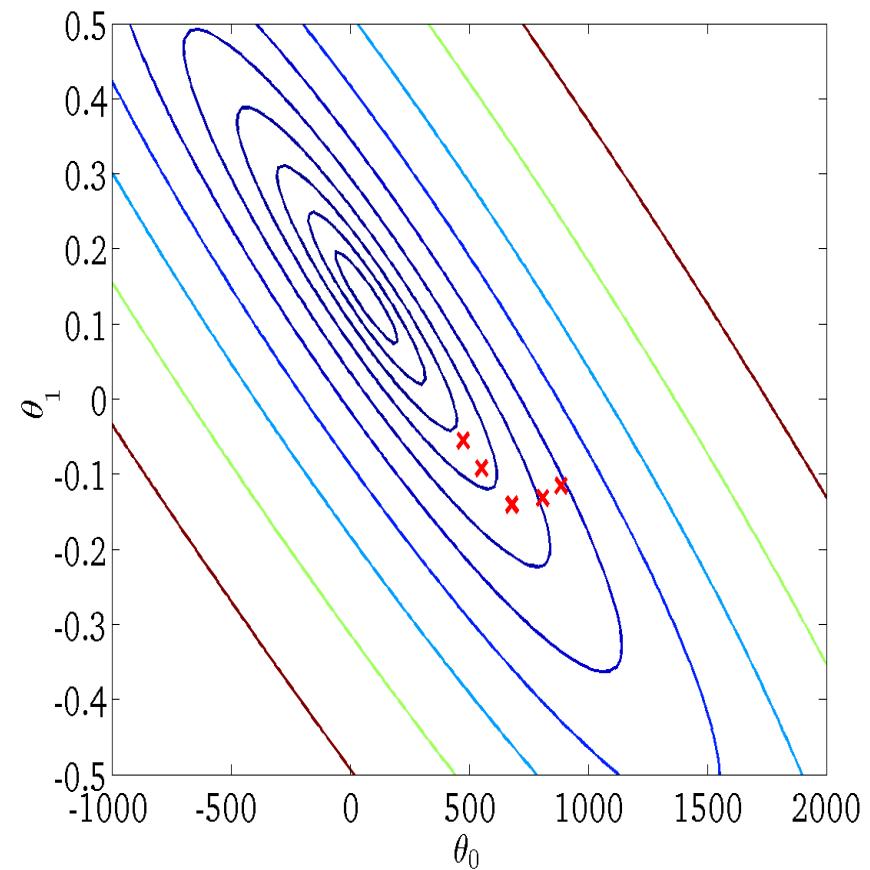
$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



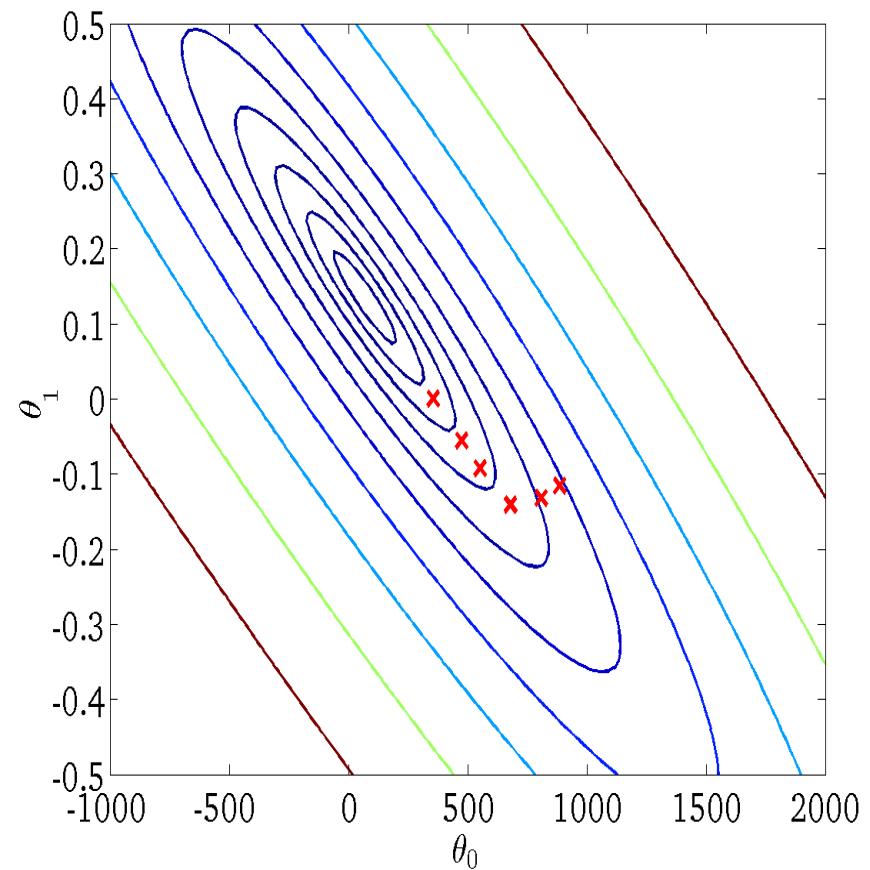
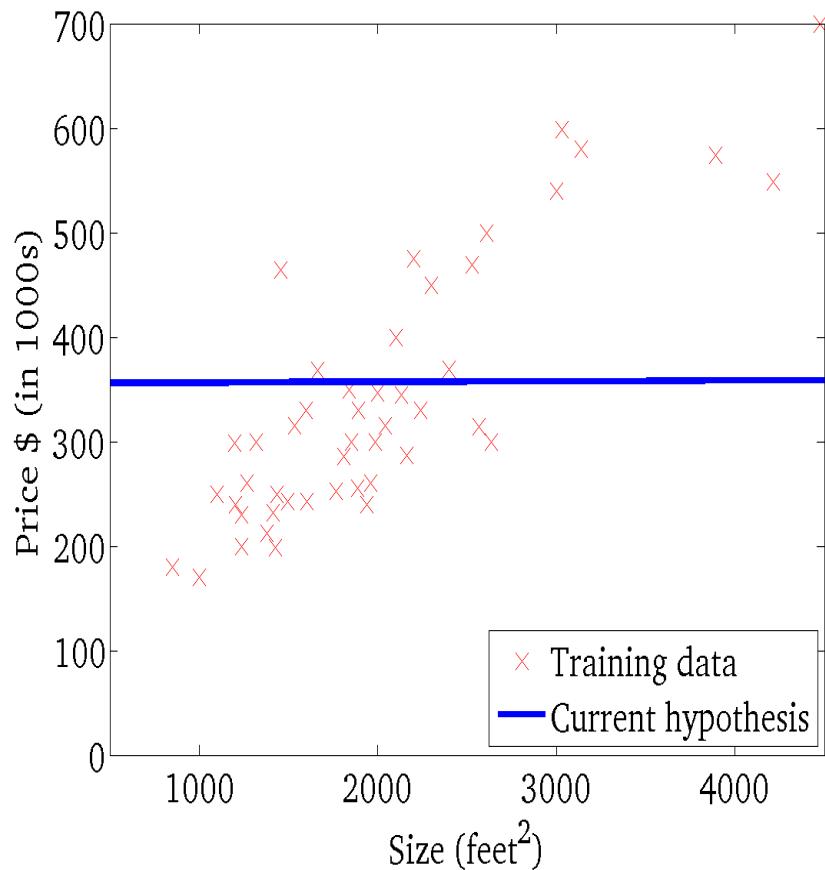
Gradient Descent for LR Price~Size – Iteration 5

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



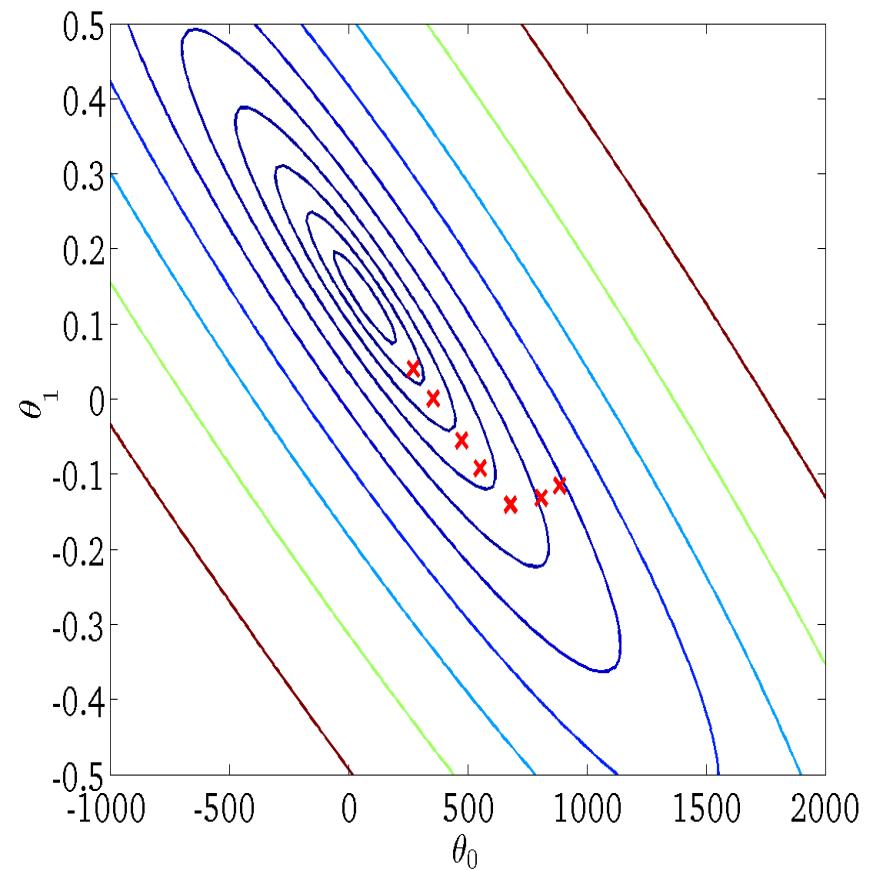
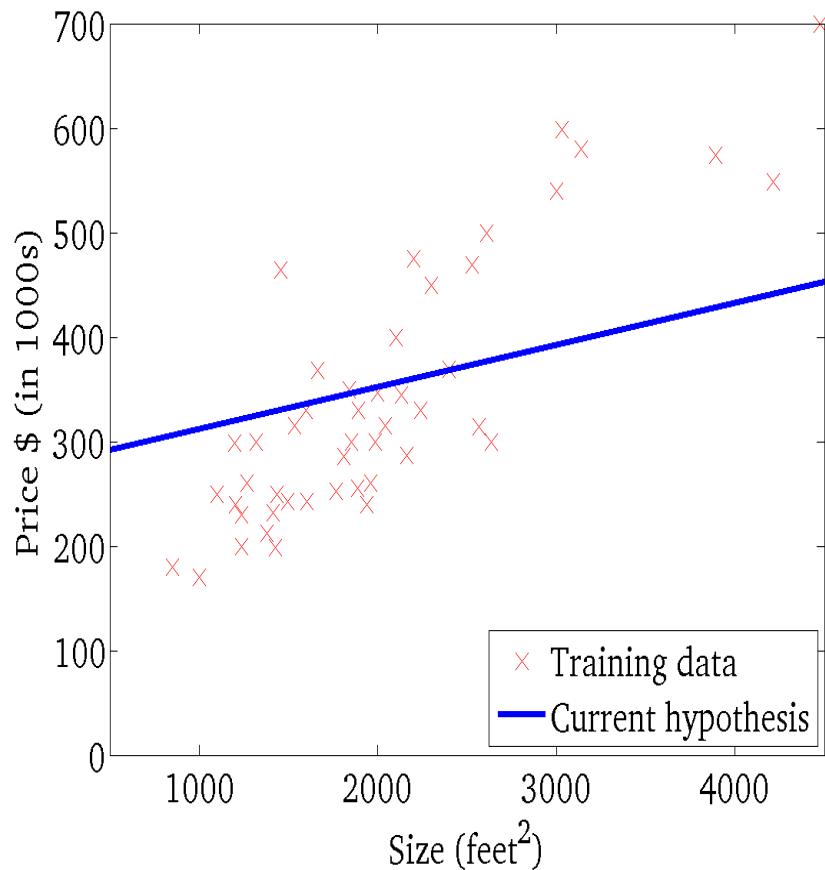
Gradient Descent for LR Price~Size – Iteration 6

