Computational Intelligence Approaches for Big Data

Francisco Herrera

Soft Computing and
Information Intelligent Systems (SCI²S)
University of Granada, Spain
Email: herrera@decsai.ugr.es
http://sci2s.ugr.es

Isaac Triguero

School of Computer Science
University of Nottingham
United Kingdom
Email: Isaac.Triguero@nottingham.ac.uk

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Big Data

Our world revolves around the data

- Science
  - Data bases from astronomy, genomics, environmental data, transportation data, …
- Humanities and Social Sciences
  - Scanned books, historical documents, social interactions data, …
- Business & Commerce
  - Corporate sales, stock market transactions, census, airline traffic, …
- Entertainment
  - Internet images, Hollywood movies, MP3 files, …
- Medicine
  - MRI & CT scans, patient records, …
- Industry, Energy, …
  - Sensors, …

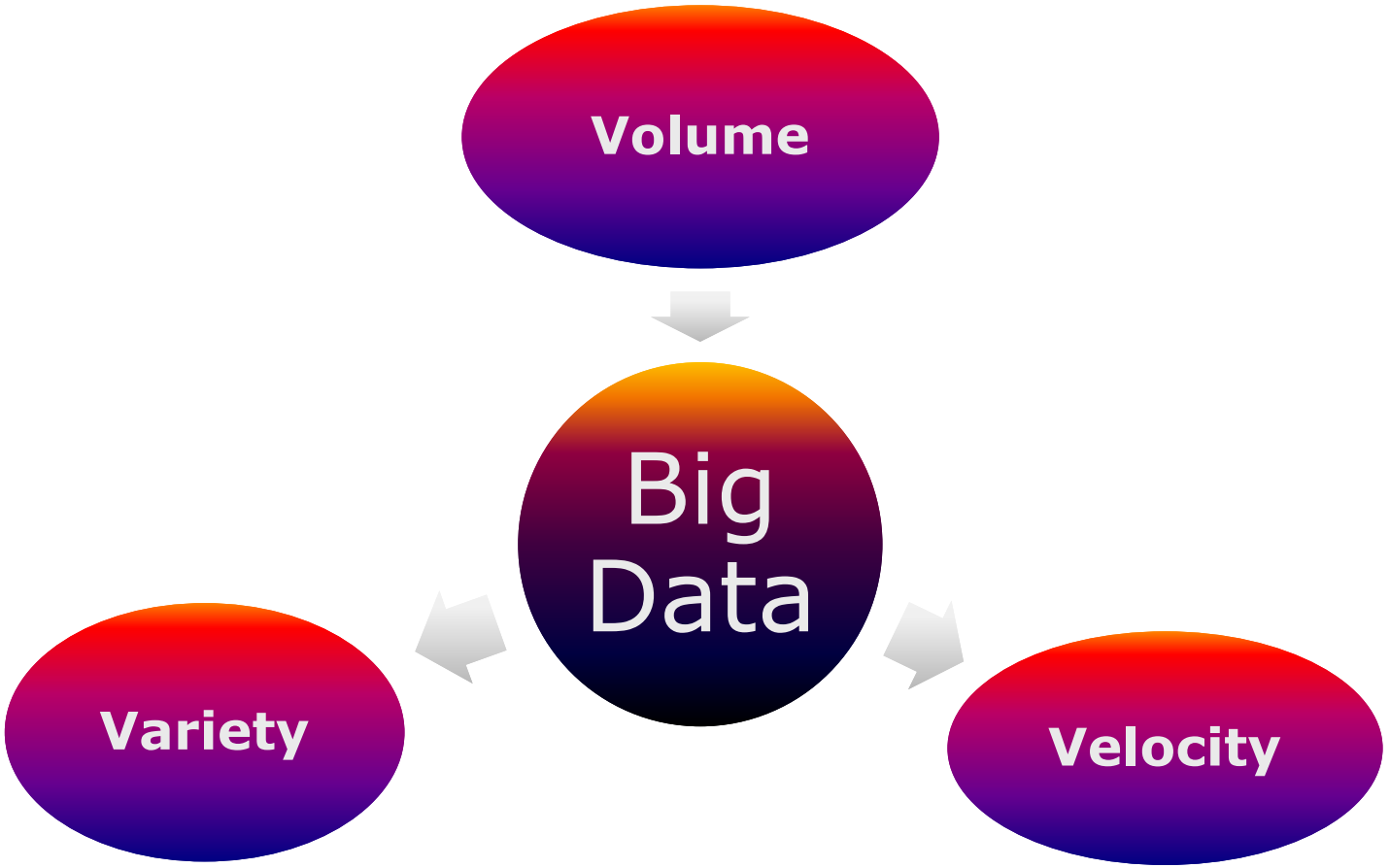
Big Data

Data is at the centre of the knowledge economy and society

Ben Chams - Fotolia

Big Data

Big Data 3 V's

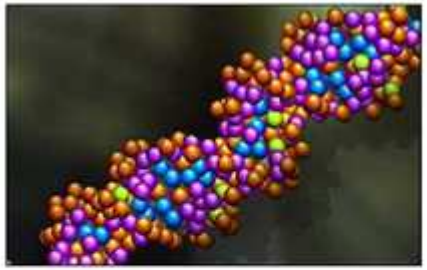


Doug Laney, Gartner 2001

What is Big Data? 3 Vs of Big Data

volume

Ej. Genomics



- 25,000 genes in human genome
- 3 billion bases
- 3 Gigabytes of genetic data

Astronomy

- Astronomical sky surveys
- 120 Gigabytes/week
- 6.5 Terabytes/year

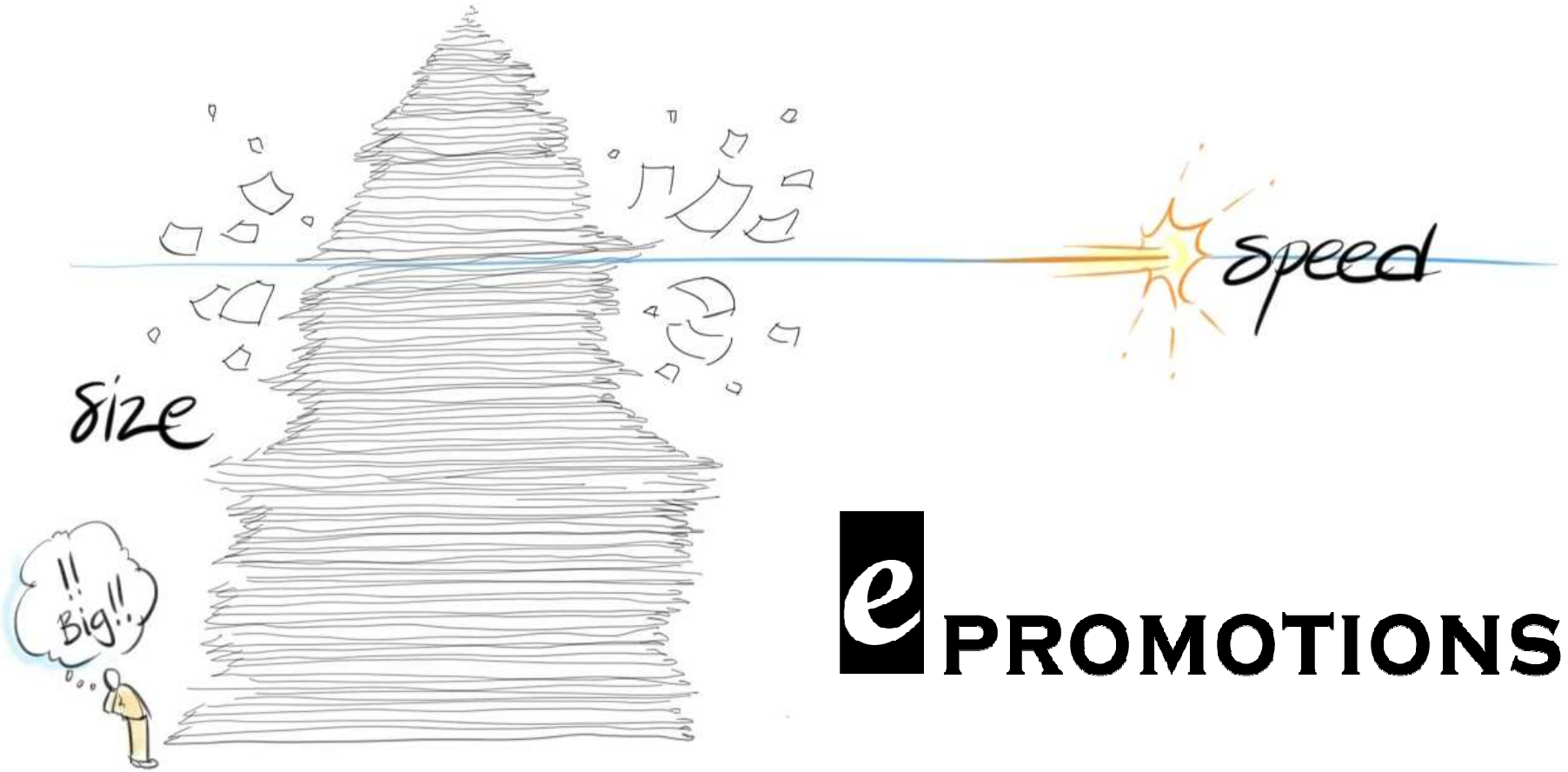
Transactions

- 47.5 billion transactions in 2005 worldwide
- 115 Terabytes of data transmitted to VisaNet data processing center in 2004

Gartner, Doug Laney, 2001

What is Big Data? 3 Vs of Big Data

velocity



ePROMOTIONS

Gartner, Doug Laney, 2001

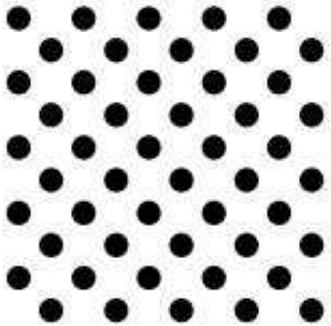
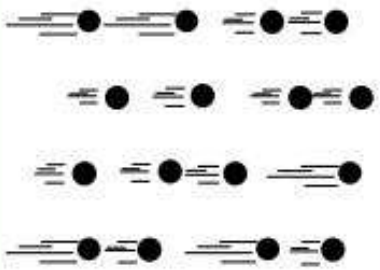
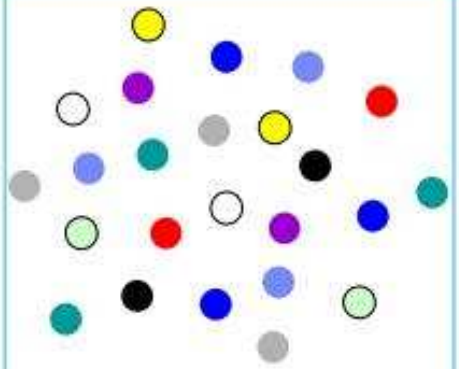
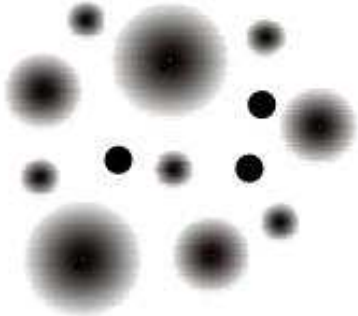
What is Big Data? 3 Vs of Big Data

variety

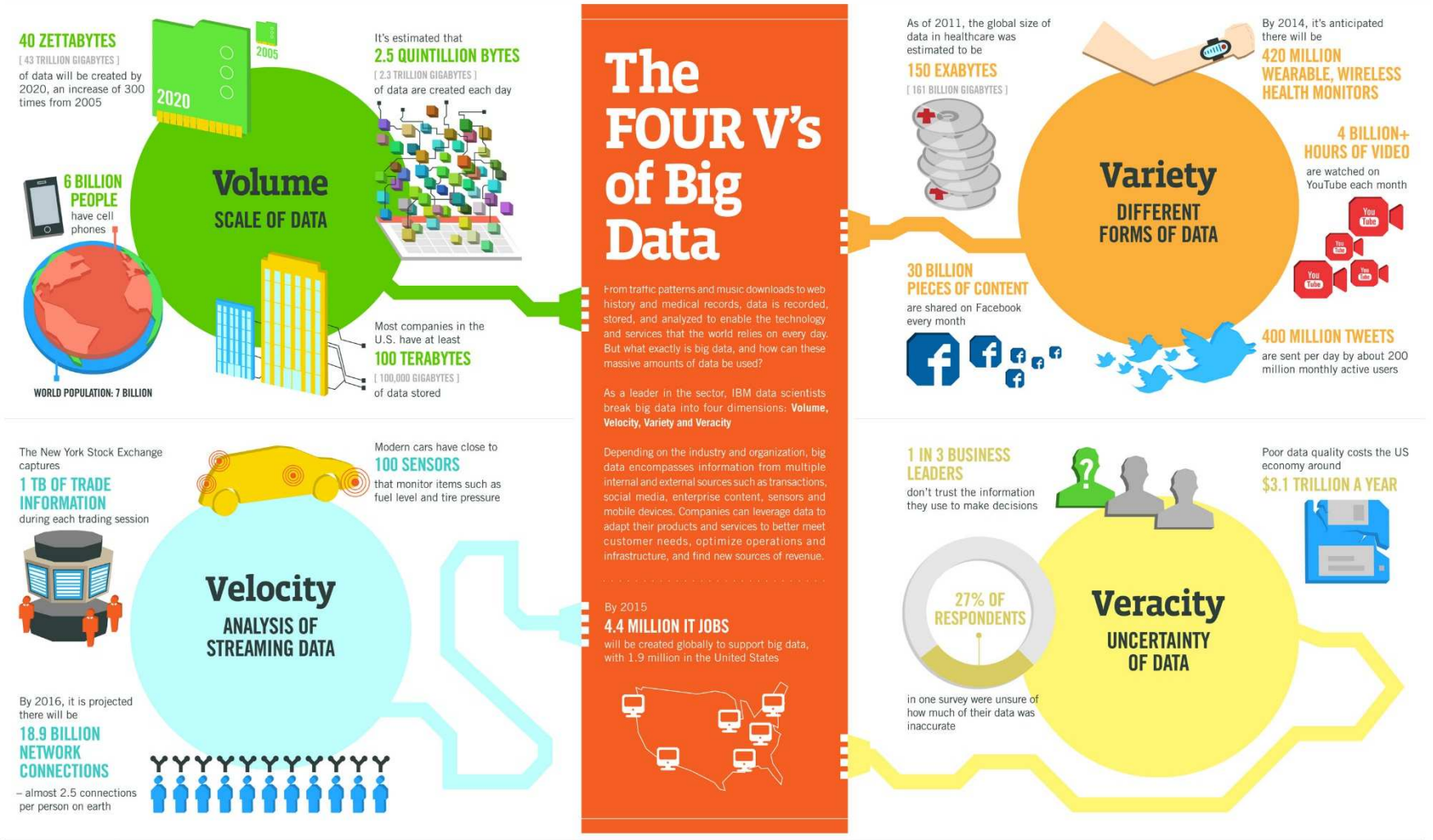
Gartner, Doug Laney, 2001

What is Big Data? 3 Vs of Big Data

Some Make it 4V's

Volume	Velocity	Variety	Veracity*
			
Data at Rest	Data in Motion	Data in Many Forms	Data in Doubt
Terabytes to exabytes of existing data to process	Streaming data, milliseconds to seconds to respond	Structured, unstructured, text, multimedia	Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

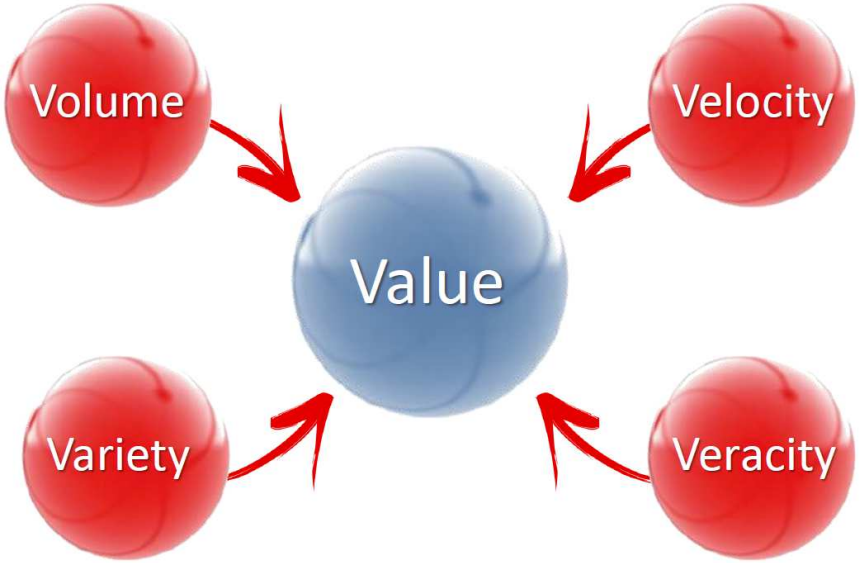
What is Big Data?



What is Big Data? One more V?

Value: data in use

- Most important motivation for big data
- Big data may result in:
  - Better statistics/models
  - Novel insights
  - New opportunities for research and industry

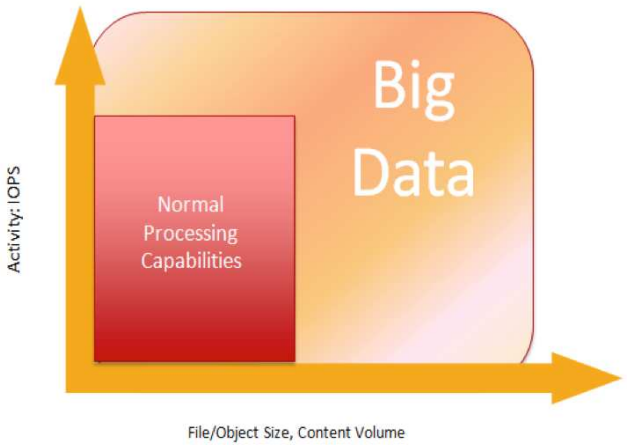


What is Big Data?

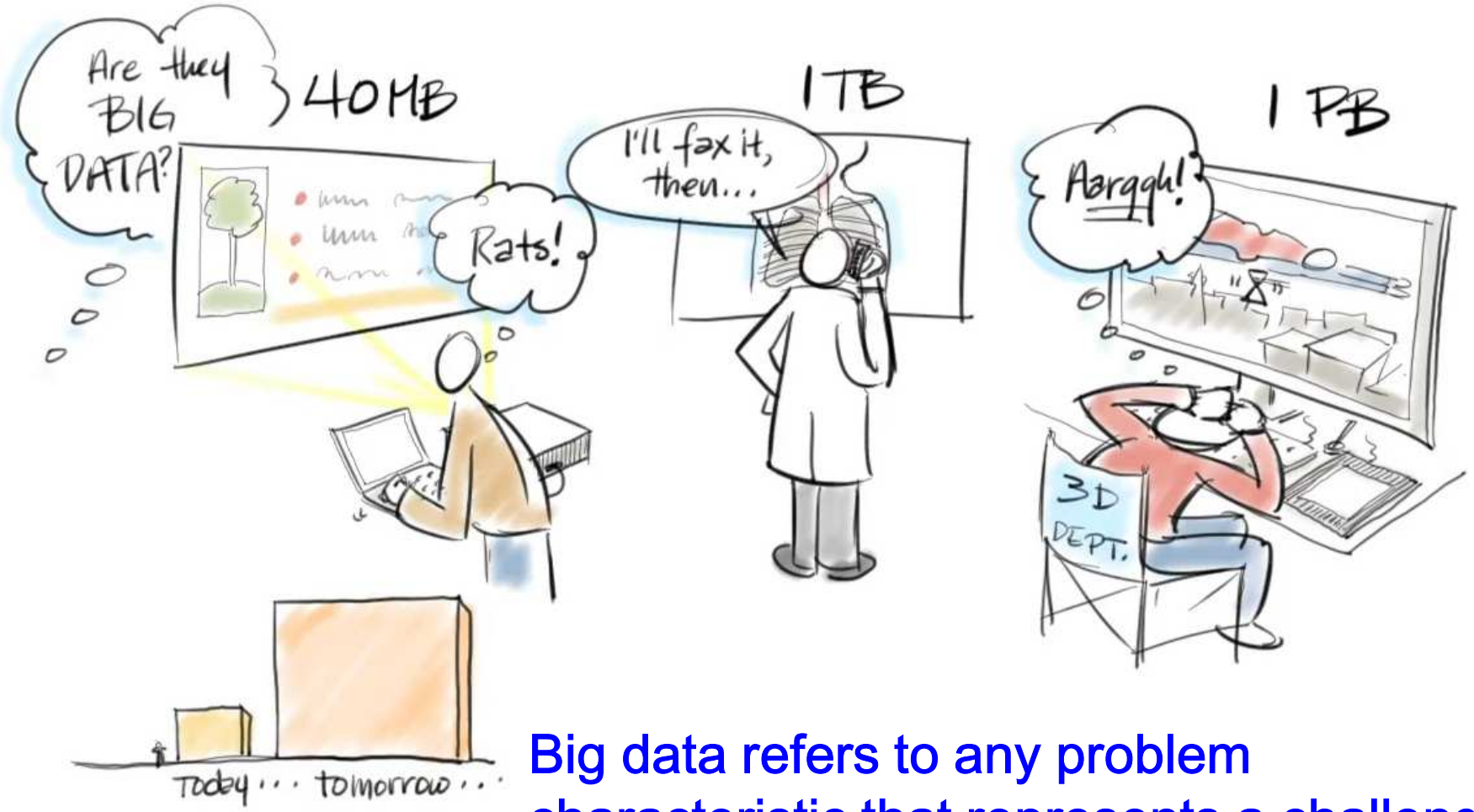
No single standard definition

Big data is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

"***Big Data***" is data whose scale, diversity, and complexity **require new architectures, techniques, algorithms, and analytics** to manage it and extract value and hidden knowledge from it…



What is Big Data? (in short)

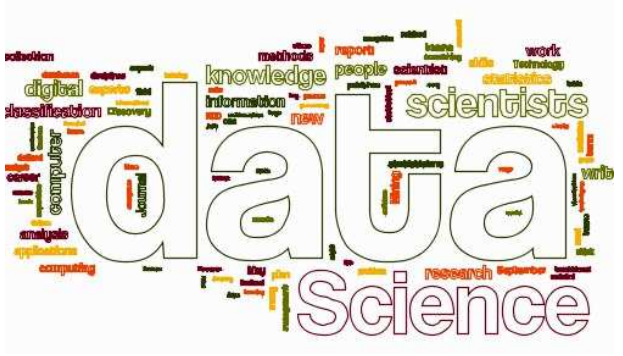


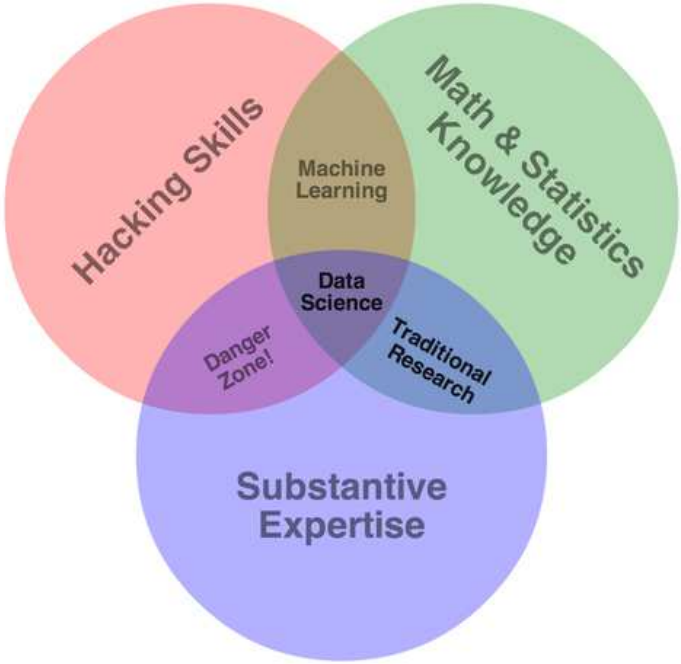
Big data refers to any problem characteristic that represents a challenge to process it with traditional applications

(Big) Data Science

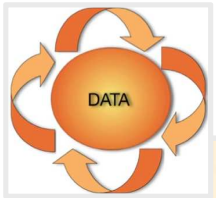
Data Science combines the traditional scientific method with the ability to munch, explore, learn and gain deep insight for (Big) Data

It is not just about finding patterns in data … it is mainly about explaining those patterns





Data Science Process



Data Preprocessing

- Clean
- Sample
- Aggregate
- Imperfect data: missing, noise, …
- Reduce dim.
- …

> 70% time!

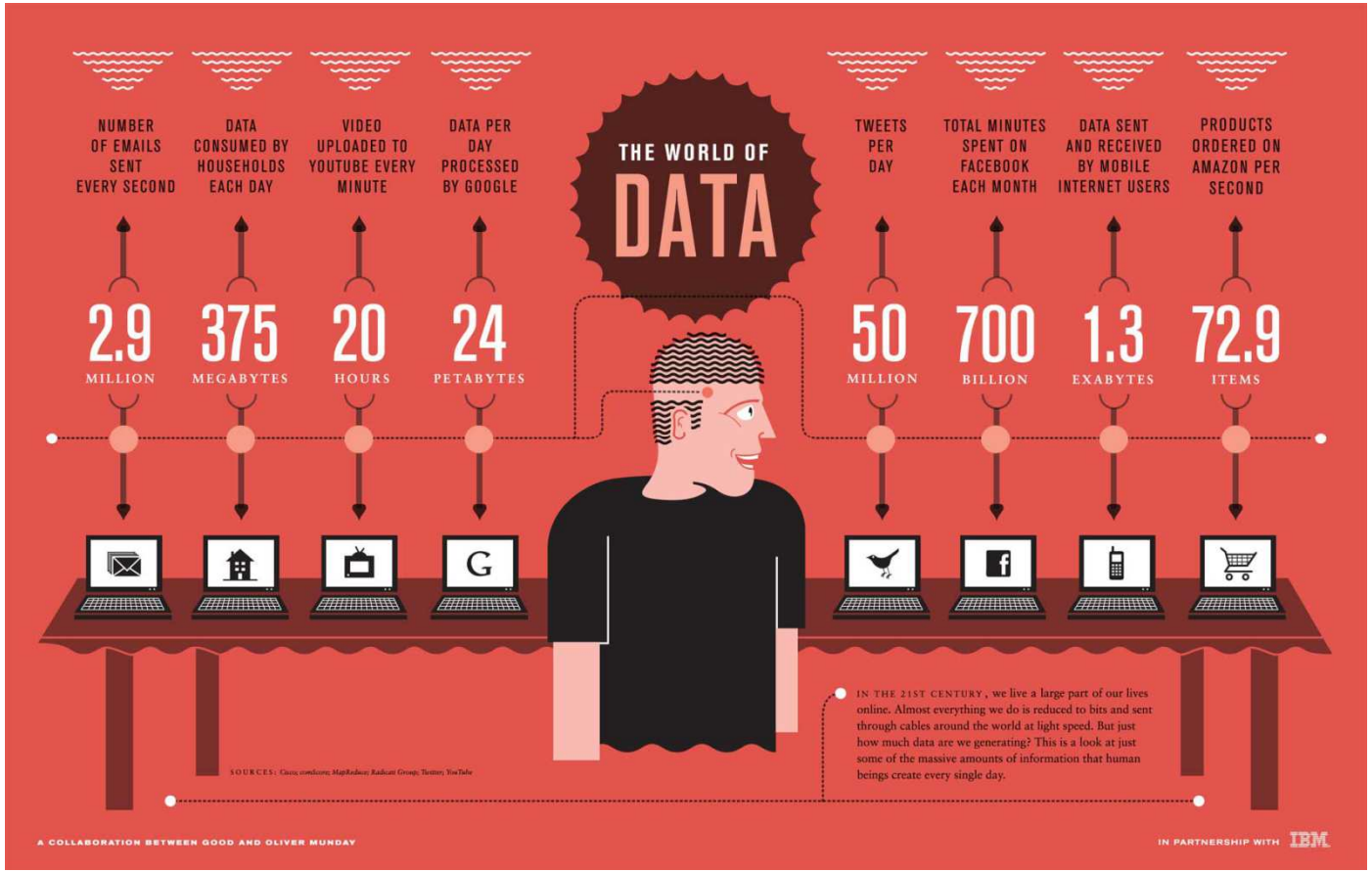
Data Processing

- Explore data
- Represent data
- Link data
- Learn from data
- Deliver insight
- …

Data Analytics

- Clustering
- Classification
- Regression
- Network analysis
- Visual analytics
- Association
- …

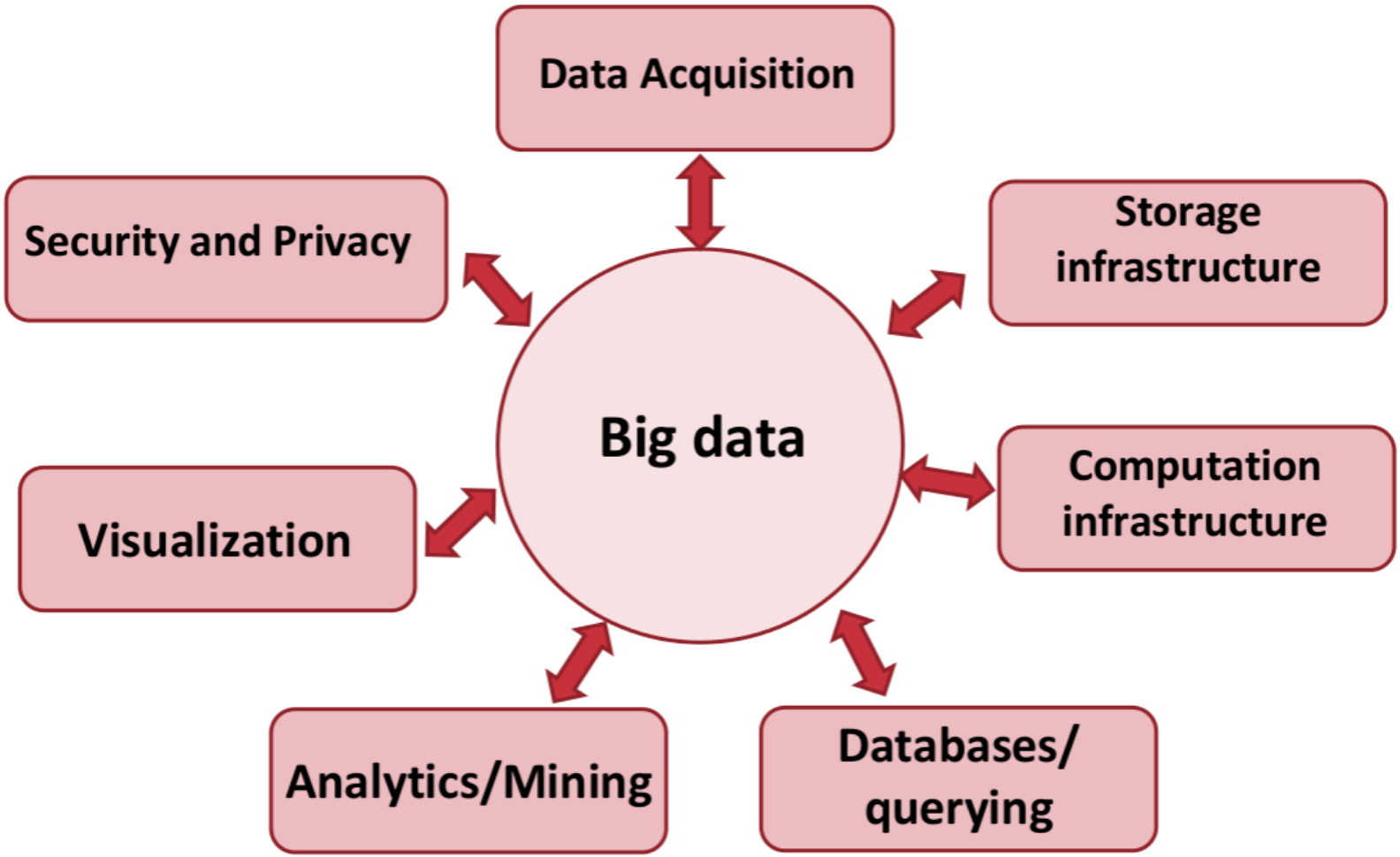
Data Science Process



Systematic methodologies dealing with data efficiently, called data science, should be done.

https://aadamov.wordpress.com/2012/03/

Big data has many faces

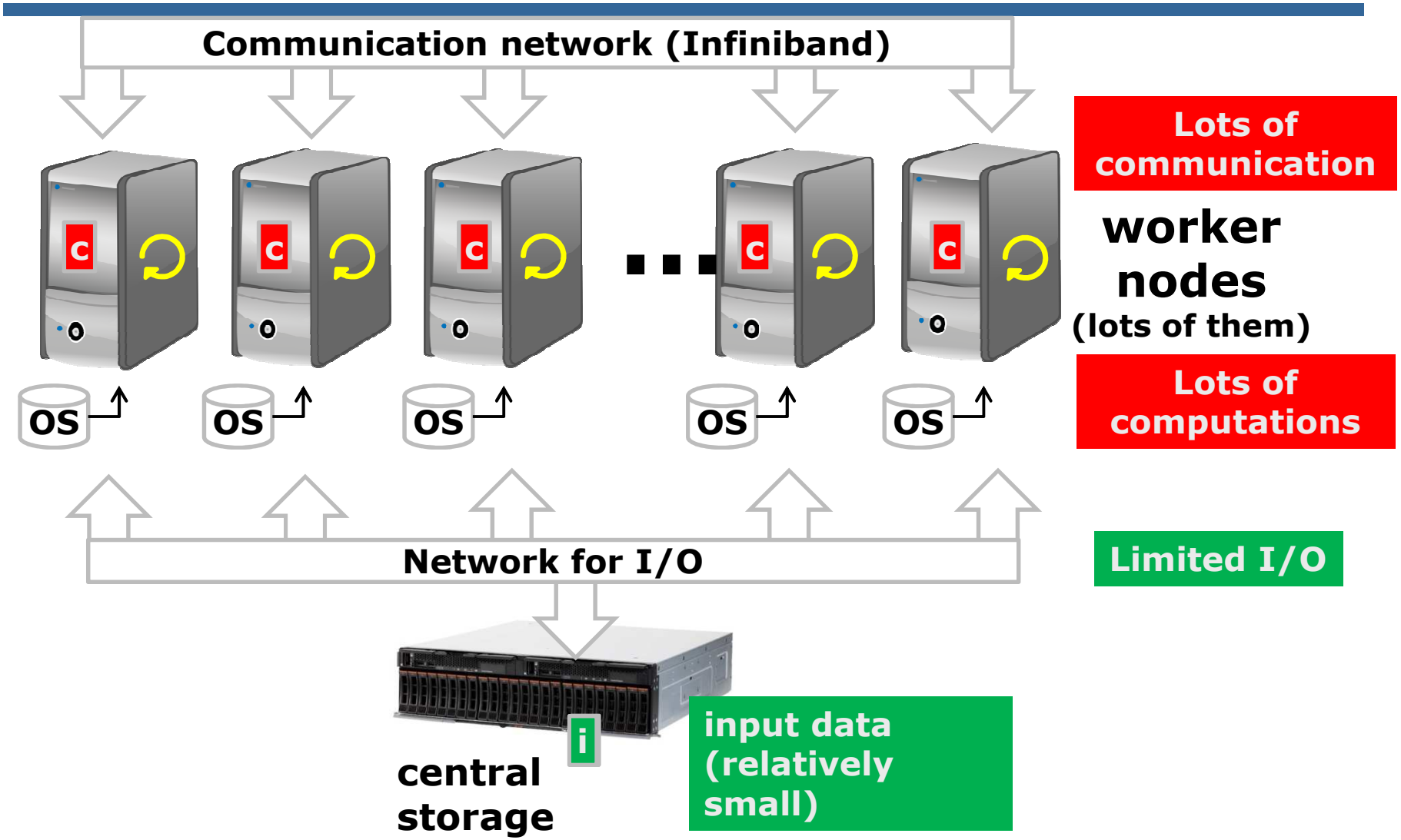


How to deal with big data?