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



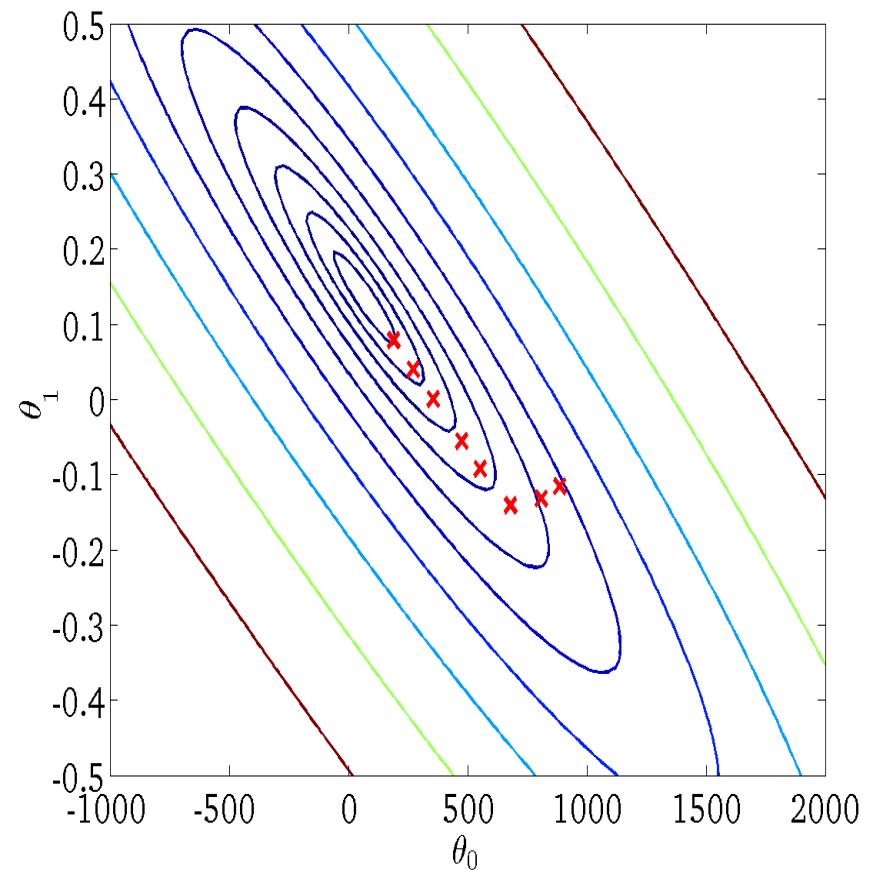
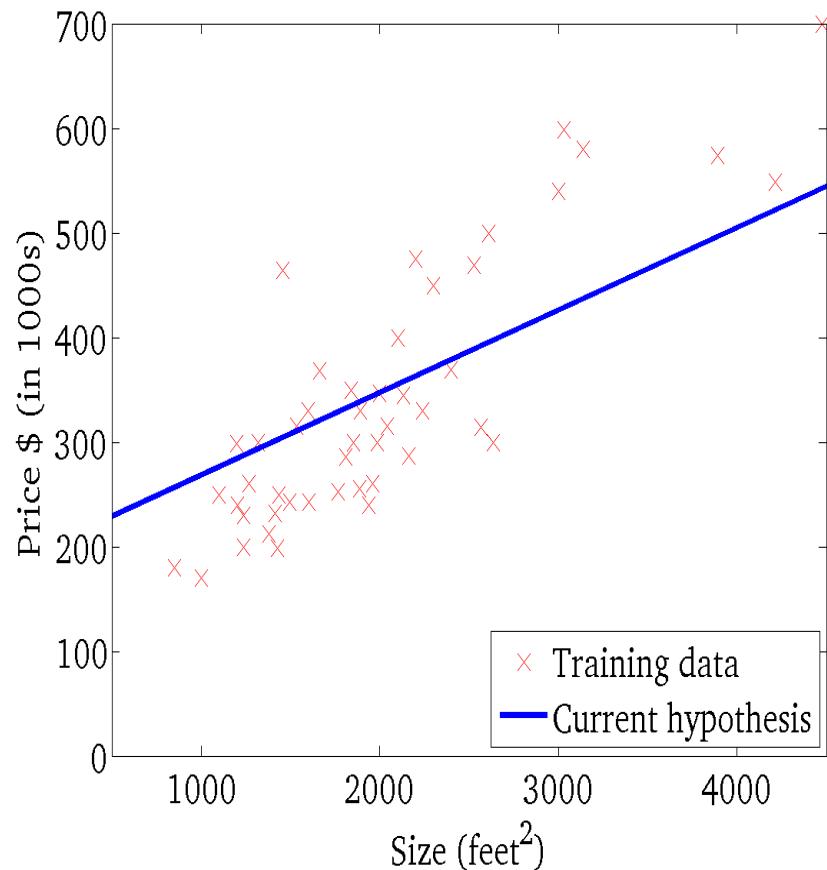
Gradient Descent for LR Price~Size – Iteration 7

$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)

$$J(\theta_0, \theta_1)$$

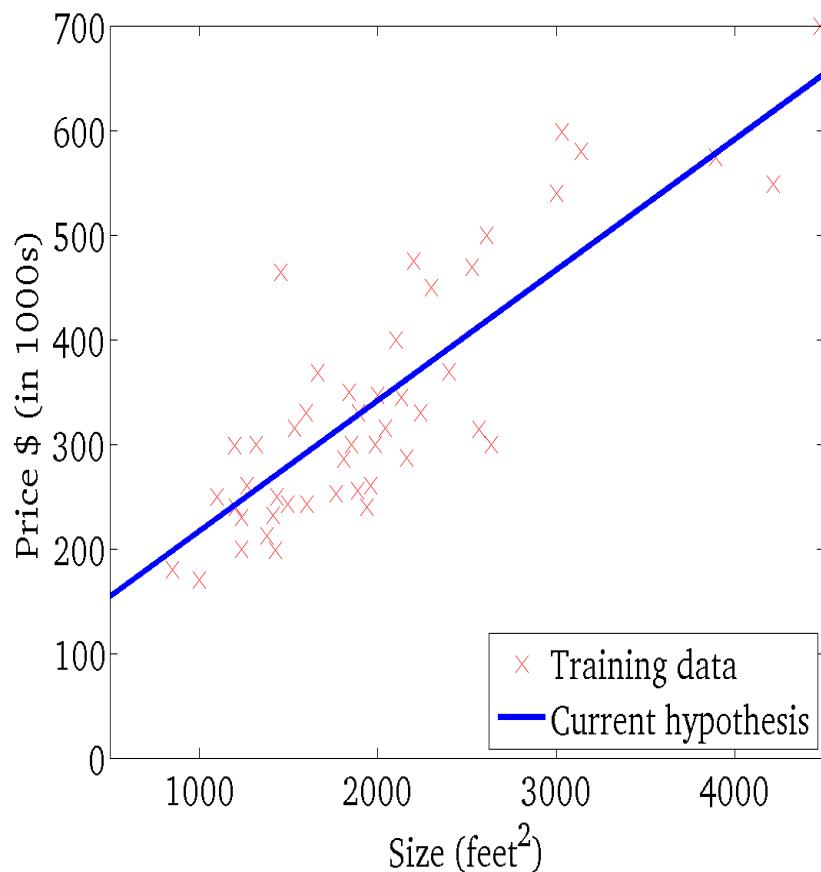
(function of the parameters θ_0, θ_1)



Gradient Descent for LR Price~Size – Converged after 9 steps

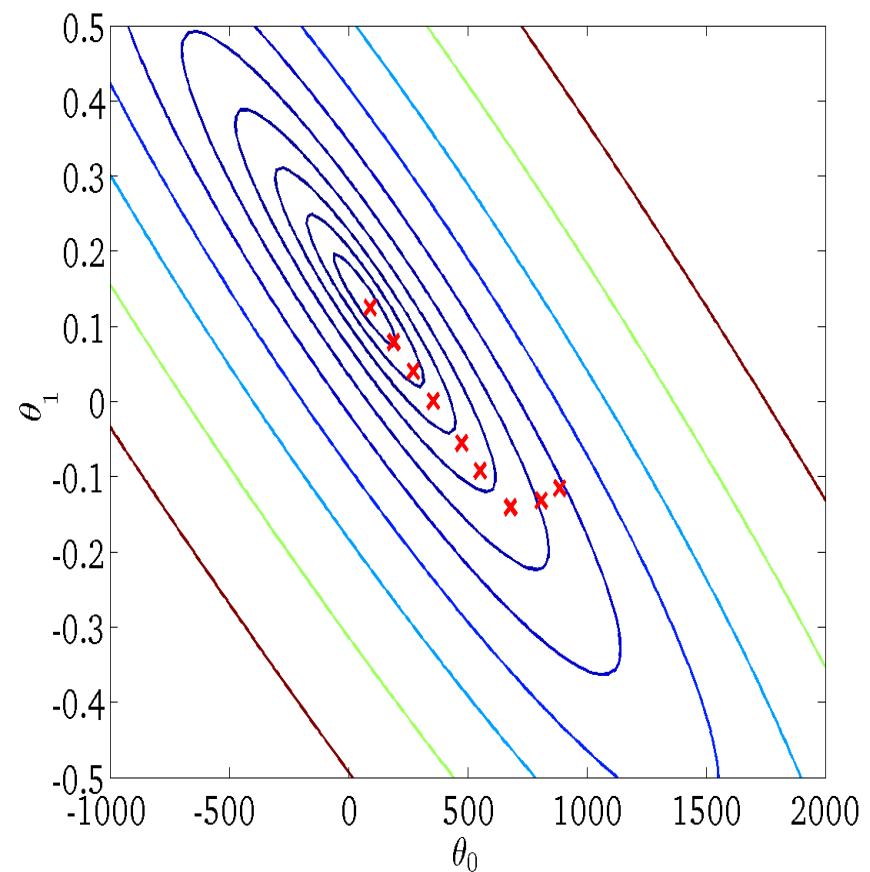
$$h_{\theta}(x)$$

(for fixed θ_0, θ_1 , this is a function of x)



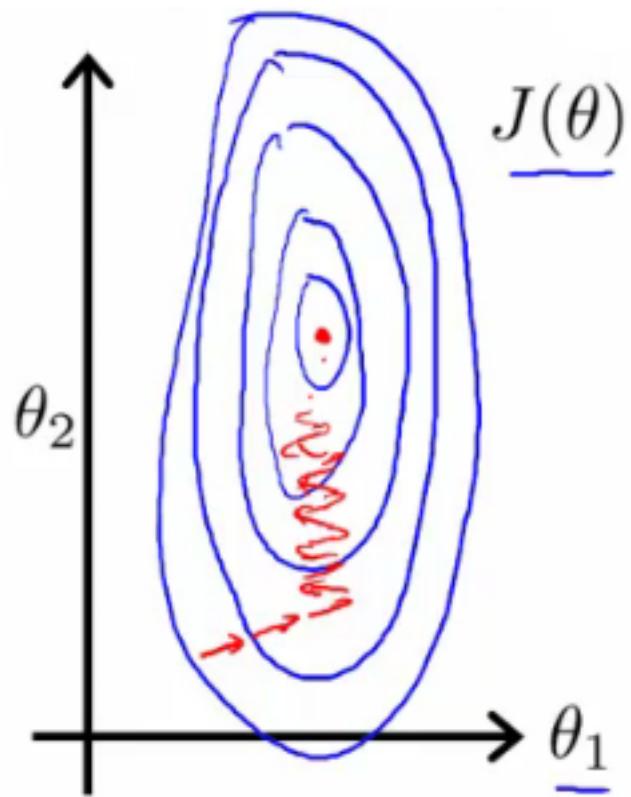
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



Sometimes we overshoot

Alpha our learning rate needs to be small enough to allow us to converge. Otherwise we will oscillate. ...can also decrease ALPHA over time... Gradient descent provides no guarantees to find the minimum but....