- **Problem statement:** scalability to big data sets.
- **Example:**
  - Explore 100 TB by 1 node @ 50 MB/sec = 23 days
  - Exploration with a cluster of 1000 nodes = 33 minutes
- **Solution➔ Divide-And-Conquer**

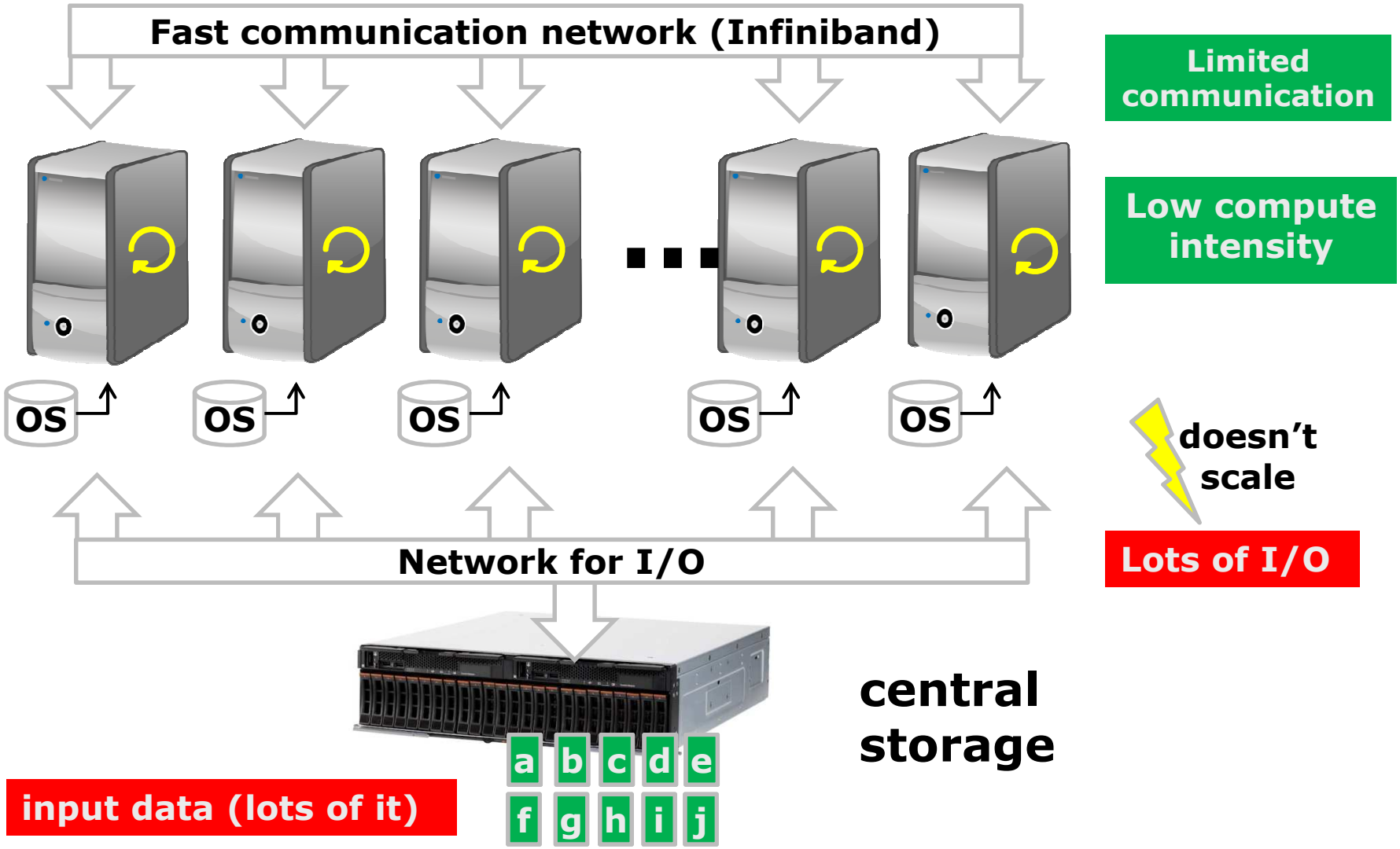
A single machine cannot efficiently manage high volumes of data.
Vertical scalability ➔
Horizontal scalability

Traditional HPC way of doing things

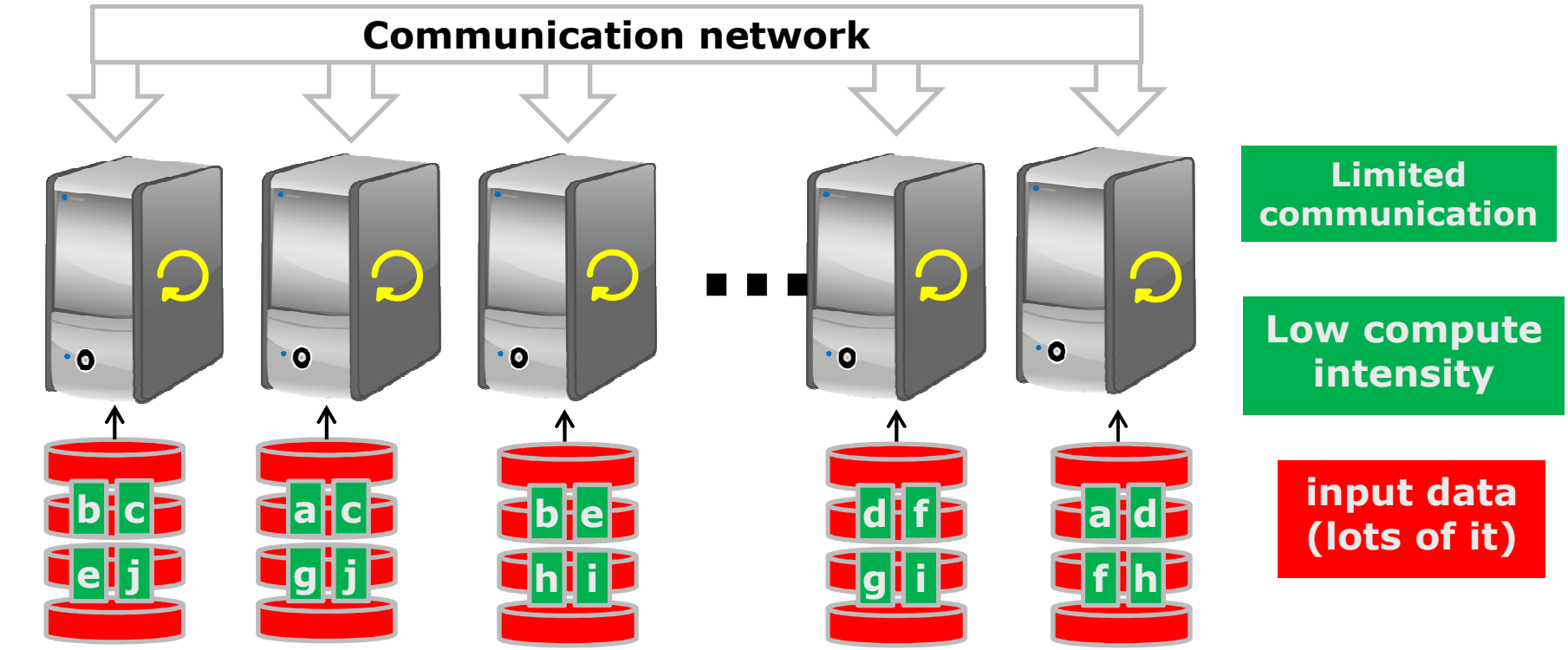


Source: Jan Fostier. Introduction to MapReduce and its Application to Post-Sequencing Analysis

Data-intensive jobs



Data-intensive jobs



Solution: store data on local disks of the nodes that perform computations on that data ("data locality")

MapReduce

- Scalability to large data volumes:
  - Scan 100 TB on 1 node @ 50 MB/sec = 23 days
  - Scan on 1000-node cluster = 33 minutes

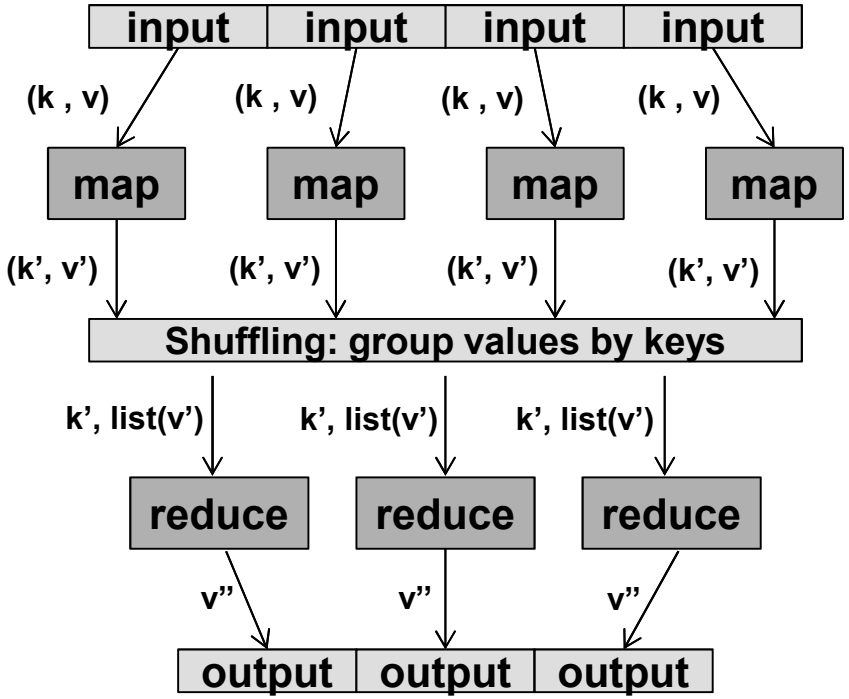
➔ Divide-And-Conquer (i.e., data partitioning)

MapReduce

- **Overview:**
  - Data-parallel programming model
  - An associated parallel and distributed implementation for commodity clusters
- **Pioneered by Google**
  - Processes 20 PB of data per day
- **Popularized by open-source Hadoop project**
  - Used by Yahoo!, Facebook, Amazon, and the list is growing …

MapReduce

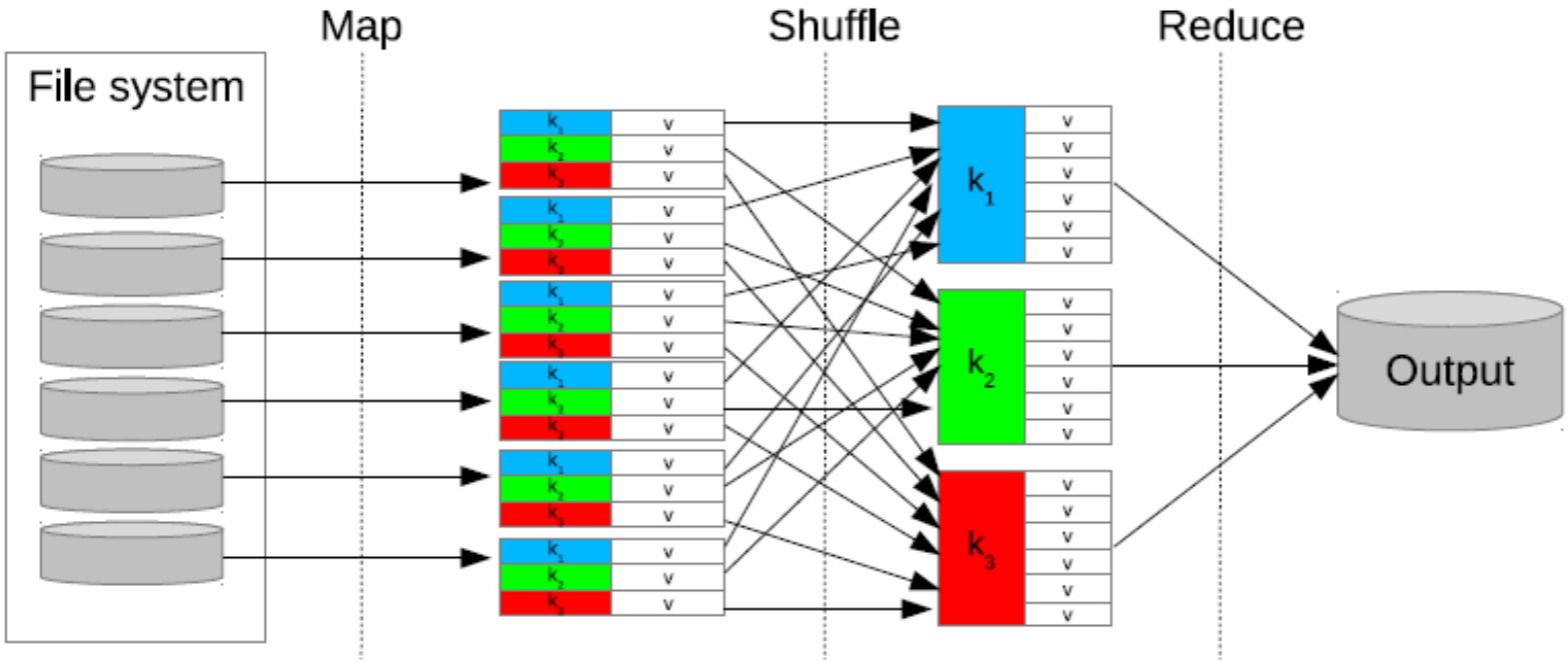
- MapReduce is a popular approach to deal with Big Data
- Based on a *key-value pair* data structure
- Two key operations:
  1. **Map function:** Process independent data blocks and outputs summary information
  2. **Reduce function:** Further process previous independent results



J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107-113.