$RSS = \text{Variance of } \varepsilon$

$$0 = \frac{\partial \sum \hat{\varepsilon}_i^2}{\partial W} = \frac{\partial \left(\sum_{j=1}^n (X_j W - y_i)^2 \right)}{\partial W}$$

$$\nabla J(W) = \left(\sum_{j=1}^n (X_j W - y_i) X_j \right)$$

$$W^{t+1} = W^t - \alpha^i \left(\sum_{j=1}^n (X_j W^t - y_i) X_j \right)$$

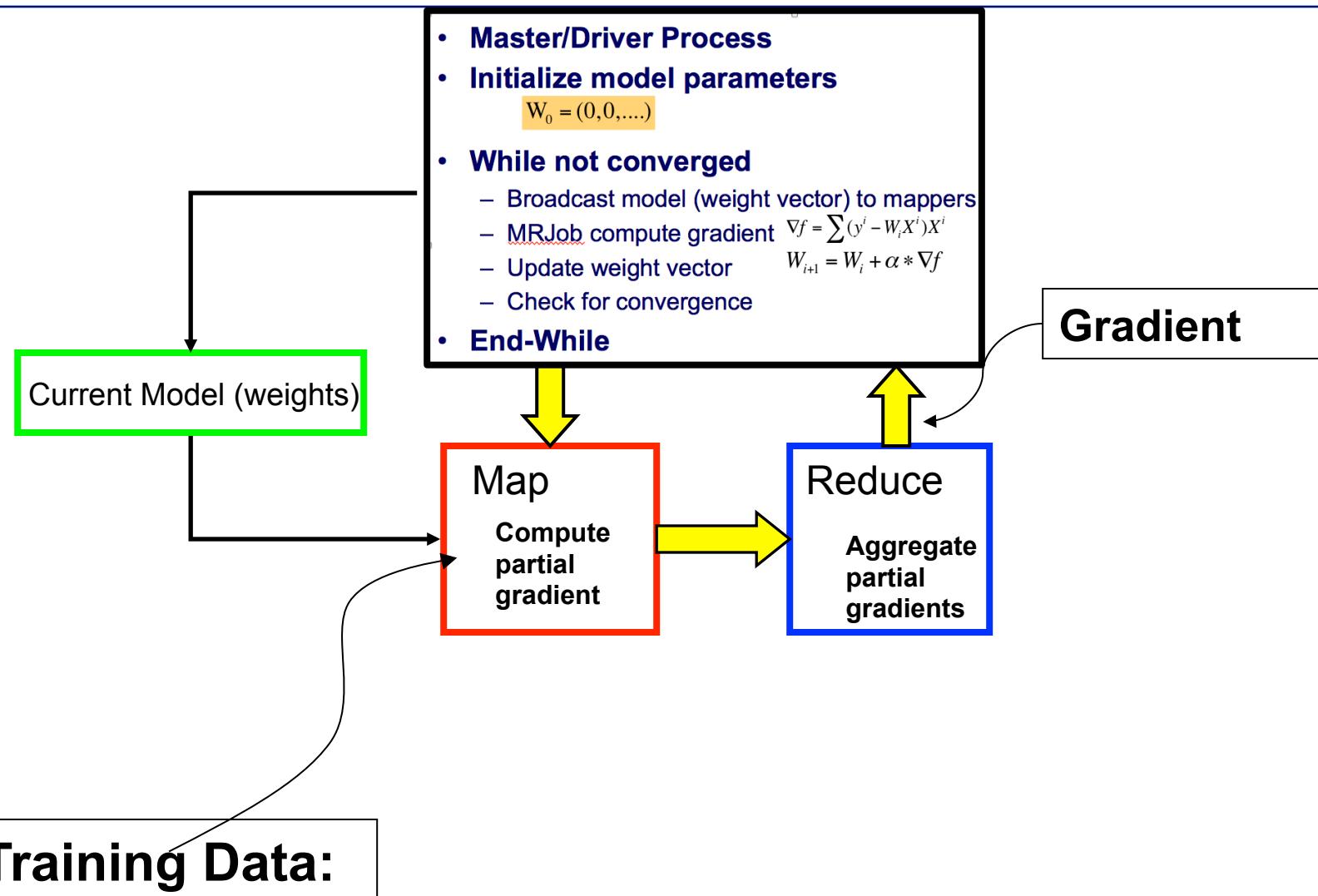
OLS via Distributed Gradient Descent

- **Master/Driver Process**
- **Initialize model parameters**

$$W_0 = (0, 0, \dots)$$

- **While not converged**
 - Broadcast model (weight vector) to mappers
 - MRJob compute gradient $\nabla f = \sum (y^i - W_i X^i) X^i$
 - Update weight vector $W_{i+1} = W_i + \alpha * \nabla f$
 - Check for convergence
- **End-While**

MapReduce Implementation



Linear Regression

Data Generation

```
import numpy as np
import csv
def data_generate(fileName, w=[0,0], size = 100):
    np.random.seed(0)
    x = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 2, size)
    y = (x * w[0] + w[1] + noise)
    data = zip(y, x)
    with open(fileName, 'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True
```

The true $y = 8x - 2$.

```
w = [8, -2]
data_generate('data.csv', w, 100)
```

True

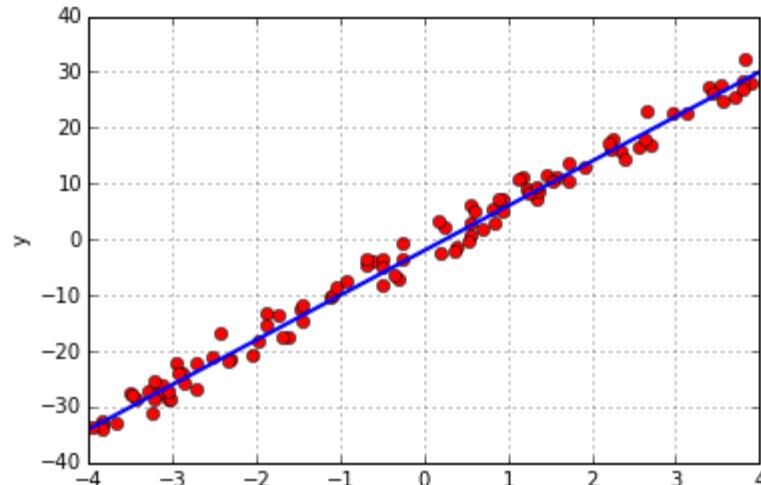
Data format: y, x

```
0.77075643829433071, 2.672964698525508, -0.80725624349150493
8.8154706781482624, 4.9760776507722362, 2.522486096383517
-9.8151480026146789, -0.02985975394819107, -5.9270633829692514
-18.969417561297352, -5.5040670893830468, -8.1264919044925392
10.600223695067733, -6.03874270480752, 9.5105193453227734
0.41965480983655812, 5.2106142439791743, -2.6531234332946063
-3.4052657239448485, -6.6177832687492906, 1.5256793588631936
-16.087150784240489, -8.2332037165197942, -5.0813586409076557
```

Data Visualization

```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    with open(file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[0]), 'o'+r')
    plt.xlabel("x")
    plt.ylabel("y")
    x = [-4, 4]
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x,y, linewidth=2.0)
    plt.grid()
    plt.show()
```

```
dataPlot('data.csv',w)
```



Linear Regression

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i$$

Gradient descent (no regularization)

```
: from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))


def linearRegressionGD(data, wInitial=None, learningRate=0.05, iterations=50):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is Large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b) Sum up gradients
        w = w - learningRate * gradient/n
    return w

: data = sc.textFile('data.csv').map(readPoint).cache()
linearRegressionGD(data)

: array([ 7.98418741, -1.61969498])
```

<https://www.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb?dl=0>

<http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb>

Get gradients for each instance

Gradient descent (no regularization)

```
: from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))

def linearRegressionGD(data, wInitial=None, learningRate=0.05, iterations=50):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        w = w - learningRate * gradient/n
    return w
```

Get gradients for each instance

```
: data = sc.textFile('data.csv').map(readPoint).cache()
linearRegressionGD(data)

: array([ 7.98418741, -1.61969498])
```

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2$$

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i$$

Linear Regression

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i$$

Gradient descent (no regularization)

```
: from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))
```

```
wBroadcast = sc.broadcast(w)
gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
    .reduce(lambda a, b: a + b)
w = w - learningRate * gradient/n
```

```
for i in range(iterations):
    wBroadcast = sc.broadcast(w)
    gradient = data.map(lambda p: -2 * (p.y - np.dot(wBroadcast.value, p.x)) * np.array(p.x)) \
        .reduce(lambda a, b: a + b) Sum up gradients
    w = w - learningRate * gradient/n
return w
```

```
: data = sc.textFile('data.csv').map(readPoint).cache()
```

```
: linearRegressionGD(data)
```

```
: array([ 7.98418741, -1.61969498])
```

<https://www.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb?dl=0>

<http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb>

Get gradients for each instance

Linear Regression

Plot w in iterations

```
def ierationsPlot(fileName, truew):
    x = [-4, 4]

    w = truew
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'b', label="True line", linewidth=4.0)

    np.random.seed(400)
    w = np.random.normal(0,1,2)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'r--', label="After 0 Iterations", linewidth=2.0)

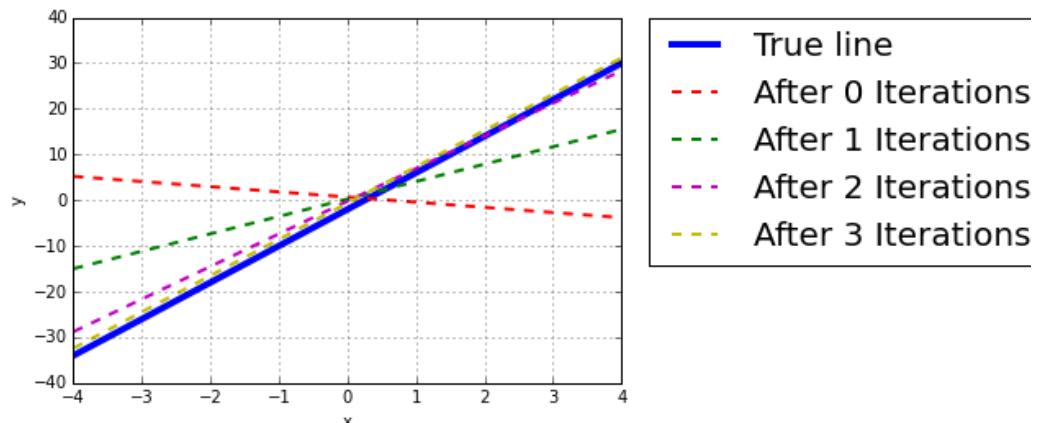
    data = sc.textFile(fileName).map(readPoint).cache()
    w = linearRegressionGD(data, w, iterations=1)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'g--', label="After 1 Iterations", linewidth=2.0)

    w = linearRegressionGD(data, w, iterations=2)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'm--', label="After 2 Iterations", linewidth=2.0)

    w = linearRegressionGD(data, w, iterations=3)
    y = [(i * w[0] + w[1]) for i in x]
    plt.plot(x, y, 'y--', label="After 3 Iterations", linewidth=2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxespad=0.)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    plt.show()
```

```
ierationsPlot('data.csv',w)
```