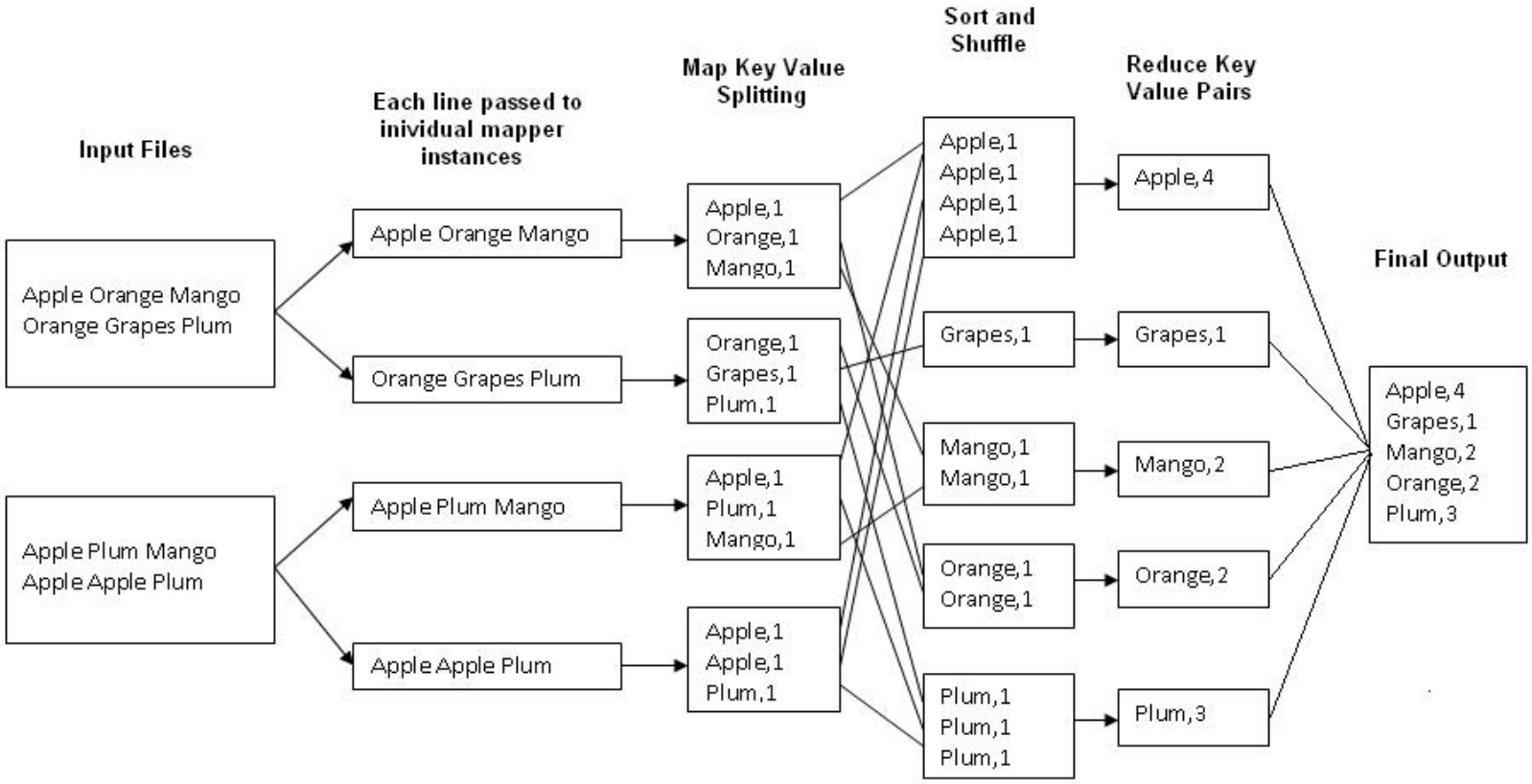
MapReduce

MapReduce workflow



The key of a MapReduce data partitioning approach is usually on the reduce phase

MapReduce: The WordCount Example



MapReduce: The WordCount Example

Pseudo-code:

```
map(key, value):
// key: document ID; value: text of document
    FOR (each word w in value)
        emit(w, 1);

reduce(key, value-list):
// key: a word; value-list: a list of integers
        result = 0;
        FOR (each count v on value-list)
            result += v;
        emit(key, result);
```

MapReduce

Experience

- **Runs on large commodity clusters:**
  - **10s to 10,000s of machines**
- **Processes many terabytes of data**
- **Easy to use since run-time complexity hidden from the users**
- **Cost-efficiency:**
  - **Commodity nodes (cheap, but unreliable)**
  - **Commodity network**
  - **Automatic fault-tolerance (fewer administrators)**
  - **Easy to use (fewer programmers)**

MapReduce

- **Advantage: MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance**

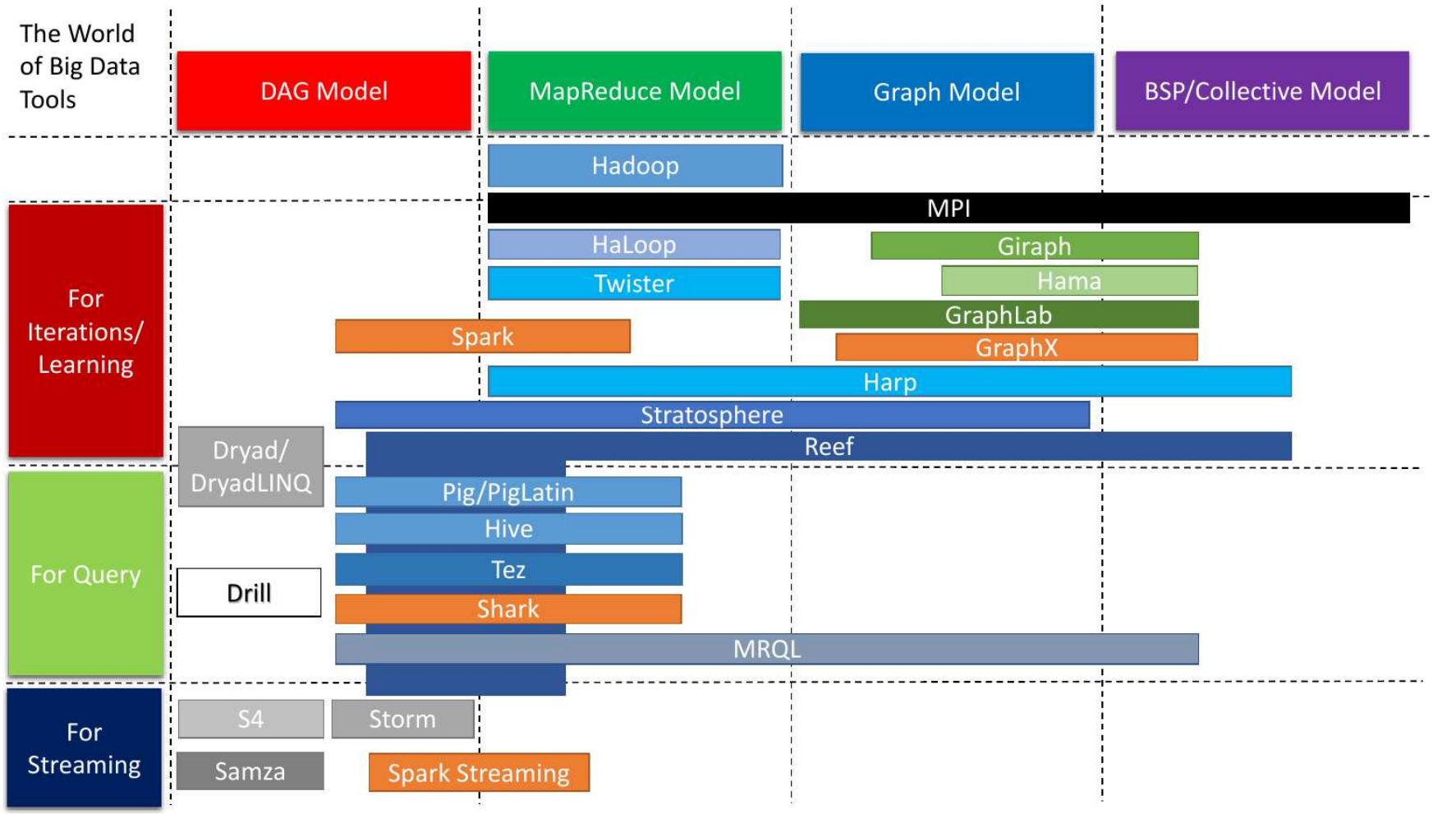
- **Key philosophy:**
  - ***Make it scale*, so you can throw hardware at problems**
  - ***Make it cheap*, saving hardware, programmer and administration costs (but requiring fault tolerance)**

- **MapReduce is not suitable for all problems, but when it works, it may save you a lot of time**

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Big data technologies



MapReduce. Hadoop

Hadoop is an open source implementation of MapReduce computational paradigm

Created by Doug Cutting (chairman of board of directors of the Apache Software Foundation, 2010)

http://hadoop.apache.org/

Hadoop birth

July 2008 - Hadoop Wins Terabyte Sort Benchmark
One of Yahoo's Hadoop clusters sorted 1 terabyte of data in 209 seconds, which beat the previous record of 297 seconds in the annual general purpose (Daytona) terabyte short bechmark. This is the first time that either a Java or an open source program has won.

2008, 3.48 minutes

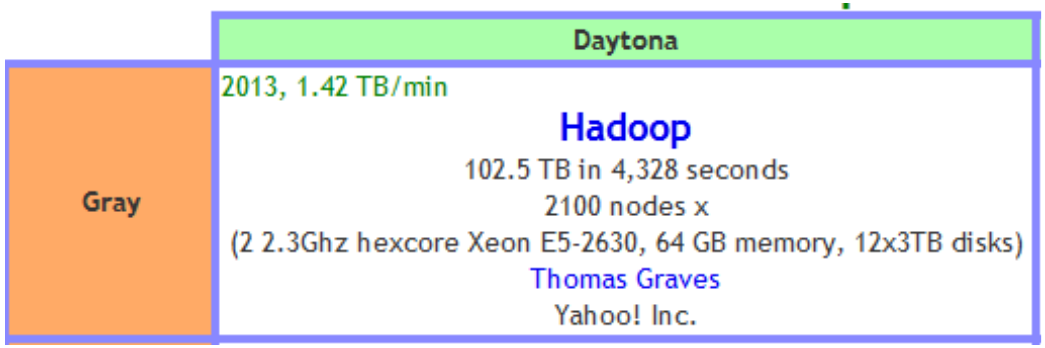
Hadoop

910 nodes x (4 dual-core processors, 4 disks, 8 GB memory)
Owen OMalley, Yahoo

2007, 4.95 min

TokuSampleSort

tx2500 disk cluster
400 nodes x (2 processors, 6-disk RAID, 8 GB memory)
Bradley C. Kuszmaul , MIT



http://developer.yahoo.com/blogs/hadoop/hadoop-sorts-petabyte-16-25-hours-terabyte-62-422.html

Hadoop

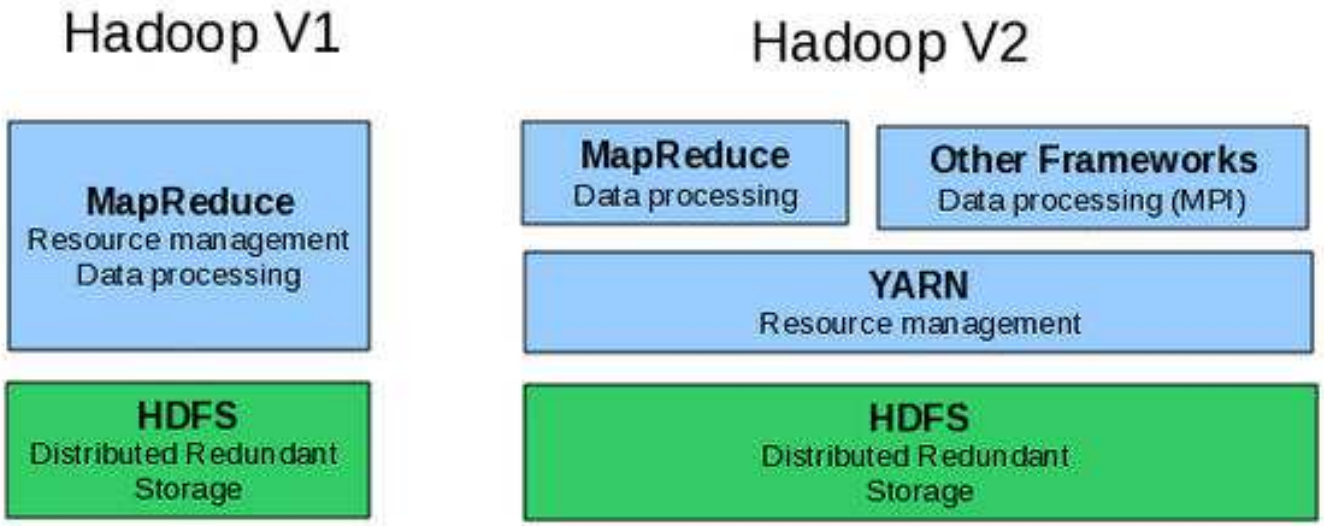
- Hadoop is:
  - An **open-source** framework written in Java
  - Distributed storage of very large data sets (Big Data)
  - Distributed processing of very large data sets

- This framework consists of a **number of modules**
  - *Hadoop Common*
  - ***Hadoop Distributed File System (HDFS)***
  - *Hadoop YARN* – resource manager
  - ***Hadoop MapReduce*** – programming model

http://hadoop.apache.org/

Hadoop Evolution

MapReduce Limitations. Graph algorithms (Page Rank, Google), iterative algorithms.



Hadoop Ecosystem

A. Fernandez, S. Río, V. López, A. Bawakid, M.J. del Jesus, J.M. Benítez, F. Herrera, **Big Data with Cloud Computing: An Insight on the Computing Environment, MapReduce and Programming Frameworks.** *WIREs Data Mining and Knowledge Discovery 4:5 (2014) 380-409*

Hadoop Ecosystem

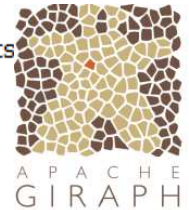
The project

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

- Avro™: A data serialization system.
- Cassandra™: A scalable multi-master database with no single points of failure.
- Chukwa™: A data collection system for managing large distributed systems.
- HBase™: A scalable, distributed database that supports structured data s
- Hive™: A data warehouse infrastructure that provides data summarization
- Mahout™: A Scalable machine learning and data mining library.
- Pig™: A high-level data-flow language and execution framework for parallel
- ZooKeeper™: A high-performance coordination service for distributed applications.



GIRAPH (APACHE Project)
(http://giraph.apache.org/)
Iterative Graphs

Recently: Apache Spark

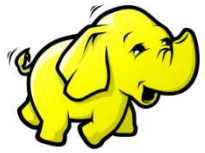
Spark (UC Berkeley)
(100 times more efficient than Hadoop, including iterative algorithms, according to creators)

http://hadoop.apache.org/

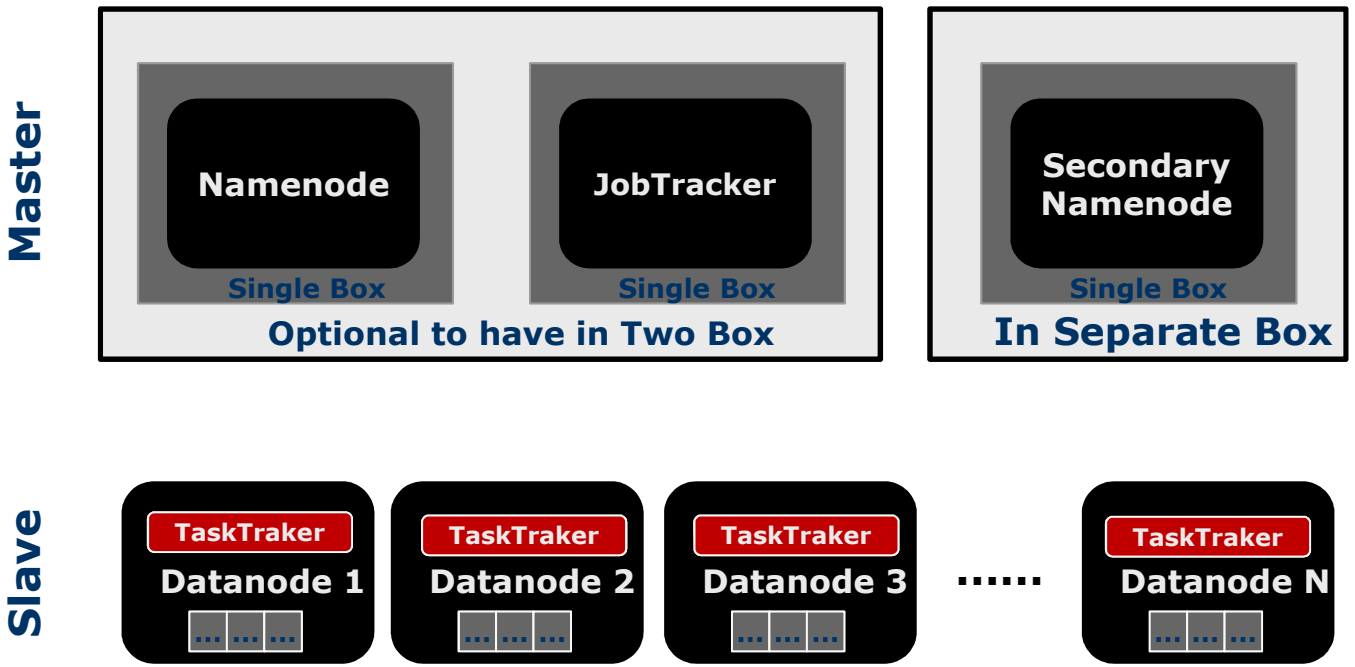
The Hadoop File System

- **Hadoop Distributed File System (HDFS)** is an scalable and flexible distributed file system, written in **Java** for Hadoop.
- **Shared-nothing cluster** of thousand nodes, built from inexpensive hardware => node failure is the norm!
- Very large files, of typically several GB, containing many objects.
- Mostly read and append (random updates are rare)
  - Large reads of bulk data (e.g. 1 MB) and small random reads (e.g. 1KB)
  - Append operations are also large and there may be many concurrent clients that append the same file.
- High throughput (for bulk data) more important than low latency.

Hadoop: A master/slave architecture



- **Master**: NameNode, JobTracker
- **Slave**: {DataNode, TaskTraker}, ..., {DataNode, TaskTraker}



Hadoop

Hadoop can be run with 3 different configurations:

1. **Local / *Standalone.* It is run in a single JVM (*Java Virtual Machine). Very useful for debugging!***
2. **Pseudo-distributed (Cluster simulator)**
3. **Distributed (Cluster)**

WordCount using Hadoop MapReduce

Map()

```
public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{
```

Member variables

```
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(Object key, Text value, Context context
                   ) throws IOException, InterruptedException {
     StringTokenizer itr = new
       StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
     }
   }
 }
```

WordCount using Hadoop MapReduce

Reduce()

```
public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }
```

WordCount using Hadoop MapReduce

The Main **Function for the WordCount program**

```
 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...] <out>");
      System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

MapReduce: summary

- **Advantages compared to classical distributed models:** Simplicity and fault tolerant mechanism!

  Appropriate for data-intensive processes!

- **Main keys**:
  - ***Scalable:*** *no matter about underlying hardware*
  - ***Cheaper:*** *Hardware, programming and administration savings!*

- **WARNING: MapReduce** could not solve any kind of problems, BUT when it works, it may save a lot time!

MapReduce limitations

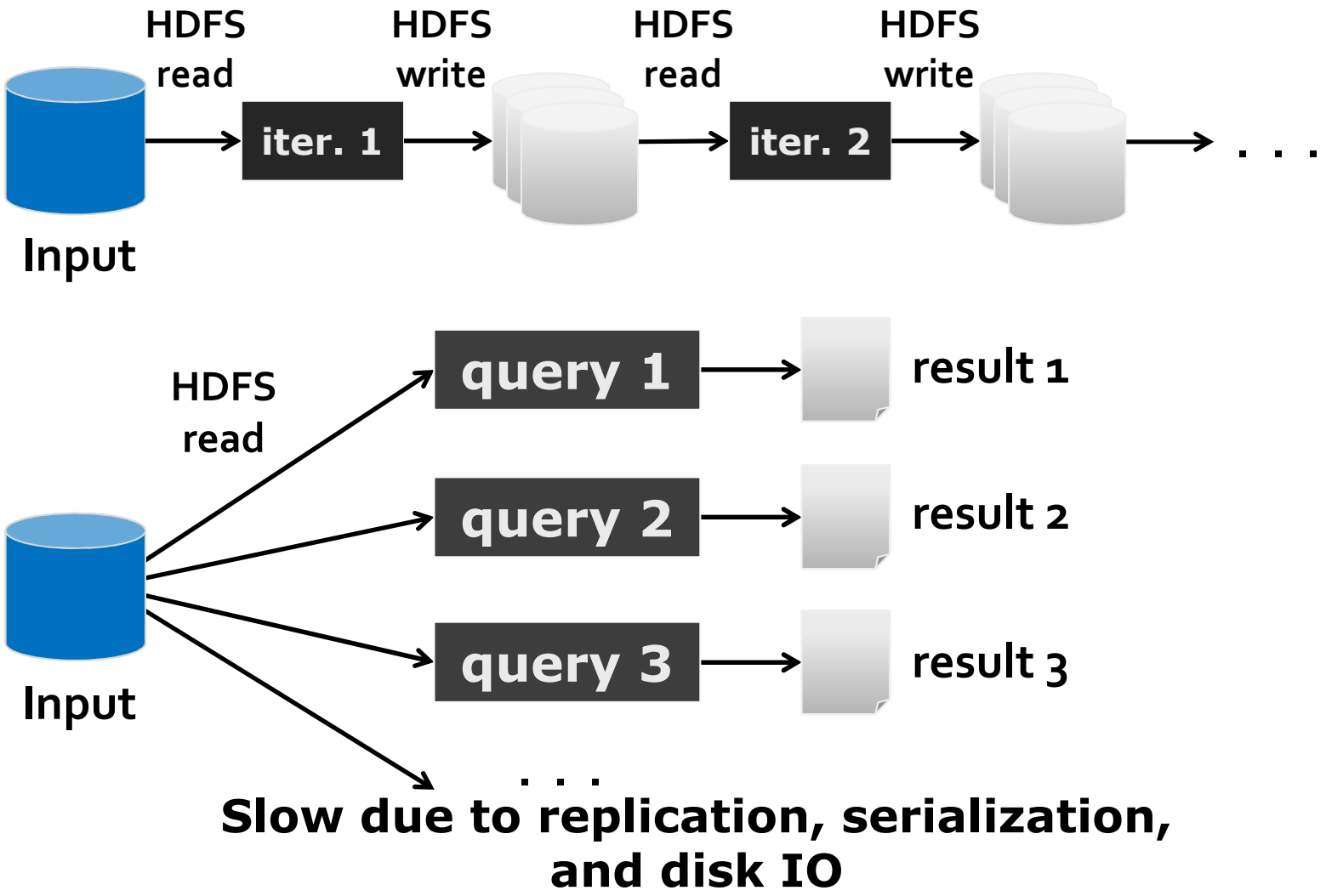
"If all you have is a hammer, then everything looks like a nail."

There are some recent extension to the MapReduce paradigm in order to ease the iterative computations!

Why a New Programming Model?

- MapReduce simplified big data processing, but users quickly found two problems:
  - **Programmability:** tangle of map/red functions
  - **Speed:** MapReduce inefficient for apps that share data across multiple steps
    - Iterative algorithms, interactive queries

Data Sharing in MapReduce



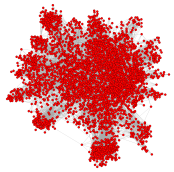
Paradigms that do not fit with Hadoop MapReduce

- **Directed Acyclic Graph (DAG) model:**
  - The DAG defines the dataflow of the application, and the vertices of the graph defines the operations that are to be performed on the data.
  - The "computational vertices" are written using sequential constructs, devoid of any concurrency or mutual exclusion semantics.
- **Graph model:**
  - More complex graph models that better represent the dataflow of the application.
  - Cyclic models -> Iterativity.
- **Iterative MapReduce model:**
  - An extented programming model that supports iterative MapReduce computations efficiently.

Hadoop

New platforms to overcome Hadoop's limitations

GIRAPH (APACHE Project)
(http://giraph.apache.org/)
Procesamiento iterativo de grafos



GPS - A Graph Processing System, (Stanford)
http://infolab.stanford.edu/gps/
para Amazon's EC2

Distributed GraphLab (Carnegie Mellon Univ.)
https://github.com/graphlab-code/graphlab
Amazon's EC2

Spark (UC Berkeley)
(100 times more efficient than Hadoop, including iterative algorithms, according to creators)
http://spark.incubator.apache.org/research.html

Twister (Indiana University)
http://www.iterativemapreduce.org/
Clusters propios

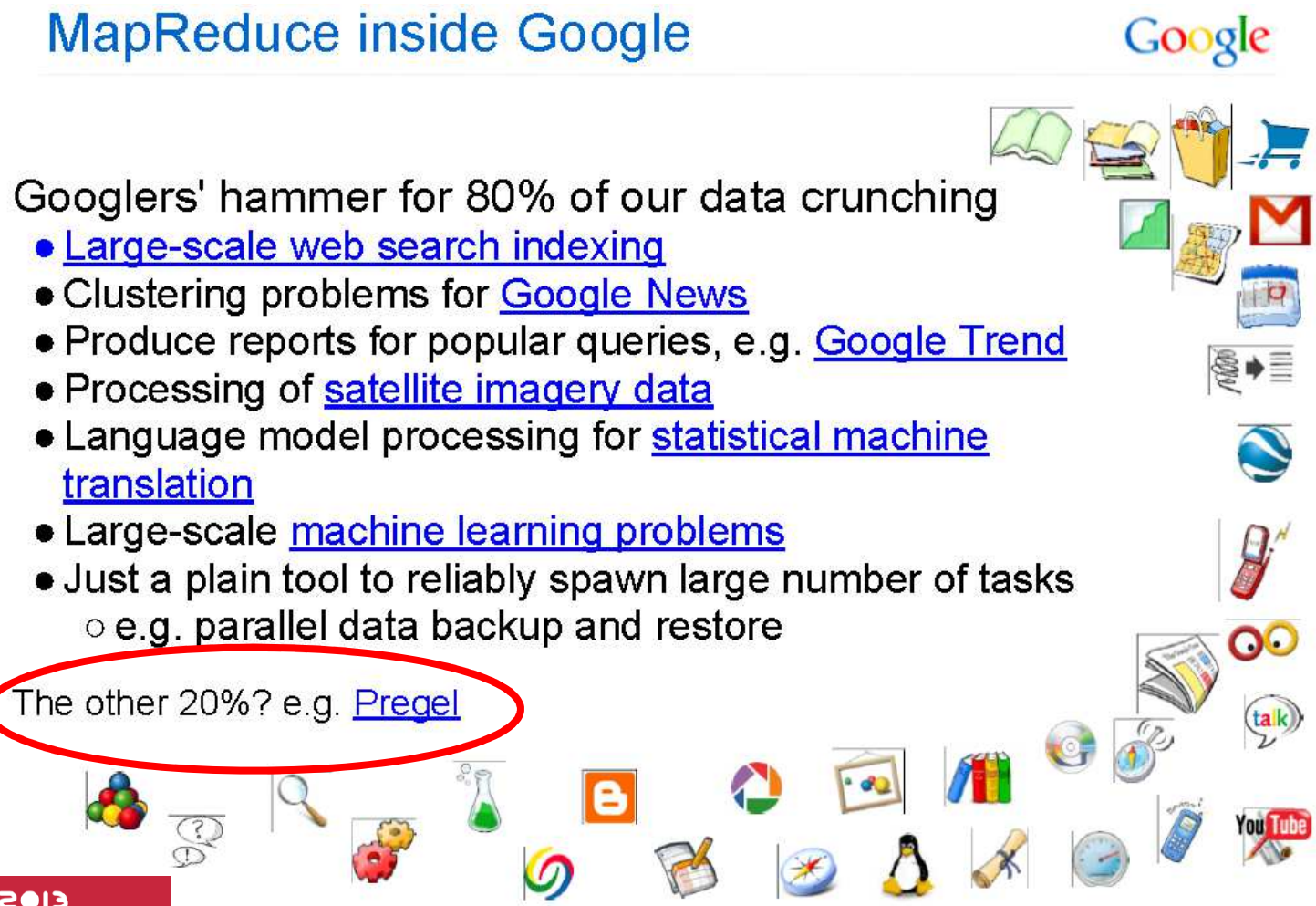
PrIter (University of Massachusetts Amherst, Northeastern University-China)
http://code.google.com/p/priter/
Cluster propios y Amazon EC2 cloud

HaLoop (University of Washington)
http://clue.cs.washington.edu/node/14
http://code.google.com/p/haloop/
Amazon's EC2

GPU based platforms
Mars
Grex

MapReduce

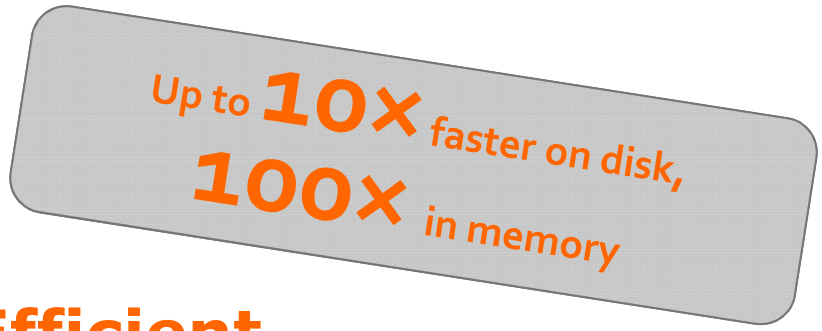
More than 10000 applications in Google

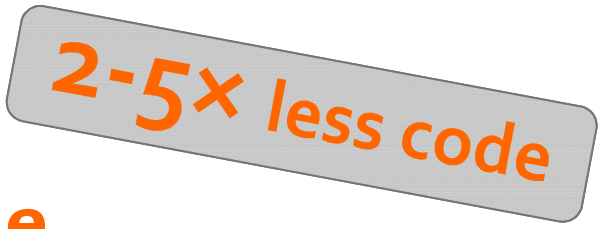


Enrique Alfonseca
Google Research Zurich

What is Spark?

Fast and Expressive Cluster Computing Engine Compatible with Apache Hadoop





Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

Spark birth

	Daytona
Gray	2013, 1.42 TB/min **Hadoop** 102.5 TB in 4,328 seconds 2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks) Thomas Graves Yahoo! Inc.

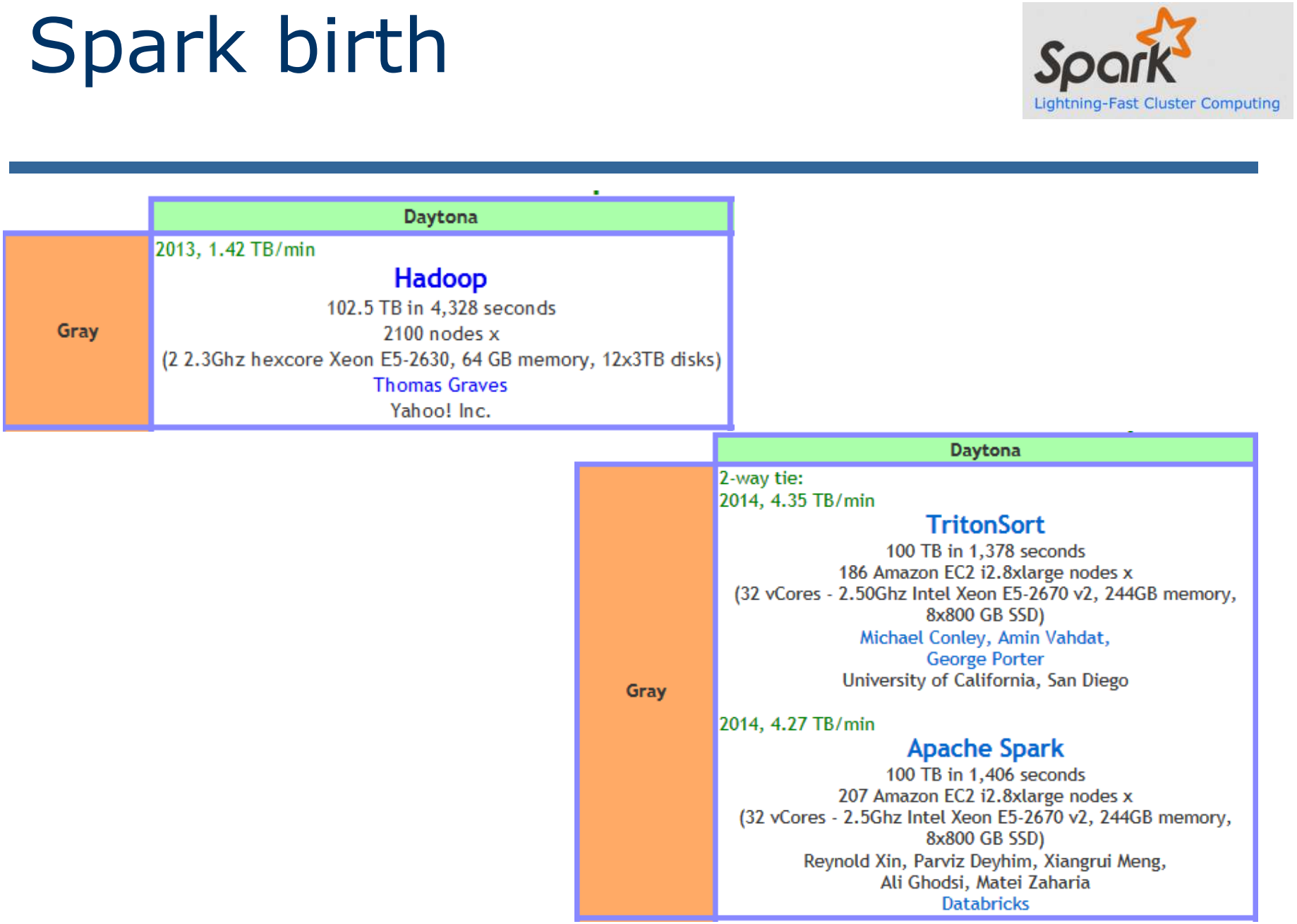
	Hadoop World Record	Spark 100 TB *
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
Rate	1.42 TB/min	4.27 TB/min

October 10, 2014

Using Spark on 206 EC2 nodes, we completed the benchmark in 23 minutes. This means that Spark sorted the same data 3X faster using 10X fewer machines. All the sorting took place on disk (HDFS), without using Spark's in-memory cache.

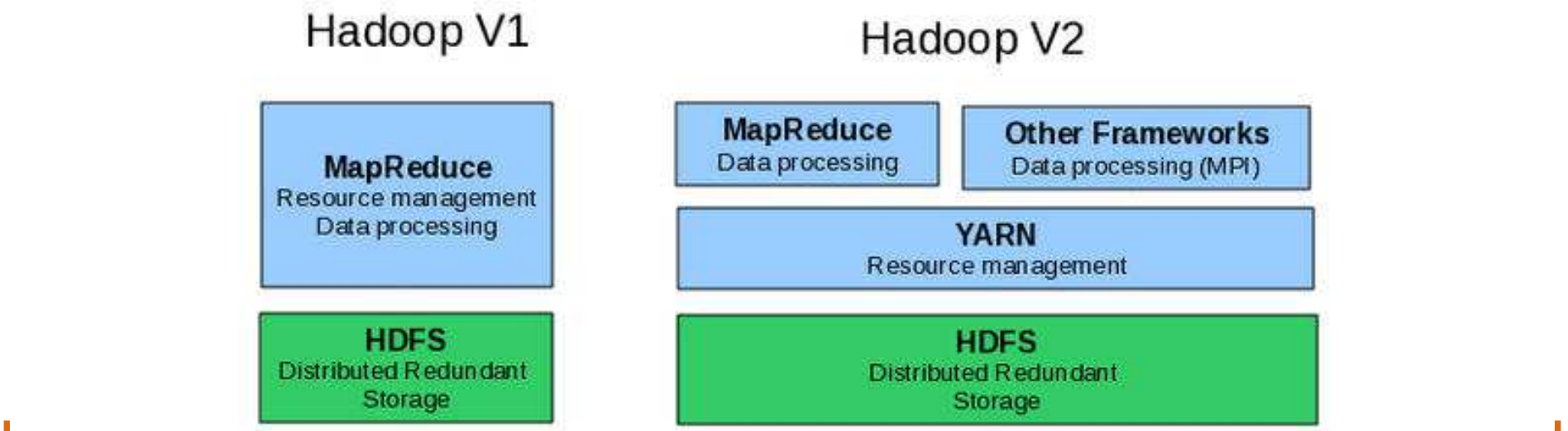
http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

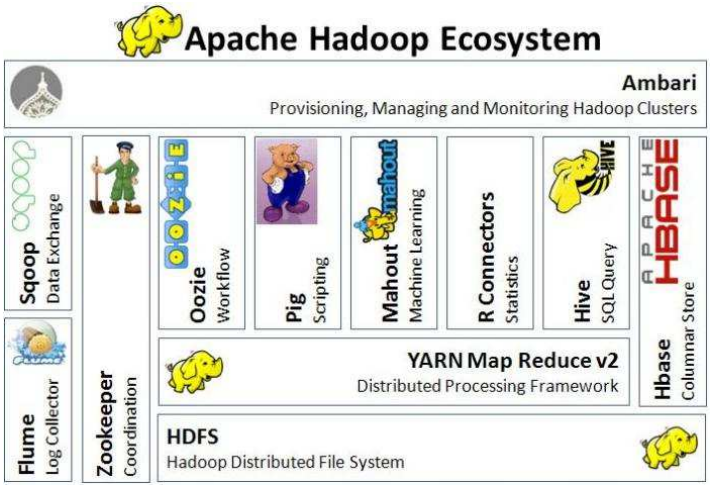
Spark birth

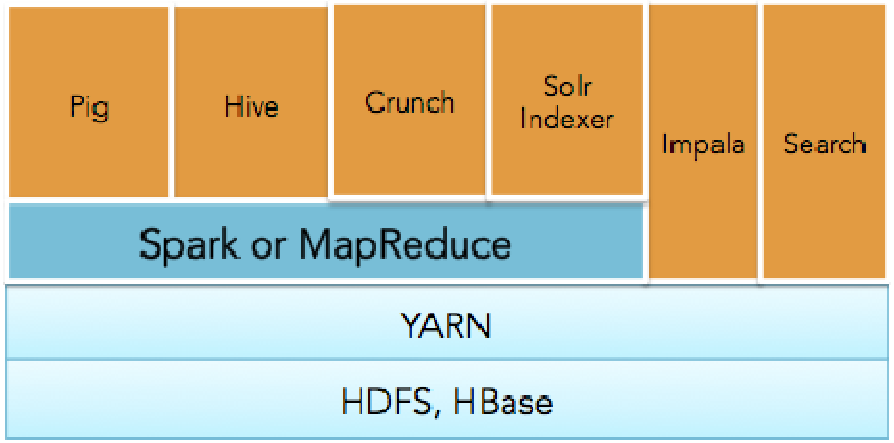


http://sortbenchmark.org/

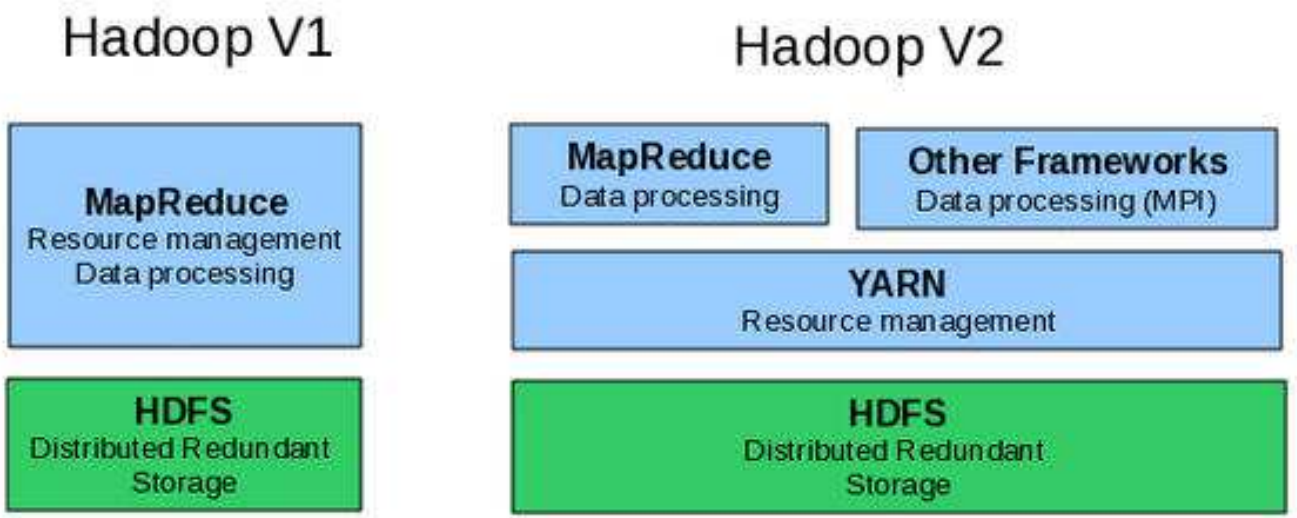
Apache Spark



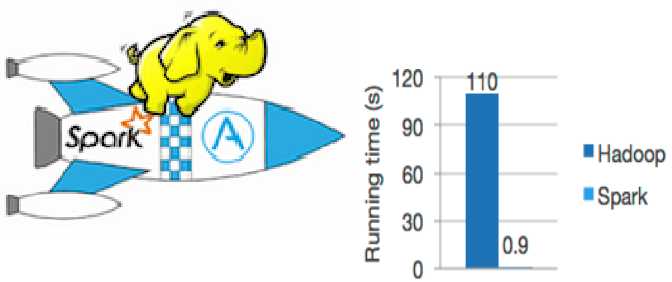




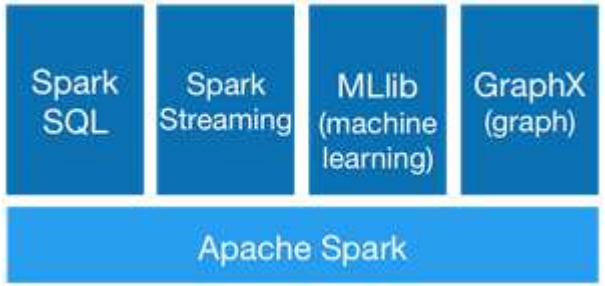
Apache Spark: InMemory



InMemory
HDFS Hadoop + SPARK



Ecosystem
Apache Spark



Future version of
Mahout for Spark

Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- **Retain the attractive properties of MapReduce**:
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

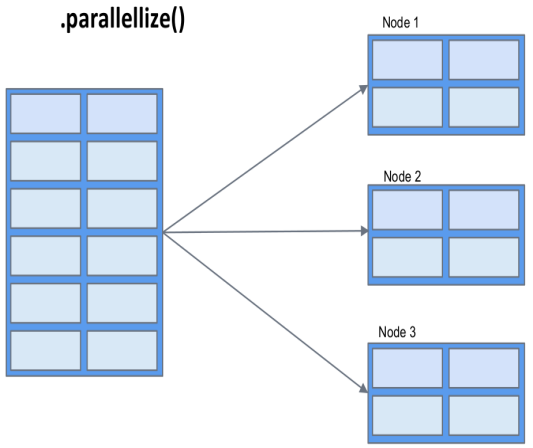
Solution: augment data flow model with "resilient distributed datasets" (RDDs)

RDDs in Detail

- An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs:
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, Hbase.
- *Can be cached for future reuse*

Creating a RDD: Parallelize() or External Datasets