- True line
- - - After 0 Iterations
- - - After 1 Iterations
- - - After 2 Iterations
- - - After 3 Iterations

namedtuple

The standard `tuple` uses numerical indexes to access its members.

```
bob = ('Bob', 30, 'male')
print 'Representation:', bob

jane = ('Jane', 29, 'female')
print '\nField by index:', jane[0]

print '\nFields by index:'
for p in [ bob, jane ]:
    print '%s is a %d year old %s' % p
```

This makes `tuples` convenient containers for simple uses.

```
$ python collections_tuple.py

Representation: ('Bob', 30, 'male')

Field by index: Jane

Fields by index:
Bob is a 30 year old male
Jane is a 29 year old female
```

Examples as namedTuple

```
: Point = namedtuple('Point', 'x y')  #Point record with fields x and y, where x will be a array of 2 real numbers
xx= Point([1.2,2.3], 1)
print xx[0], xx[1]
print xx.x, xx.y

[1.2, 2.3] 1
[1.2, 2.3] 1
```

Challenge: Extend OLS to Regularized OLS

- Challenge: Extend OLS to Regularized OLS
- Code up Lasso and Ridge based linear regression

<https://www.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb?dl=0>

<http://nbviewer.ipython.org/urls/dl.dropbox.com/s/9k6t4q9ew3nl1lh/LinearRegression-Notebook.ipynb>

(1) One Norm $\|\vec{v}\|_1$

The one-norm (also known as the L_1 -norm, ℓ_1 norm, or mean norm) of a vector \vec{v} is denoted $\|\vec{v}\|_1$ and is defined as the sum of the absolute values of its components:

$$\|\vec{v}\|_1 = \sum_{i=1}^n |v_i| \quad (1)$$

for example, given the vector $\vec{v} = (1, -4, 5)$, we calculate the one-norm:

$$\|(1, -4, 5)\|_1 = |1| + |-4| + |5| = 10$$

(2) Two Norm $\|\vec{v}\|_2$

The two-norm (also known as the L_2 -norm, ℓ_2 -norm, mean-square norm, or least-squares norm) of a vector \vec{v} is denoted $\|\vec{v}\|_2$ and is defined as the square root of the sum of the squares of the absolute values of its components:

$$\|\vec{v}\|_2 = \sqrt{\sum_{i=1}^n |v_i|^2} \quad (2)$$

for example, given the vector $\vec{v} = (1, -4, 5)$, we calculate the two-norm:

$$\|(1, -4, 5)\|_2 = \sqrt{|1|^2 + |-4|^2 + |5|^2} = \sqrt{42}$$

Ridge and Lasso Regression

Linear Regression

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2$$
$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i$$

Ridge Regression (L2 regularization)

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot w^T \cdot w$$

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda w$$

Lasso Regression (L1 regularization)

Objective Function

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot |w|$$

Vector of size of W of -1 or +1

Gradient Descent

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda \cdot \text{sgn}(w)$$

Hints

- **vector of sign()**

```
In [7]: np.sign(wReg)  
Out[7]: array([ 1, -1,  1, -1,  0])
```

```
In [7]: np.sign(wReg)  
Out[7]: array([ 1, -1,  1, -1,  0])
```

```
In [9]: wReg.shape[0]  
Out[9]: 5
```

```
In [ ]:
```

Regularized Ridge Linear Regression

```
:def linearRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType='Ridge'):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y-np.dot(wBroadcast.value,p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term,
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot w^T \cdot w$$
$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda w$$

In [7]: `np.sign(wReg)`

Out[7]: `array([1, -1, 1, -1, 0])`

Requires a simple modification for ridge and lasso (as those terms depend on the weight vector only. So no need for mappers/reducers etc.!!

Regularized Ridge Linear Regression

```
:def linearRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType='Ridge'):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y-np.dot(wBroadcast.value,p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

$$\min_w L(w) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i)^2 + \lambda \cdot |w|$$

Lasso

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=0}^{n-1} (y_i - w^T \cdot x_i) \cdot x_i + \lambda \cdot \text{sgn}(w)$$

Regularized Linear Regression

```
def linearRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: -2 * (p.y-np.dot(wBroadcast.value,p.x)) * np.array(p.x)) \
                    .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

Ridge Regression

```
np.random.seed(400)
linearRegressionGDReg(data, iterations=50, regParam=0.1, regType="Ridge")
array([ 7.98398871, -1.60876051])
```

Lasso Regression

```
np.random.seed(400)
linearRegressionGDReg(data, iterations=50, regParam=0.1, regType="Lasso")
array([ 7.98466543, -1.60811709])
```

Part 3: Machine Learning in Spark

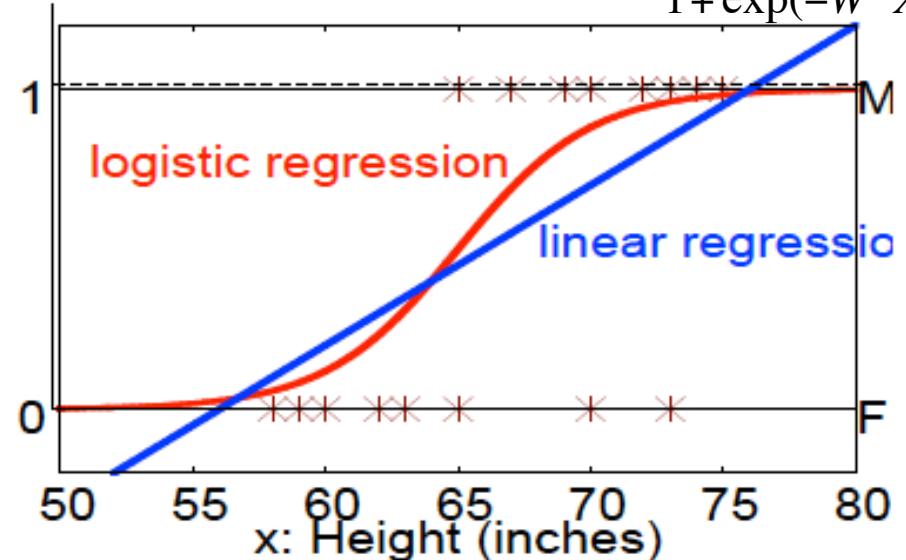
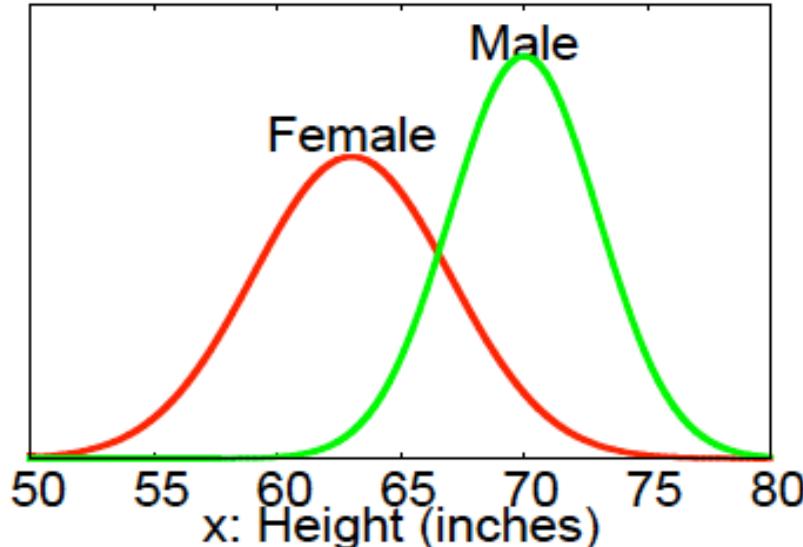
- **Data Frames**
- **MLLib**
- **Write your own algos**
 - Linear regression (Ridge and Lasso)
 - Logistic Regression
- **Pipelines**
- **R and Spark**
 - Linear Regression example
- **Graphs in Spark**
- **Case studies**
 - Flight delay
 - Mobile advertising
 - Social Networks

• Logistic regression

Logistic Regression in One Slide

Example: Predict the *gender* ($y=M/F$) of a person given their *height* ($x=a$ number).

$$\hat{p} = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1 + \exp(-W^T X)}$$



In logReg
Target = $\ln(p/(1-p))$

$$p(y = M | x) = \beta_0 + \beta x \quad (\text{linear regression})$$

Large S

$$\ln \frac{p(y = M | x)}{p(y = F | x)} = \beta_0 + \beta x \quad (\text{logistic regression}) \quad 7$$

Maximum Likelihood Estimator for Logistic Regression

$$P(y = 1 | X) = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1 + \exp(-W^T X)}$$

Cost Function

- Model output

$$\hat{y}_k = P(y_k = 1 | \varphi_k)$$

- Likelihood contribution

$$l_{\theta,k} = \hat{y}_k^{y_k} (1 - \hat{y}_k)^{1-y_k}$$

- Likelihood function

$$L_\theta = \prod_{k=1}^N l_{\theta,k}$$

- Log-likelihood function

$$\ln L_\theta = \sum_{k=1}^N (y_k \ln \hat{y}_k + (1 - y_k) \ln(1 - \hat{y}_k))$$

Maximum Likelihood Criterion

$$\max_{\theta} \ln L_\theta \Leftrightarrow \min_{\theta} -2 \ln L_\theta$$

Logistic Regression assumes a parametric form for $P(Y|X)$

$$P(y = 1 | X) = \text{logistic}(W^T X) = \text{logit}^{-1}(W^T X) = \frac{1}{1 + \exp(-W^T X)}$$

Logistic Regression assumes a parametric form for the distribution $P(Y|X)$, then directly estimates its parameters from the training data. The parametric model assumed by Logistic Regression in the case where Y is boolean is:

$$P(Y = 1 | X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (16)$$

and

$$P(Y = 0 | X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)} \quad (17)$$

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

Loss term in our loss function

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Estimating Parameters using Gradient Descent

- Unfortunately, there is no closed form solution to maximizing $I(W)$ with respect to W .
 - Therefore, one common approach is to use gradient ascent, in which we work with the gradient, which is the vector of partial derivatives.
 - The i th component of the vector gradient has the form

$$l(W) = \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Beginning with initial weights of zero, we repeatedly update the weights in the direction of the gradient, changing the *i*th weight according to *this formula*, where η is a small constant (e.g., 0.01) which determines the step size.

Effectively we are pulling weight vector closer to the examples where we make mistakes

Logistic Regression: Gradient Descent

$$y = \{-1, 1\}$$

- **Objective Function (Logloss):**

$$\arg \min_w L(w) = \operatorname{argmin}_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i})$$

- **Gradient**

$$\nabla w = \frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i$$

- **Update w till it converges**

Gradient Descent

$$w = w - \eta \cdot \nabla w$$

LogReg Regularization: Constrained optimization

- Add regularization to prevent over-fitting
- Lasso
 - L1 Norm regularization: $\|\mathbf{w}\|_1$
- Ridge
 - L2 Norm regularization: $\|\mathbf{w}\|_2^2$
- shrinkage of \mathbf{w}

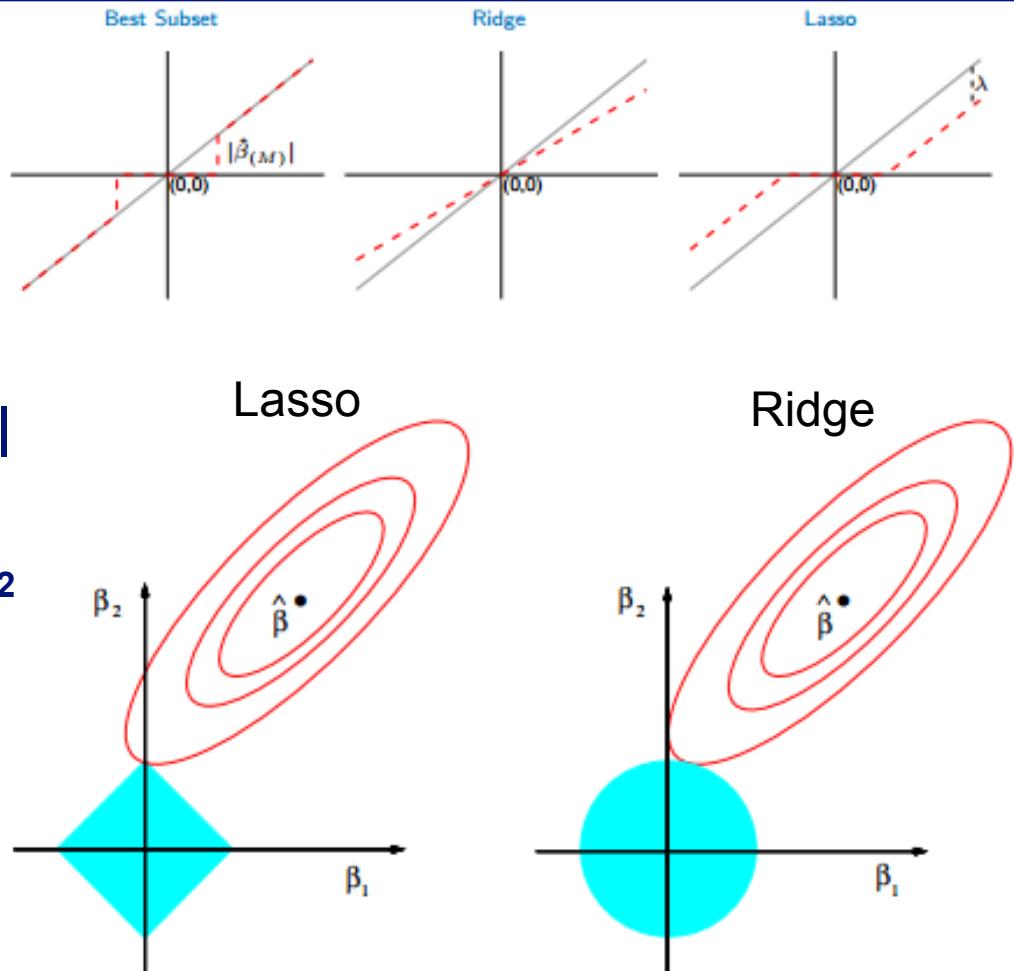


FIGURE 3.11. Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.

From The Elements of Statistical Learning

Logistic Regression

Data Generation