```
// Parallelizing collections:
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)

// Parallelizing external datasets
val distFile = sc.textFile("data.txt")
```



- **Spark supports text files, SequenceFiles, and any other Hadoop InputFormat**

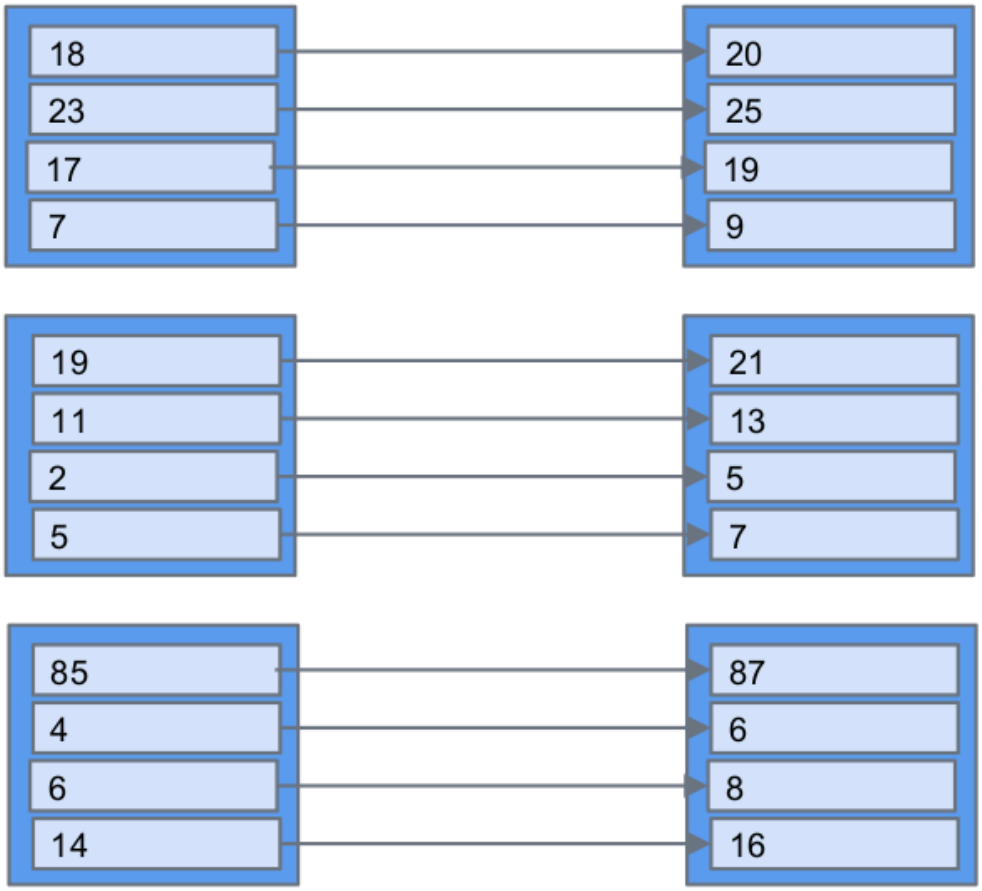
Operations with RDDs

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage

Transformations (define a new RDD)
map filter sample union groupByKey reduceByKey join cache …

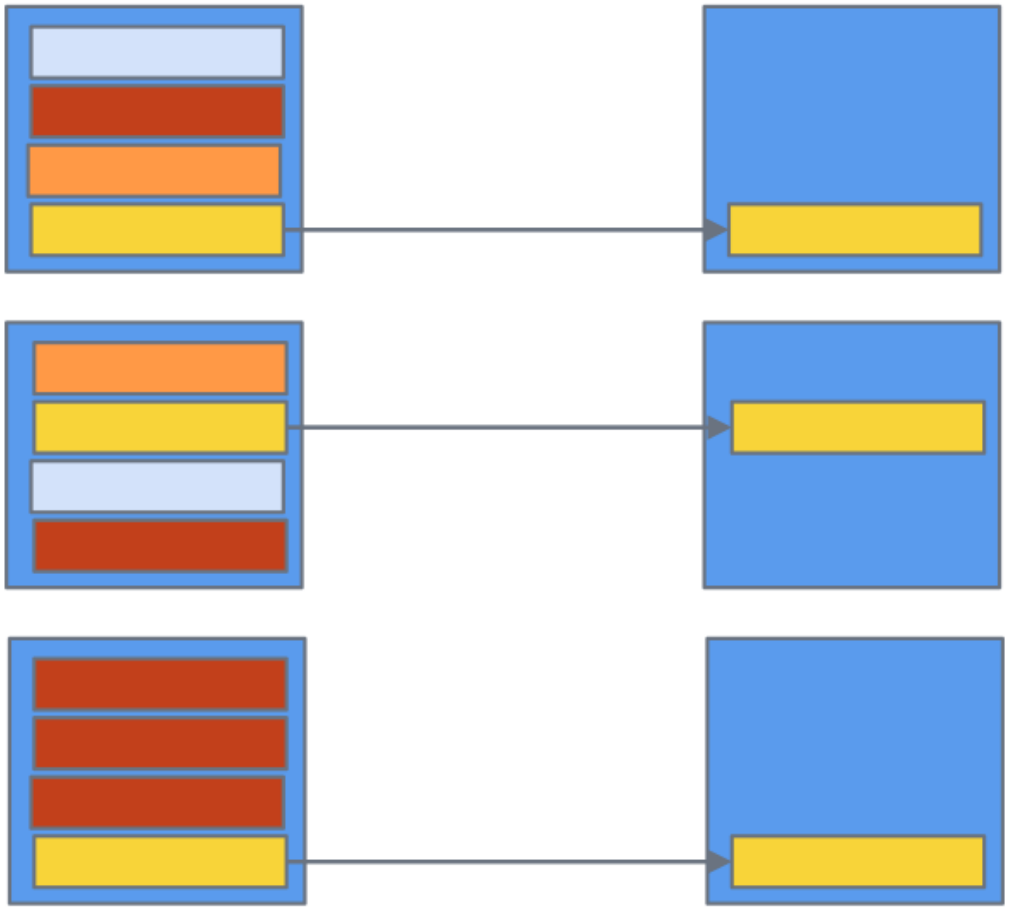
Parallel operations (return a result to driver)
reduce collect count save lookupKey …

Transformations: map(lambda x: x+2)

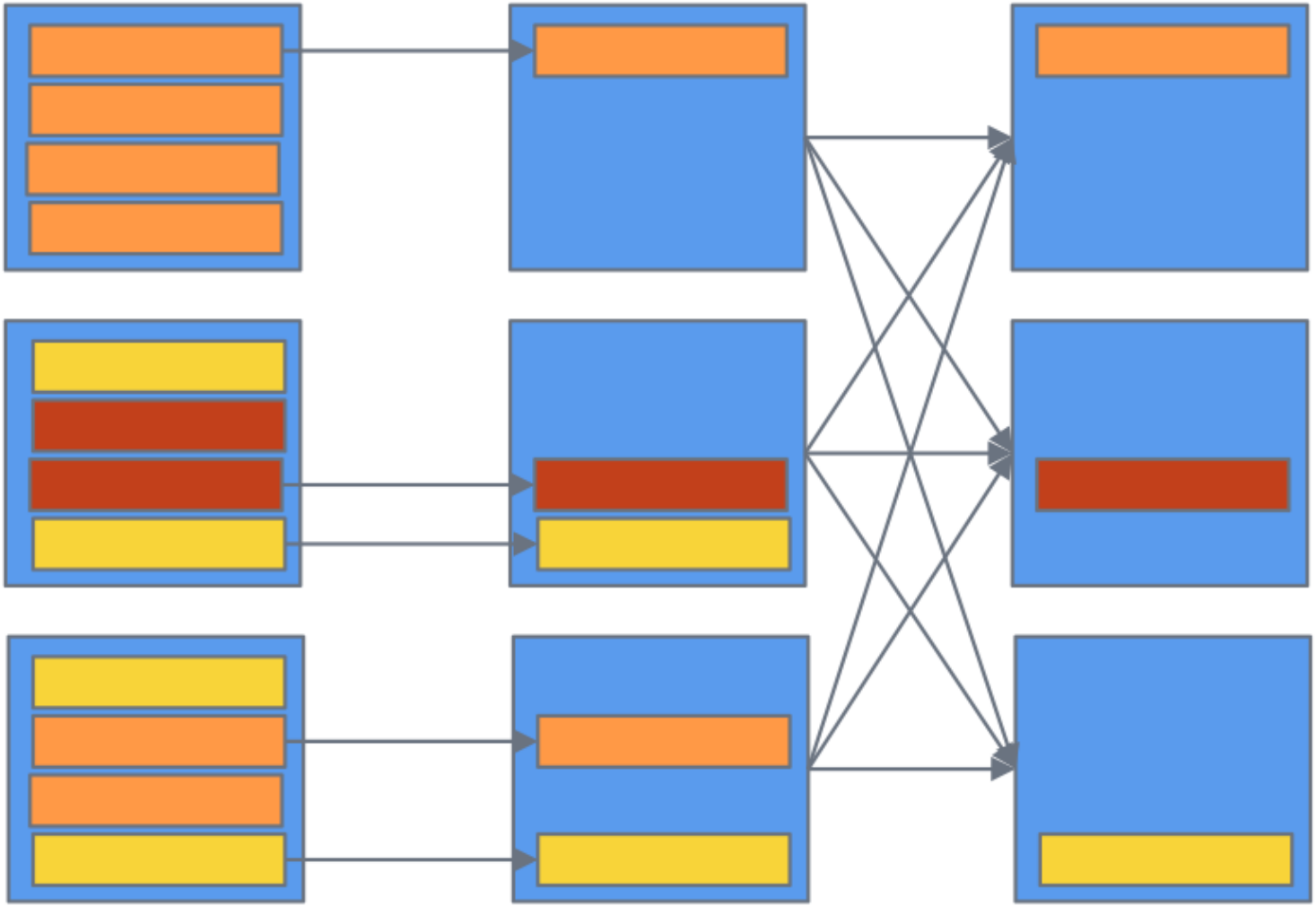


Source: Dirk Van den Poel. Spark: The new kid on the block

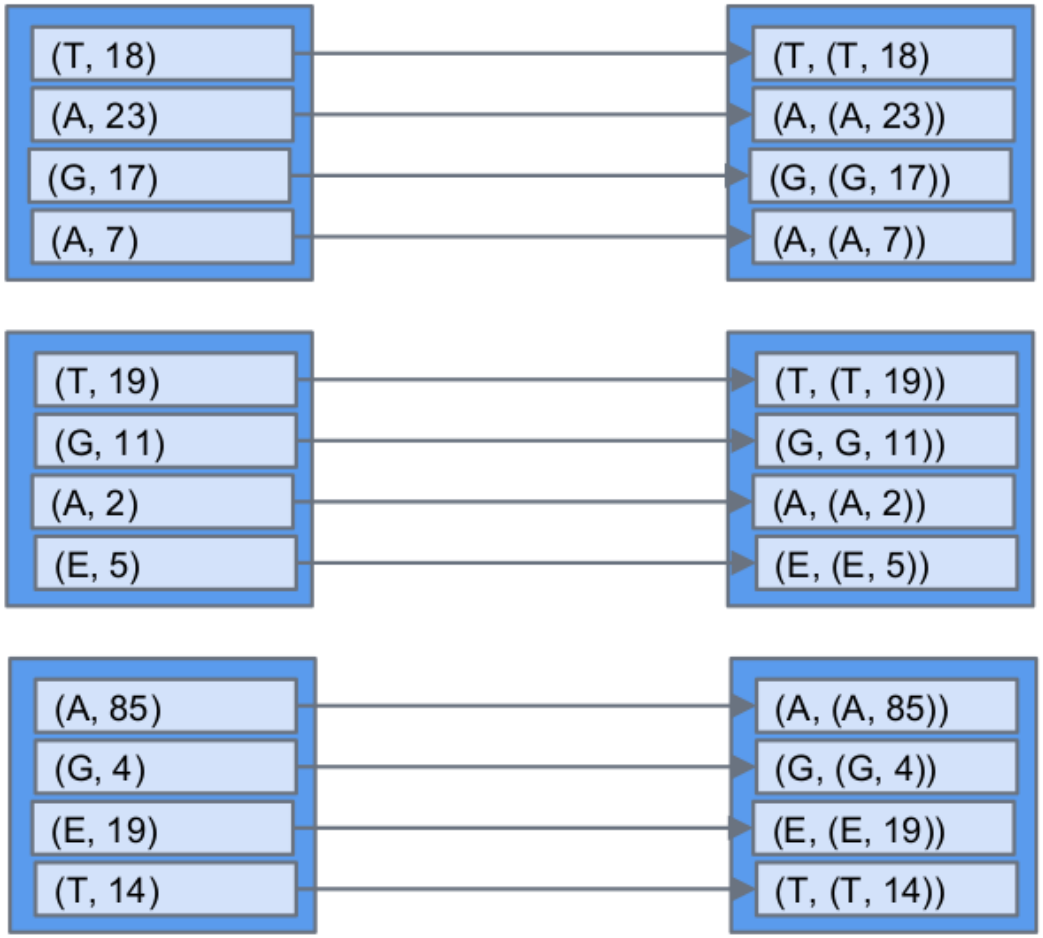
Transformations : filter(only yellow)



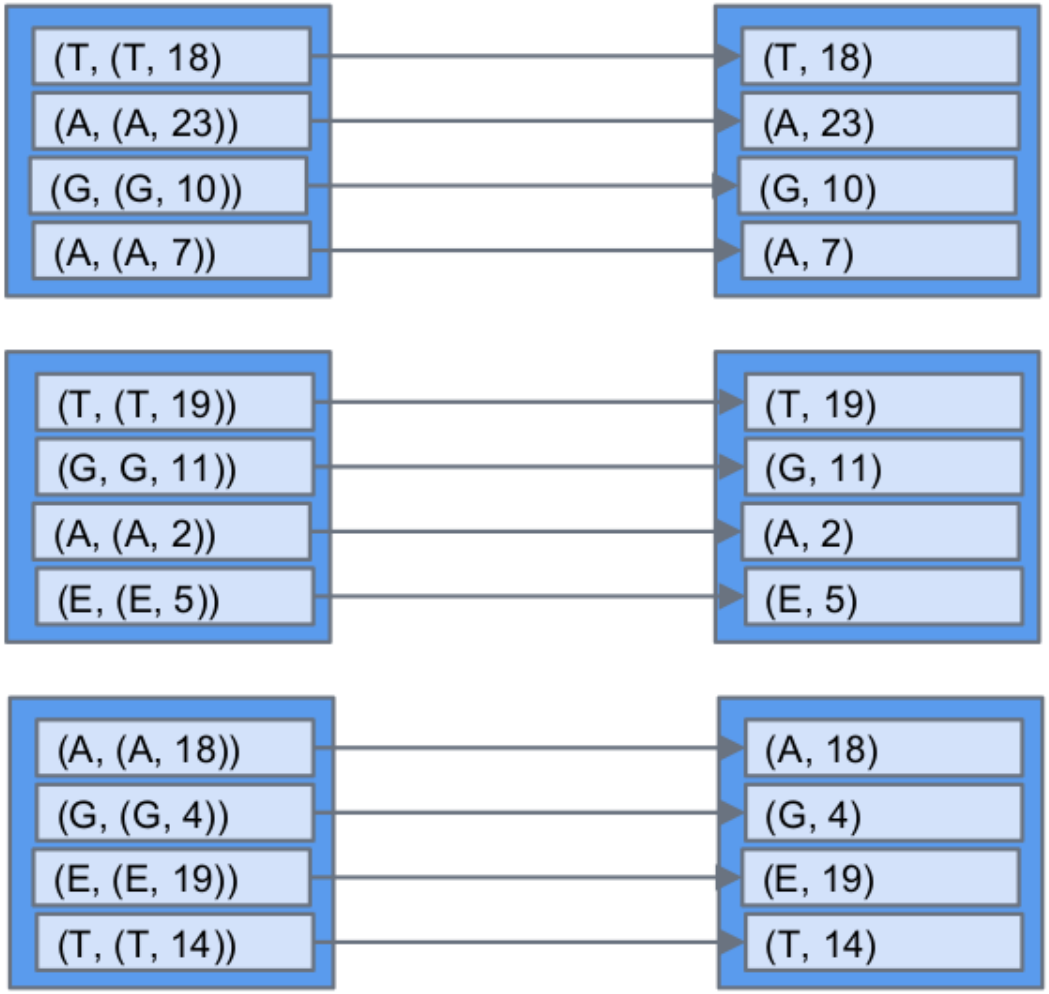
Transformations : distinct()



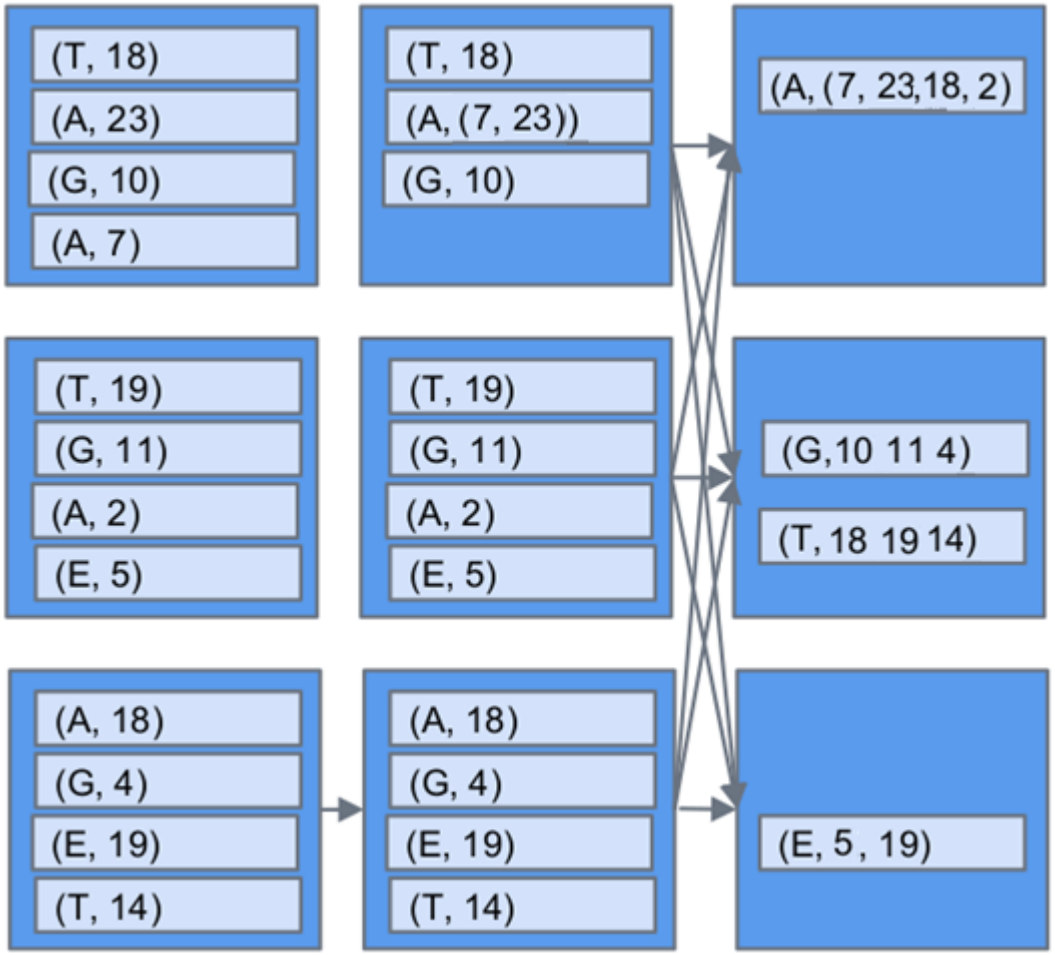
Transformations : keyBy(lambda x: x[0])



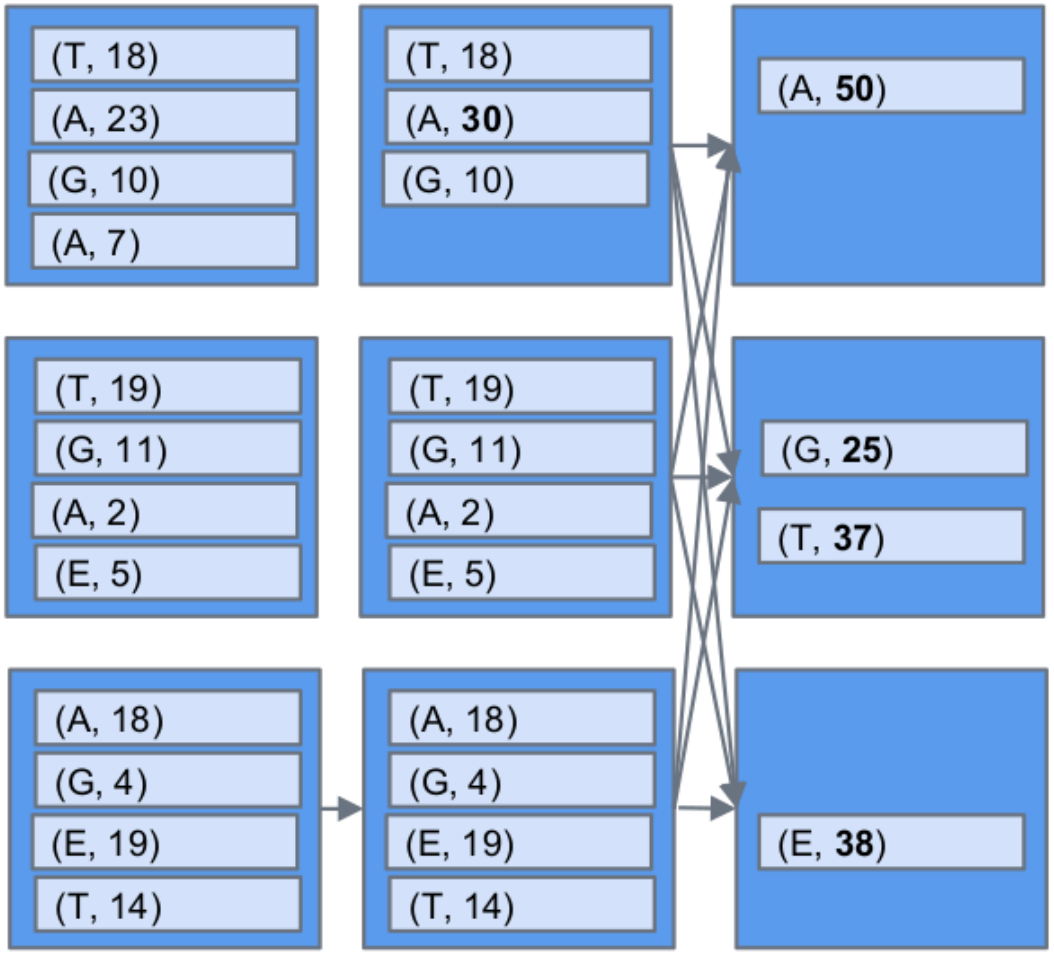
Transformations: mapValues(lambda x: x(1))



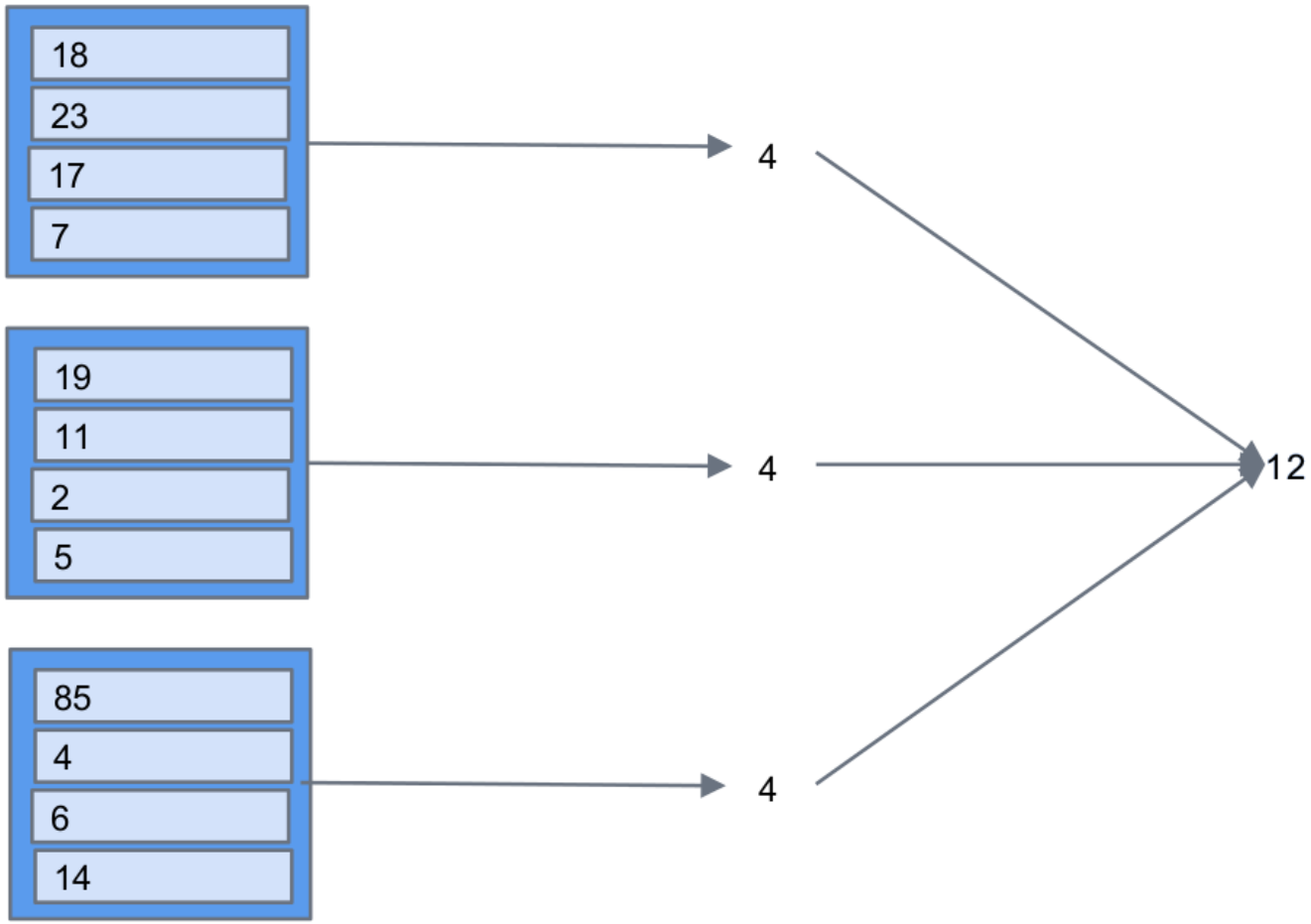
Transformations: groupByKey()



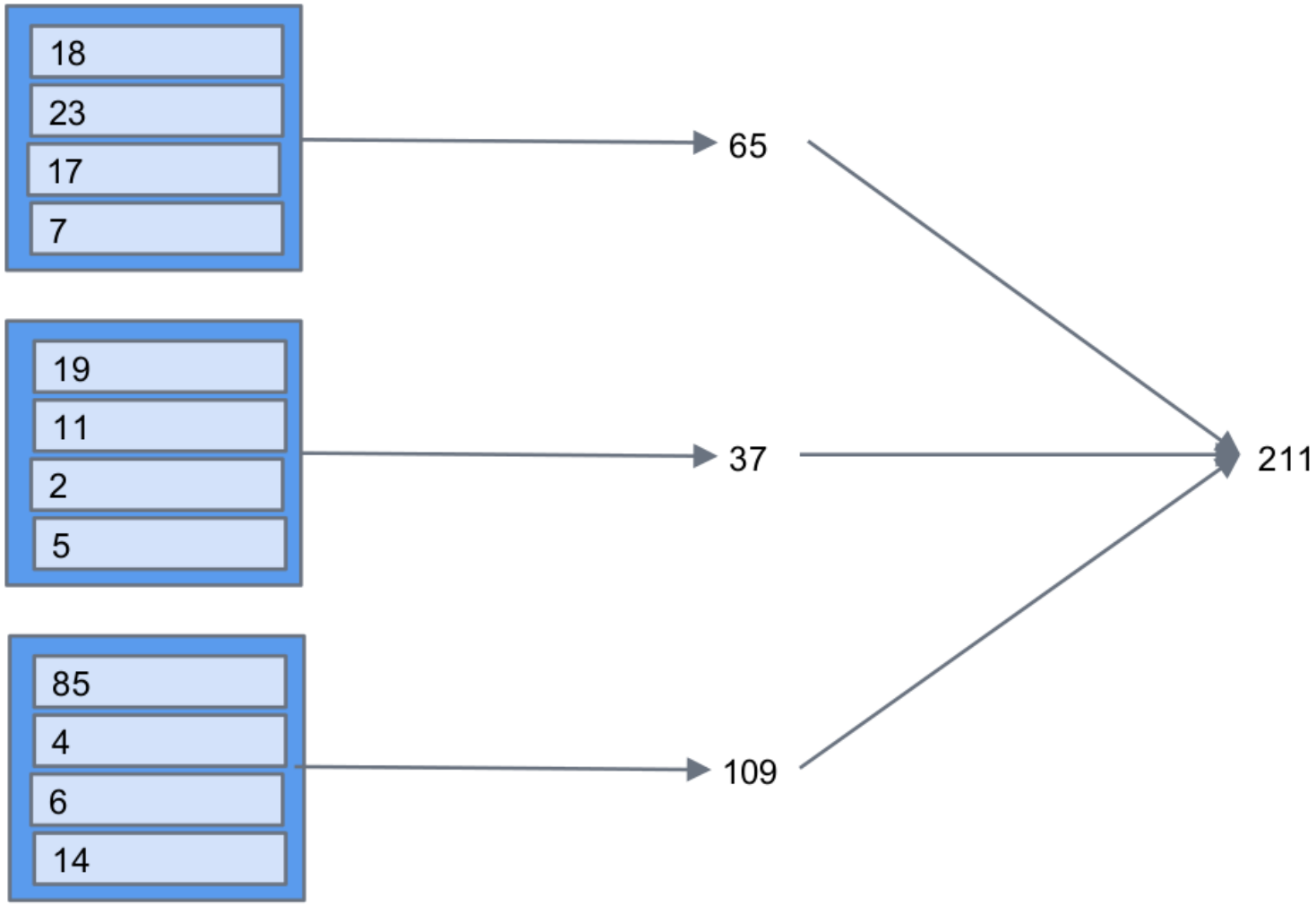
Transformations: reduceByKey(add)



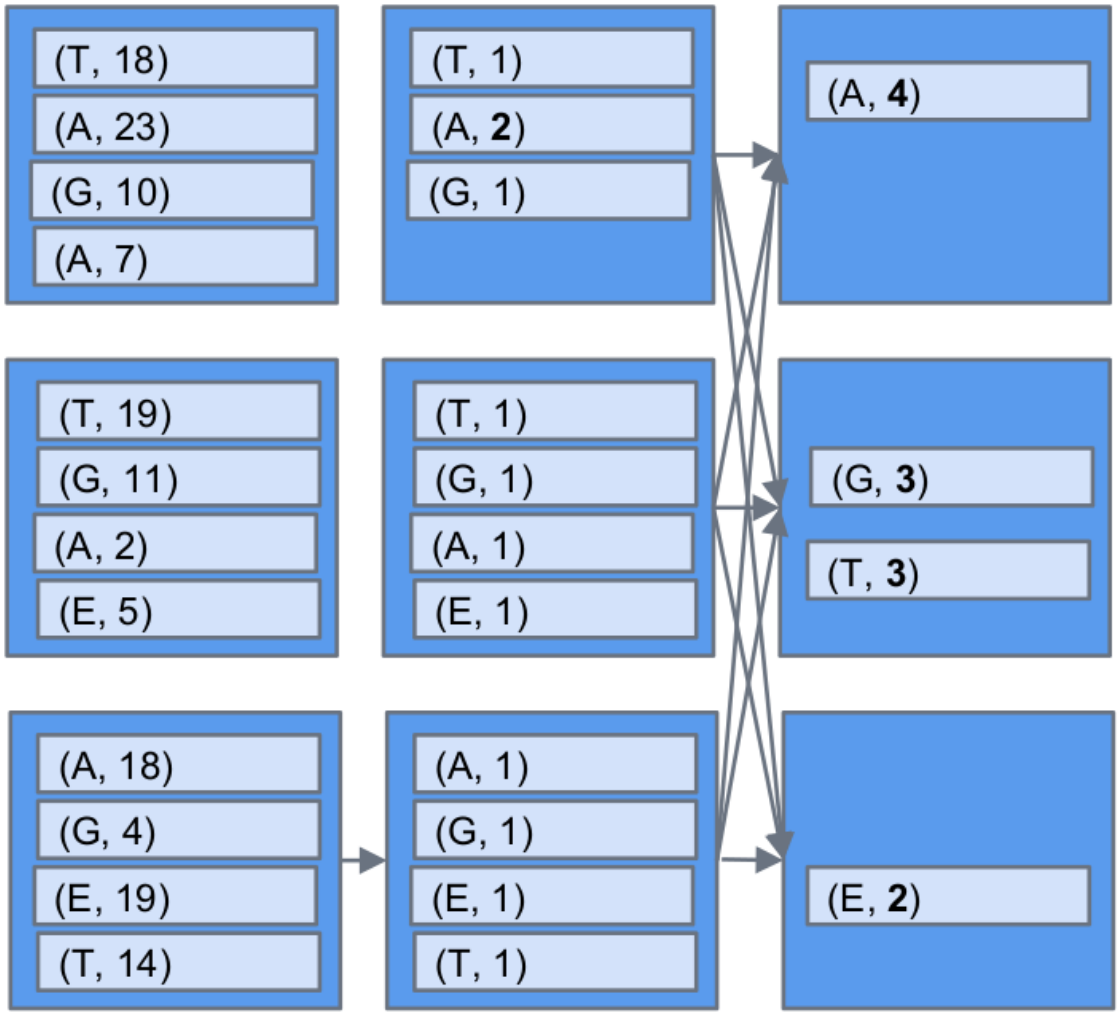
Actions: count()



Actions: reduce(add)



Actions: countByKey()



Apache Spark – other collections

DataFrames

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, 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.

The DataFrame API is available in Scala, Java, Python, and R.

Datasets

A Dataset is a new experimental interface added in Spark 1.6 that tries to provide the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.).

The unified Dataset API can be used both in Scala and Java. Python does not yet have support for the Dataset API, but due to its dynamic nature many of the benefits are already available (i.e. you can access the field of a row by name naturally `row.columnName`). Full python support will be added in a future release.

Zaharia-2012- Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I.
Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing.
In: *9th USENIX Conference on Networked Systems Design and Implementation, San Jose, CA, 2012,* 1–14.

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: “ERROR” in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x =>
x.contains(“ERROR”)).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>()
{
  Boolean call(String s) {
    return s.contains(“error”);
  }
}).count();
```