```
import numpy as np
import csv
def data_generate(fileName, w=[0,0,0], size = 100):
    size = 100
    np.random.seed(0)
    x1 = np.random.uniform(-4, 4, size)
    x2 = np.random.uniform(-4, 4, size)
    noise = np.random.normal(0, 3, size)
    v = (x1 * w[0] + x2 * w[1] + w[2] + noise)
    y = (v>0) * 2-1
    data = zip(y, x1, x2)
    with open(fileName,'wb') as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)
    return True

w = np.array([8, -3, -1])
data_generate('data.csv', w, 100)
```

True

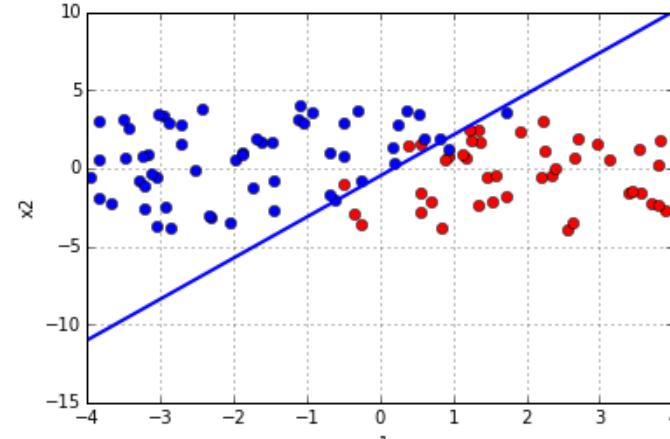
Data format: y, x1,x2

```
1,-6.886661137329684879e-01
-1,1.756760849652599932e+00
-1,7.656676365450297839e-01
-1,3.740612089503772886e+00
```

Data Visualization

```
%matplotlib inline
import matplotlib.pyplot as plt
def dataPlot(file, w):
    cols = {'1': 'r', '-1': 'b'}
    with open(file,'r') as f:
        reader = csv.reader(f)
        for row in reader:
            plt.plot(float(row[1]), float(row[2]), cols[row[0]]+'o')
    plt.xlabel("x1")
    plt.ylabel("x2")
    x1 = [-4,4]
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, linewidth=2.0)
    plt.grid()
    plt.show()

dataPlot('data.csv',w)
```



Logistic Regression

Objective Function

$$\arg \min_w L(w) = \operatorname{argmin}_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i}) \quad y = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i$$

Gradient descent (no regularization)

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))

def logisticRegressionGD(data, wInitial=None, learningRate=0.05, iterations=100):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen)
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y * np.dot(wBroadcast.value, p.x))) - 1) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b) Sum up gradients
        w = w - learningRate * gradient / n
    #w = w / np.linalg.norm(w) #normalization
    return w

data = sc.textFile('data.csv').map(readPoint).cache()
logisticRegressionGD(data)
```

Get gradients for each instance

Logistic Regression

Objective Function

$$\arg \min_w L(w) = \operatorname{argmin}_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i}) \quad y = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i$$

Gradient descent (no regularization)

Error

```
from collections import namedtuple
import numpy as np
Point = namedtuple('Point', 'x y')

def readPoint(line):
    d = line.split(',')
    x = [float(i) for i in d[1:]]
    x.append(1.0) #bias term
    return Point(x, float(d[0]))

def logisticRegressionGD(data, wInitial=None, learningRate=0.05, iterations=100):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen)
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)

        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y * np.dot(wBroadcast.value, p.x)))) - 1) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b) Get gradients for each instance
Sum gradients for each instance

    return w

data = sc.textFile('data.csv').map(readPoint).cache()
logisticRegressionGD(data)

Let's print the result:
array([ 1.35420091, -0.46407845, -1.01559616])
```

Logistic Regression

Plot w in iterations

```
def ierationsPlot(fileName, truew):
    x1 = [-4, 4]

    w = truew
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'b', label="True line", linewidth=4.0)

    np.random.seed(800)
    w = np.random.normal(0,1,3)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'r--', label="After 0 Iterations", linewidth=2.0)

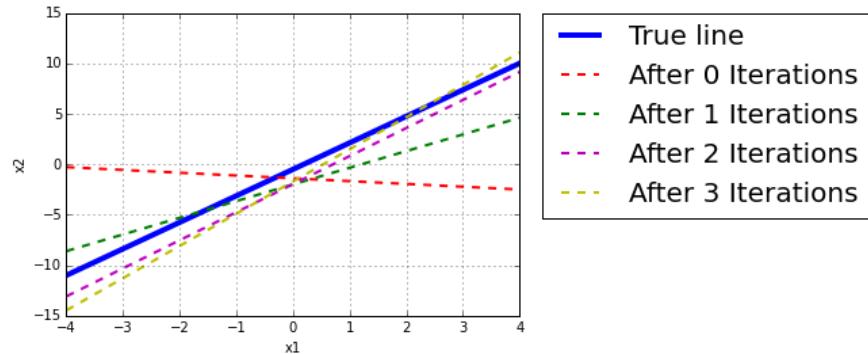
    data = sc.textFile(fileName).map(readPoint).cache()
    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'g--', label="After 1 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'm--', label="After 2 Iterations", linewidth=2.0)

    w = logisticRegressionGD(data, w, iterations=30)
    x2 = [-(i * w[0] + w[2]) / w[1] for i in x1]
    plt.plot(x1, x2, 'y--', label="After 3 Iterations", linewidth=2.0)

    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, fontsize=20, borderaxespad=0.)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.grid()
    plt.show()

ierationsPlot('data.csv',w)
```



Logistic Regression vs Ridge and Lasso

Objective Function

$$\arg \min_w L(w) = \operatorname{argmin}_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i}) \quad y = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i$$

- **Ridge (L2 regularization)**

Objective Function

$$\arg \min_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot w^T \cdot w \quad y = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i + \lambda w$$

- **Lasso(L1 regularization)**

Objective Function

$$\arg \min_w \frac{1}{n} \sum_{i=0}^{n-1} \log(1 + e^{-y_i \cdot w^T \cdot x_i}) + \lambda \cdot |w| \quad y = \{-1, 1\}$$

Gradient Descent

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{1}{1 + e^{-y_i \cdot w^T \cdot x_i}} - 1 \right) \cdot y_i \cdot x_i + \lambda \cdot \operatorname{sgn}(w)$$

Logistic regression

Gradient descent (regularization)

```
def logisticRegressionGDReg(data, wInitial=None, learningRate=0.05, iterations=50, regParam=0.01, regType=None):
    featureLen = len(data.take(1)[0].x)
    n = data.count()
    if wInitial is None:
        w = np.random.normal(size=featureLen) # w should be broadcasted if it is large
    else:
        w = wInitial
    for i in range(iterations):
        wBroadcast = sc.broadcast(w)
        gradient = data.map(lambda p: (1 / (1 + np.exp(-p.y*np.dot(wBroadcast.value, p.x))))-1) * p.y * np.array(p.x))\
            .reduce(lambda a, b: a + b)
        if regType == "Ridge":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
        elif regType == "Lasso":
            wReg = w * 1
            wReg[-1] = 0 #last value of weight vector is bias term, ignored in regularization
            wReg = (wReg>0).astype(int) * 2-1
        else:
            wReg = np.zeros(w.shape[0])
        gradient = gradient + regParam * wReg #gradient: GD of Squared Error+ GD of regularized term
        w = w - learningRate * gradient / n
    return w
```

Ridge Regression

```
np.random.seed(400)
logisticRegressionGDReg(data, iterations=50, regParam=0.1, regType="Ridge")
array([ 0.74835721, -0.17134752, -0.39953122])
```

L Lasso Regression

Contact:James.Shanahan@gmail.com 636

Broadcast variables

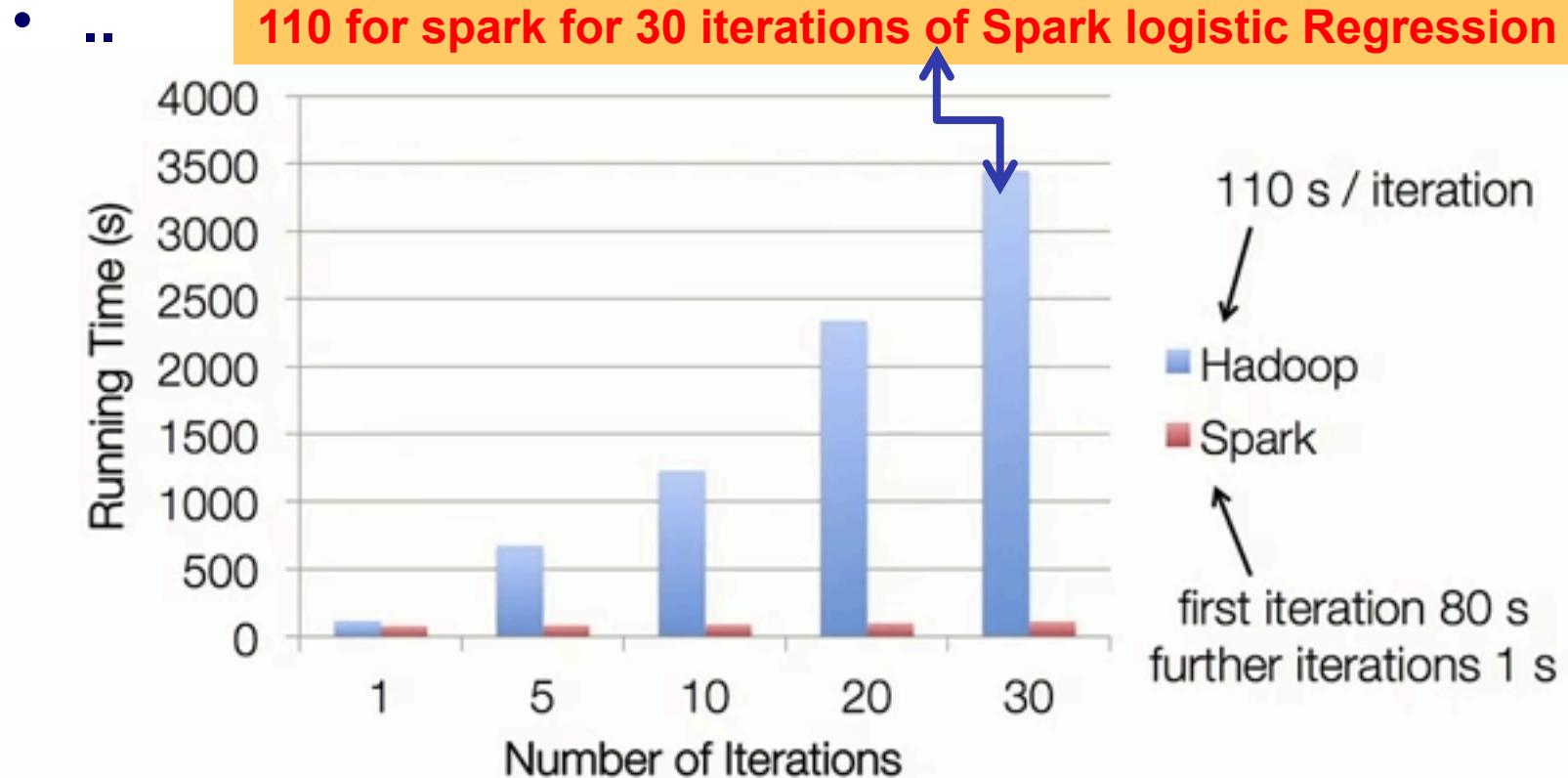
- Improve this using broadcast variables
 - Each iteration
 - Create a new RDD (with a DAG tree; map+Reduce that are shipped to the workers)
-
- Fault tolerant, distributed, scaleable gradient descent

Separable Updates

Can be generalized for

- » Unconstrained optimization
- » Smooth or non-smooth
- » LBFGS, Conjugate Gradient, Accelerated Gradient methods, ...

Logistic Regression: Hadoop vs Spark

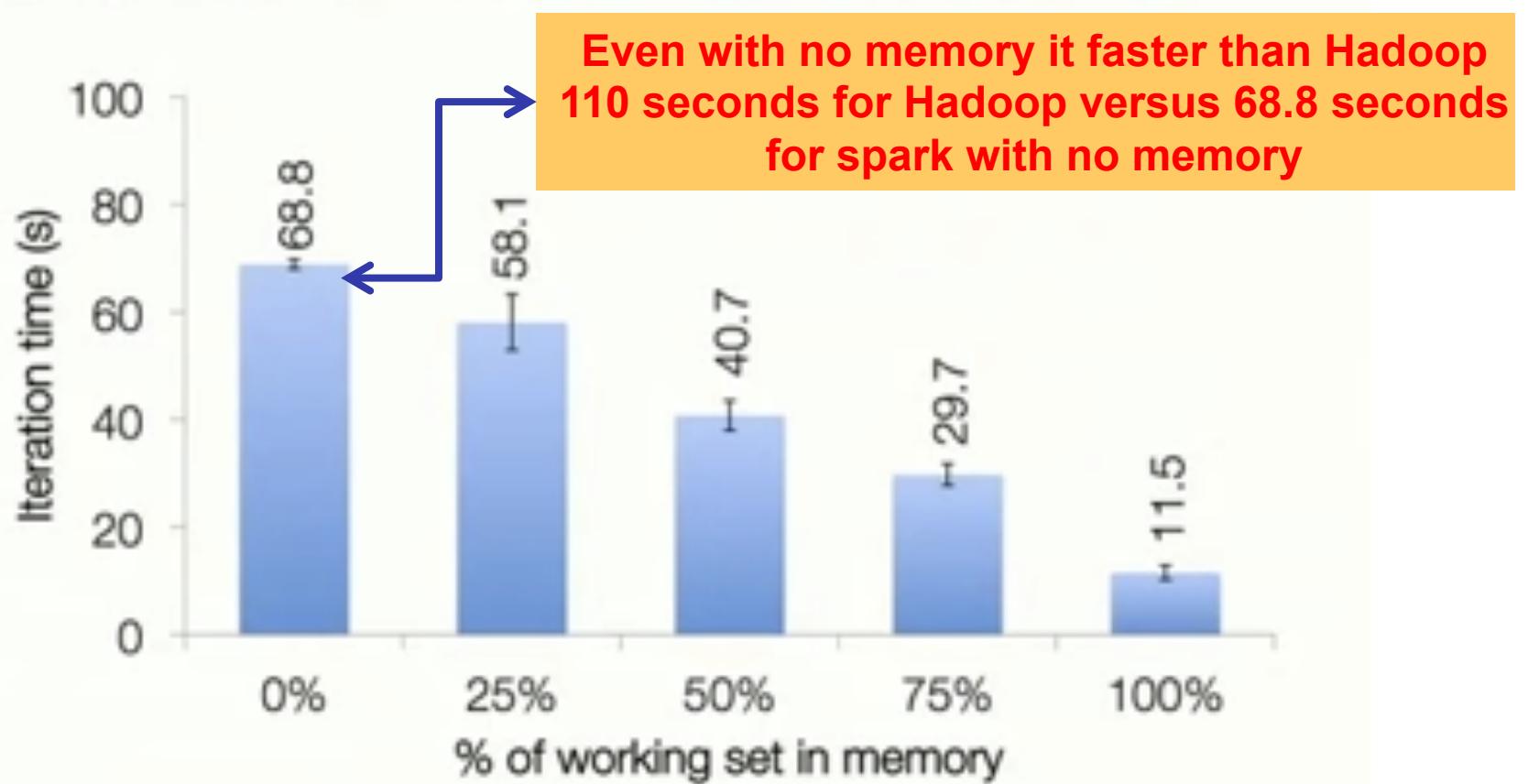


100 GB of data on 50 m1.xlarge EC2 machines

[From Reza Zadeh, Spark Summit 2015]

If the working set does not fit in memory

Behavior with Less RAM



[From Reza Zadeh, Spark Summit 2015]

- **TODO SVMs Section**

- Perceptron
- SVMs

- **Clustering**

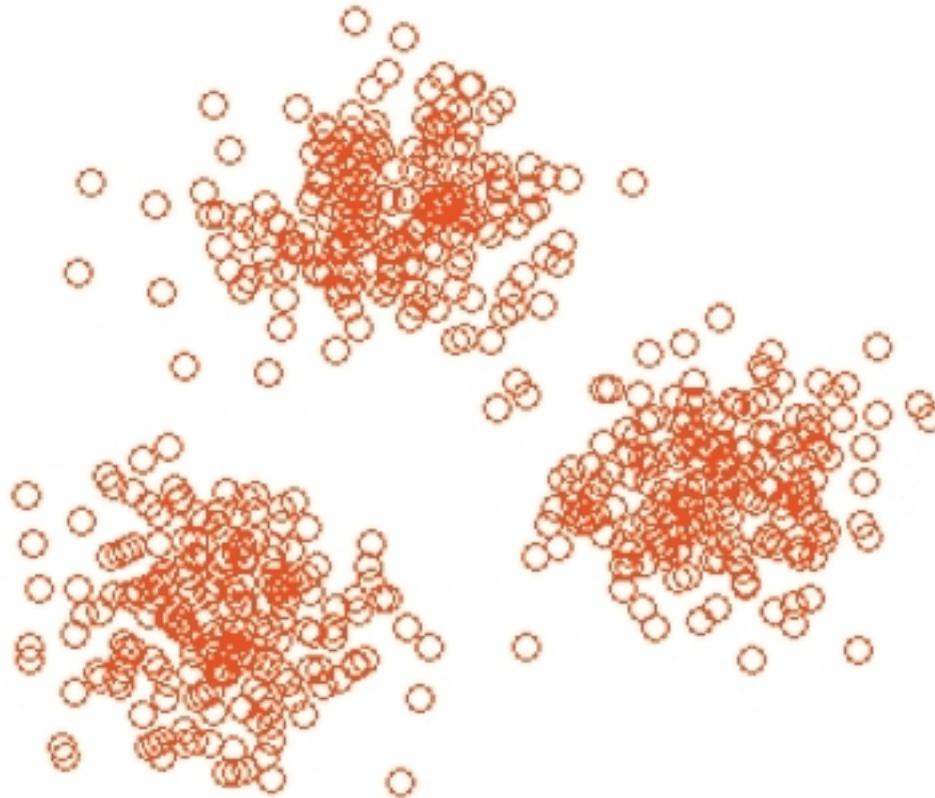
Clustering

- **Clustering is a data mining (machine learning) technique used to place data elements into related groups without advance knowledge of the group definitions.**
 - Clustering is an unsupervised learning activity in that we do not have labels associated
 - Clustering is a divide and conquer strategy to thinking about our world of examples
 - Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).
- **Applications in EDA, machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.**

Hard vs. soft clustering

- Hard clustering: Each object (say a document) belongs to exactly one cluster
 - More common and easier to do
- Soft clustering: A object (say a document) can belong to more than one cluster.
 - Makes more sense for applications like creating browsable hierarchies
 - You may want to put a pair of sneakers in two clusters: (i) sports apparel and (ii) shoes
 - You can only do that with a soft clustering approach.
- See book IIR Book Section 16.5, 18

Clustering Algorithms



- Clustering is a data mining/machine learning algorithm used to cluster observations into groups of related observations without any prior knowledge of those relationships
 - Try to locate homogenous groups of observations
 - Clustering algorithms try to formalize this

distance metric is more important than the Clustering Algorithm

- Attach label to each observation or data points in a set
- You can say this “unsupervised classification”
- Intuitively, if you would want to assign same label to a data points that are “close” to each other
- Thus, clustering algorithms rely on a distance metric between data points
- Sometimes, it is said that the for clustering, the distance metric is more important than the clustering algorithm

Distances: Quantitative Variables

Some examples

Identity (absolute) error

$$d_j(x_{ij}, x_{i'j}) = I(x_{ij} \neq x_{i'j})$$

Data point:

$$x_i = [x_{i1} \dots x_{ip}]^T$$

Squared distance

$$d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$$

L_q norms

$$L_{qii'} = \left[\sum_i |x_{ij} - x_{i'j}|^q \right]^{1/q}$$

Canberra distance

$$d_{ii'} = \sum_i \frac{|x_{ij} - x_{i'j}|}{|x_{ii} + x_{i'i}|}$$

Correlation

$$\rho(x_i, x_{i'}) = \frac{\sum_j (x_{ij} - \bar{x}_i)(x_{i'j} - \bar{x}_{i'})}{\sqrt{\sum_j (x_{ij} - \bar{x}_i)^2 \sum_j (x_{i'j} - \bar{x}_{i'})^2}}$$

Distances: Ordinal and Categorical Variables

- **Ordinal variables can be forced to lie within (0, 1) and then a quantitative metric can be applied:**

$$\frac{k - 1/2}{M}, k = 1, 2, \dots, M$$

- **For categorical variables, distances must be specified by user between each pair of categories**
 - E.g., Distance in a hierarchy of categories

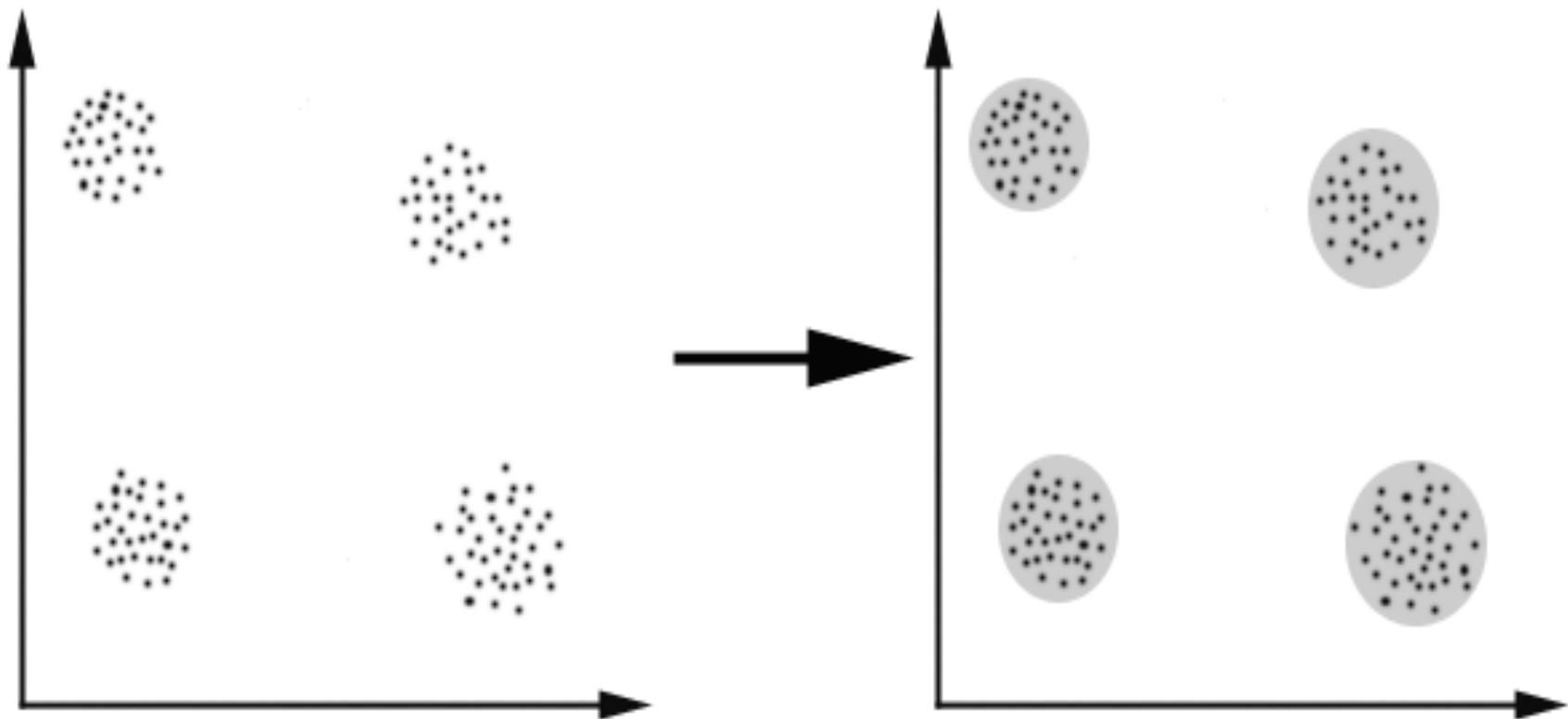
Combining Distances

- Often weighted sum is used:

$$D(x_i, x_j) = \sum_{l=1}^p w_l d(x_{il}, x_{jl}), \quad \sum_{l=1}^p w_l = 1, \quad w_l > 0.$$

Clustering Algorithms

- Here we have 4 homogenous groups (clusters) and the similarity criterion is distance



Clustering Algorithms

- **Euclidean Distance**

$$dist = \sqrt{\sum_{k=1}^n (p_k - q_k)^2}$$

- Where n is the number of dimensions (attributes) and pk and qk are, respectively, the kth attributes (components) or data objects p and q.

Clustering Algorithms

- Minkowski Distance is a generalization of Euclidean Distance

$$dist = \left(\sum_{k=1}^n |p_k - q_k|^r \right)^{\frac{1}{r}}$$

Where r is a parameter, n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k th attributes (components) or data objects p and q .

Clustering Algorithms

- **Non-Hierarchical Clustering**
 - Exclusive Clustering (k-means) [In R, cluster package]
 - Overlapping Clustering (fuzzy c means) [In R, e1071]
 - Probabilistic Clustering (Mixture of Gaussians)
- **Hierarchical Clustering**
 - Agglomerative approach (bottom-up)
 - In R: hclust() and agnes()
 - 2. Divisive approach (top-down)
 - In R: diana()

Clustering Algorithms

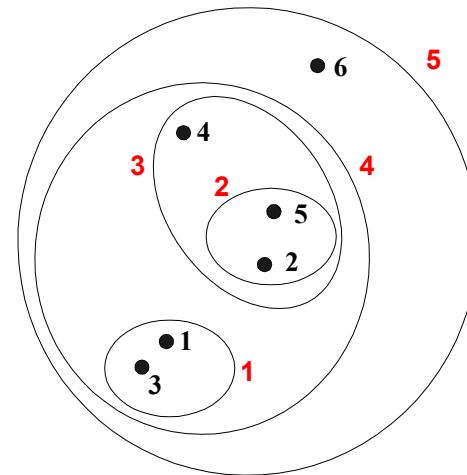
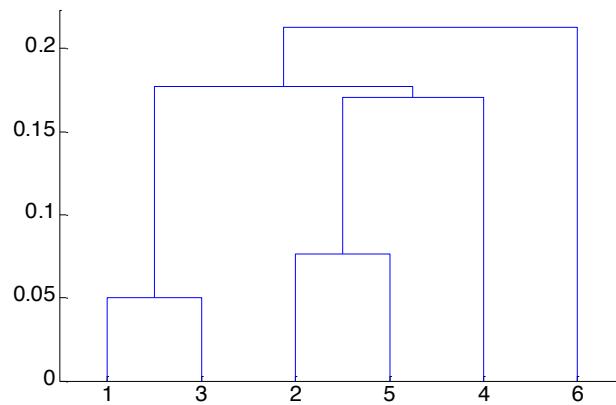
- **Non-Hierarchical Clustering**
 - Exclusive Clustering (k-means) [In R, cluster package]
 - Overlapping Clustering (fuzzy c means) [In R, e1071]
 - Probabilistic Clustering (Mixture of Gaussians)
- **Hierarchical Clustering**
 - Agglomerative approach (bottom-up)
 - In R: hclust() and agnes()
 - 2. Divisive approach (top-down)
 - In R: diana()

Question: can both types of algorithm be effectively implemented in Hadoop?

Clustering Algorithms

- **Hierarchical Clustering**

- Produces a set of *nested clusters* organized as a hierarchical tree
- Can be visualized as a **dendrogram**
 - A tree-like diagram that records the sequences of merges or splits

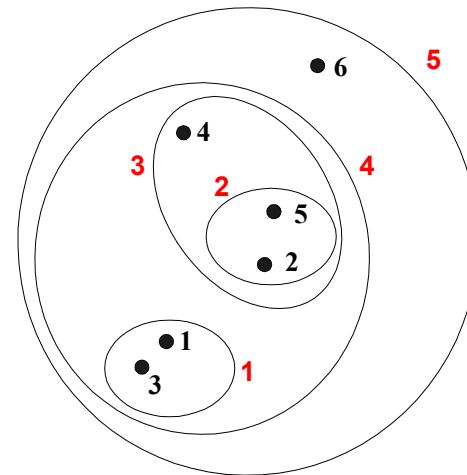
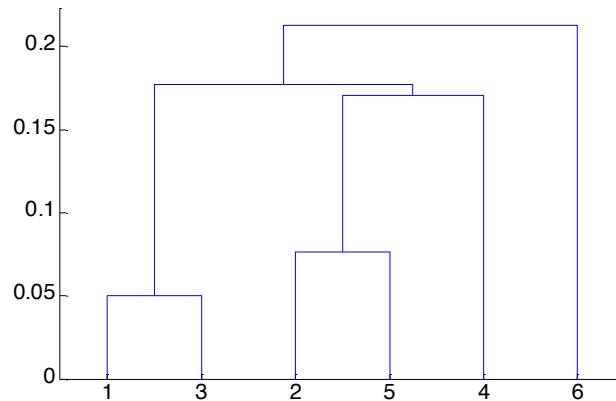


Clustering Algorithms

- Two main types of hierarchical clustering
 - **Agglomerative:**
 - Start with the points as individual clusters
 - At each step, merge the closest pair of clusters until only one cluster (or **k** clusters) left
 - **Divisive:**
 - Start with one, all-inclusive cluster
 - At each step, split a cluster until each cluster contains a point (or there are **k** clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
 - Merge or split one cluster at a time

Clustering Algorithms

- **Strengths of Hierarchical Clustering**
 - No assumptions on the number of clusters
 - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level
 - Hierarchical clusterings may correspond to meaningful taxonomies
 - Example in biological sciences (e.g., phylogeny reconstruction, etc), web (e.g., product catalogs) etc



Clustering Algorithms

- 4.1 Background and Motivation (5)
- 4.2 MrJob
 - Installation (5)
 - BLT Install MrJob and verify [5]
 - MrJob Fundamentals and Concepts (5)
 - Writing Mrjob code (10)
 - BLT Join **operation** (15)
 - Log file processing (10)
 - BLT: most frequently visited pages [15]
 - Oyster: Thought experiment: bad logs [5]
 - BLT Challenge: Most frequently visited titles (15)
 - Serializable, JSON and MrJob benchmarks (15) [75]
 - MrJob benchmark study(5) (60)
- 4.4 Clustering Algorithms [35]
 - Clustering overview (10 mins)
 - Kmeans algorithm (5)
 - Distributed Kmeans in MrJob [ScreenFlow] (15 minutes)
 - Initialization (Canopy Clustering)

Clustering Algorithms

- **Flat/Non-Hierarchical Clustering**
 - Exclusive Clustering (k-means) [In R, cluster package]
 - Overlapping Clustering (fuzzy c means) [In R, e1071]
 - Probabilistic Clustering (Mixture of Gaussians)
- **Hierarchical Clustering**
 - Agglomerative approach (bottom-up)
 - In R: hclust() and agnes()
 - 2. Divisive approach (top-down)
 - In R: diana()

Kmeans Algorithm

- **k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.**
 - This results in a partitioning of the data space into Voronoi cells.
- **The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum.**
- **These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms.**

K-Means

- Partition data such that it minimizes the within group sum of squares over all variables
- N examples with k clusters
- Impractical to explore all possible partitions

The k -means clustering technique seeks to partition a set of data into a specified number of groups, k , by minimising some numerical criterion, low values of which are considered indicative of a ‘good’ solution. The most commonly used approach, for example, is to try to find the partition of the n individuals into k groups, which minimises the within-group sum of squares over all variables. The problem then appears relatively simple; namely, consider every possible partition of the n individuals into k groups, and select the one with the lowest within-group sum of squares. Unfortunately, the problem in practise is not so straightforward. The numbers involved are so vast that complete enumeration of every possible partition remains impossible even with the fastest computer. The scale of the problem is illustrated by the numbers in Table 18.3.

Table 18.3: Number of possible partitions depending on the sample size n and number of clusters k .

n	k	Number of possible partitions
15	3	2,375,101
20	4	45,232,115,901
25	8	690,223,721,118,368,580
100	5	10^{68}

3^{15} possible partitions

The impracticability of examining every possible partition has led to the development of algorithms designed to search for the minimum values of the clustering criterion by rearranging existing partitions and keeping the new one only if it provides an improvement. Such algorithms do not, of course, guarantee finding the global minimum of the criterion. The essential steps in these algorithms are as follows:

K-Means Clustering Algorithm

- K-Means Loop
 - E – The "assignment" step is also referred to as expectation step,
 - M – The "update step" as maximization step, making this algorithm a variant of the *generalized expectation-maximization algorithm*.
- Until Convergence

K-means Algorithm

- For a current set of cluster means, assign each observation as:

$$C(i) = \arg \min_{1 \leq k \leq K} \|x_i - m_k\|^2, i = 1, \dots, N$$

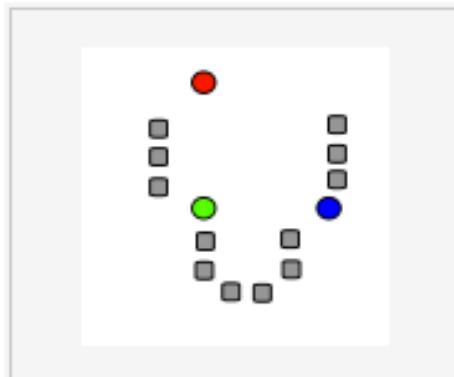
- For a given cluster assignment C of the data points, compute the cluster means m_k :

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, k = 1, \dots, K.$$

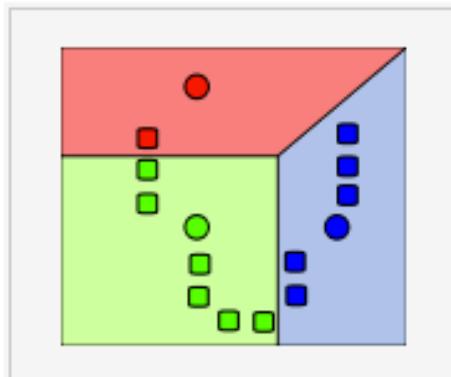
- Iterate above two steps until convergence

Clustering Algorithms

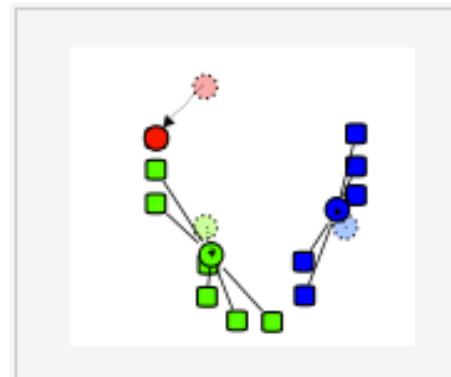
E-Step
Initialization "assignment" step M-Step
update step Converged



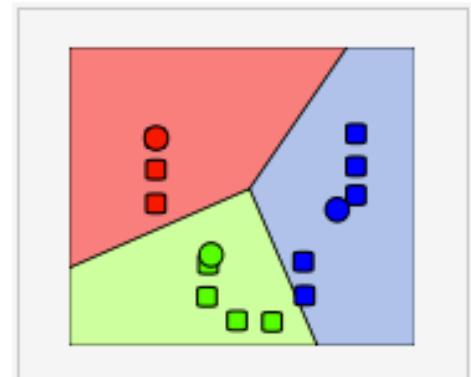
1) k initial "means" (in this case $k=3$) are randomly selected from the data set (shown in color).



2) k clusters are created by associating every observation with the nearest mean. The partitions here represent the [Voronoi diagram](#) generated by the means.



3) The [centroid](#) of each of the k clusters becomes the new means.



4) Steps 2 and 3 are repeated until convergence has been reached.

Clustering Algorithms

- **Non-Hierarchical Clustering**

- Exclusive Clustering (k-means) [In R]
- Overlapping Clustering (fuzzy c means)
- Probabilistic Clustering (Mixture of Gaussians)

Question: can both types of algorithm be effectively implemented in Hadoop?

- **Hierarchical Clustering**

- Agglomerative approach (bottom-up)

Hierarchical Clustering is tough to deploy:

Distributing a bottom-up algorithm is tricky because each distributed process needs the entire dataset to make choices about appropriate clusters. It also needs a list of clusters at its current level so it doesn't add a data point to more than one cluster at the same level.

Kmeans in Spark

- **Notebook**
 - <https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0>
- **NBViewer**
 - <http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb>

Kmeans In Spark: E-Step: Cluster Assignment Function

SparkContext available as sc, HiveContext available as sqlContext.

$$C(i) = \arg \min_{1 \leq k \leq K} \|x_i - m_k\|^2, i = 1, \dots, N$$

```
1 import numpy as np
2
3 #Calculate which class each data point belongs to
4 def nearest_centroid(line):
5     x = np.array([float(f) for f in line.split(',')])
6     closest_centroid_idx = np.sum((x - centroids)**2, axis=1).argmin()
7     return (closest_centroid_idx, (x,1))
8
9 #plot centroids and data points for each iteration
10 def plot_iteration(means):
11     pylab.plot(samples1[:, 0], samples1[:, 1], '.', color = 'blue')
12     pylab.plot(samples2[:, 0], samples2[:, 1], '.', color = 'blue')
13     pylab.plot(samples3[:, 0], samples3[:, 1], '.', color = 'blue')
14     pylab.plot(means[0][0], means[0][1], '*', markersize = 10, color = 'red')
15     pylab.plot(means[1][0], means[1][1], '*', markersize = 10, color = 'red')
16     pylab.plot(means[2][0], means[2][1], '*', markersize = 10, color = 'red')
17     pylab.show()
```

Kmeans In Spark:

```
1 K = 3
2 # Initialization: initialization of parameter is fixed to show an example
3 centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
4
5 D = sc.textFile("./data.csv").cache()
6 iter_num = 0
7 for i in range(10):    E-Step
8     res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).collect()
9     #res [(0, (array([ 2.66546663e+00,  3.94844436e+03]), 1001) ),
10    # (2, (array([ 6023.84995923,  5975.48511018]), 1000)),
11    # (1, (array([ 3986.85984761,  15.93153464]), 999))]
12    # res[1][1][1] returns 1000 here
13    res = sorted(res,key = lambda x : x[0])  #sort based on clustered ID
14    centroids_new = np.array([x[1][0]/x[1][1] for x in res]) #divide by cluster size
15    if np.sum(np.absolute(centroids_new-centroids))<0.01:      M-Step Sum Part 2
16        break
17    print "Iteration" + str(iter_num)
18    iter_num = iter_num + 1
19    centroids = centroids_new
20    print centroids
21    plot_iteration(centroids)
22 print "Final Results:"
23 print centroids
```

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, k=1,\dots,K$$

M-Step Sum Part 1

M-Step Sum Part 2

**Kmeans In Spark:
Cluster Assignment Function**

Iteration0

```
[[ 0.80217059  0.61622248]
 [ 3.94191443  2.67285704]
 [ 2.18436067  5.72221018]]
```

```

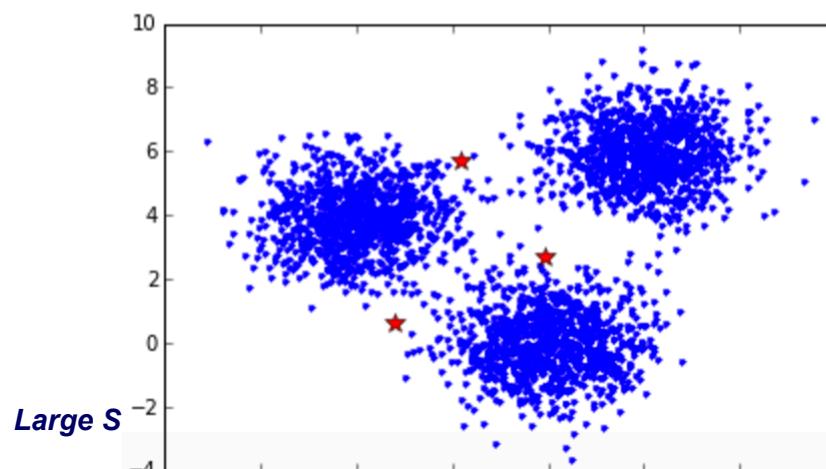
1 K = 3
2 # Initialization: initialization of parameter is fixed to show an example
3 centroids = np.array([[0.0,0.0],[2.0,2.0],[0.0,7.0]])
4
5 D = sc.textFile("./data.csv").cache()
6 iter_num = 0
7 for i in range(10):    E-Step          M-Step Sum Part 1
8     res = D.map(nearest_centroid).reduceByKey(lambda x,y : (x[0]+y[0],x[1]+y[1])).collect()
9     #res [(0, (array([ 2.66546663e+00, 3.94844436e+03]), 1001) ),
10    # (2, (array([ 6023.84995923, 5975.48511018]), 1000)),
11    # (1, (array([ 3986.85984761, 15.93153464]), 999))]
12    # res[1][1][1] returns 1000 here
13    res = sorted(res,key = lambda x : x[0]) #sort based on clustered ID
14    centroids_new = np.array([x[1][0]/x[1][1] for x in res]) #divide by cluster size
15    if np.sum(np.absolute(centroids_new-centroids))<0.01:
16        break
17    print "Iteration" + str(iter_num)
18    iter_num = iter_num + 1
19    centroids = centroids_new
20    print centroids
21    plot_iteration(centroids)
22 print "Final Results:"
23 print centroids

```

```

Iteration0
[[ 0.80217059  0.61622248]
 [ 3.94191443  2.67285704]
 [ 2.18436067  5.72221018]]

```



Kmeans In Spark: Cluster Assignment Function

Kmeans in Spark

- **Notebook**
 - <https://www.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb?dl=0>
- **NBViewer**
 - <http://nbviewer.ipython.org/urls/dl.dropbox.com/s/41q9lgyqhy8ed5g/EM-Kmeans.ipynb>