Standalone Programs

- Python, Scala, & Java

Interactive Shells

- Python & Scala

Performance

- Java & Scala are faster due to static typing
- …but Python is often fine

Learning Spark: Interactive mode

- **Easiest way**: the shell (*spark-shell* or *pyspark*)
  - Special Scala / Python interpreters for cluster use

- It runs as an application on an existing Spark Cluster…

```
spark-shell --master=masterNODE       #  cluster
```

- OR Can run locally

```
MASTER=local     ./spark-shell    # local, 1 thread
MASTER=local[2] ./spark-shell    # local, 2 threads
```

Learning Spark: Standalone mode

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “WordCount”, sys.argv[0],
None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(“ ”)) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```

SparkContext (1)

- Main entry point to Spark functionality
- Created for you in `spark-shell` as variable `sc`
- The first thing a Spark program must do is to create a SparkContext object, which tells Spark how to access a cluster. To create a SparkContext you first need to build a SparkConf object that contains information about your application.

SparkContext (2)

- Only one SparkContext may be active per JVM. You must stop() the active SparkContext before creating a new one.

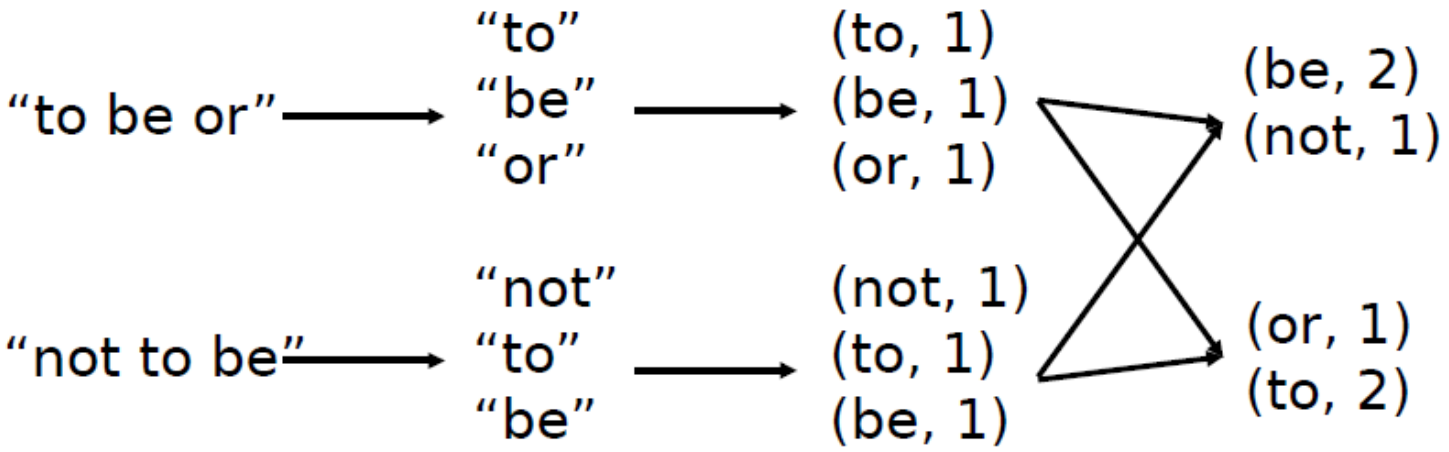
```
val conf = new
  SparkConf().setAppName(appName).setMaster(master)
```

- The appName parameter is a name for your application to show on the cluster.

- **master** is a Spark, Mesos or YARN cluster URL, or a special “local” string to run in local mode.

The WordCount example with Spark

```
lines = sc.textFile(“hamlet.txt”)
counts = lines.flatMap(lambda line: line.split(“ ”))
              .map(lambda word => (word, 1))
              .reduceByKey(lambda x, y: x + y)
```

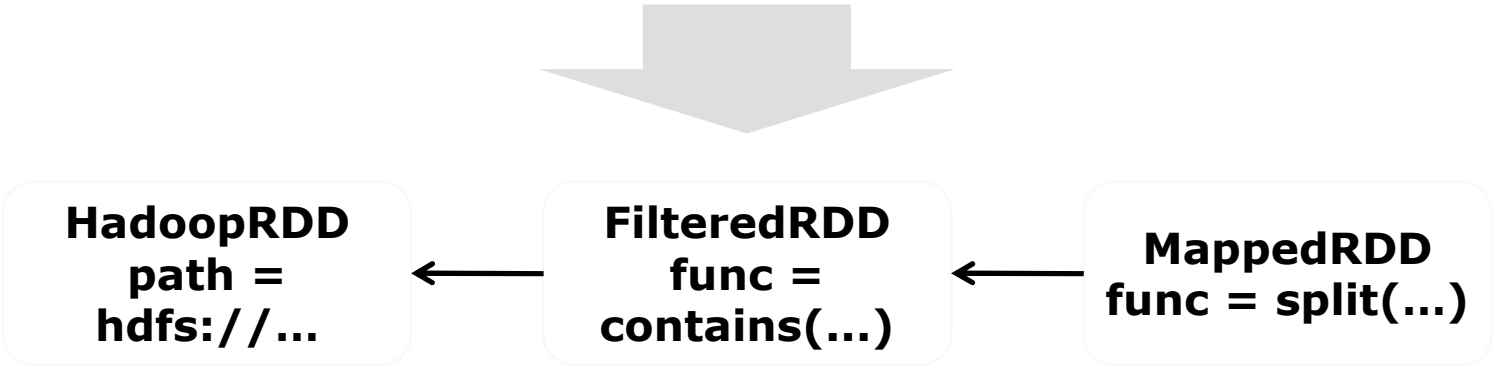


RDD Fault Tolerance

RDDs track the transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(lambda s: s.contains("ERROR"))
                        .map(lambda s: s.split('\t')[2])
```



Spark in Java and Scala

Java API:

```
JavaRDD<String> lines =
spark.textFile(…);

errors = lines.filter(
  new Function<String, Boolean>() {
    public Boolean call(String s) {
      return s.contains(“ERROR”);
    }
});

errors.count()
```

Scala API:

```
val lines = spark.textFile(…)

errors = lines.filter(s =>
s.contains(“ERROR”))
// can also write
filter(_.contains(“ERROR”))

errors.count
```

Which Language Should I Use?

- Standalone programs can be written in any, ***but console is only Python & Scala***
- **Python developers:** can stay with Python for both
- **Java developers:** consider using Scala for console (to learn the API)

- Performance: Java / Scala will be faster (statically typed), but Python can do well for numerical work with NumPy

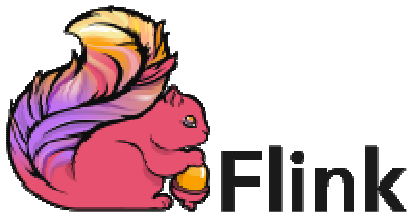
Spark and RDDs: Summary

Advantages:

- Spark RDDs → efficient data sharing
- In-memory caching accelerates performance
  - Up to 20x faster than Hadoop
- Easy to use high-level programming interface
  - Express complex algorithms ~100 lines.

Flink

https://flink.apache.org/



Apache Flink is an open source platform for distributed stream and batch data processing.

Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams.

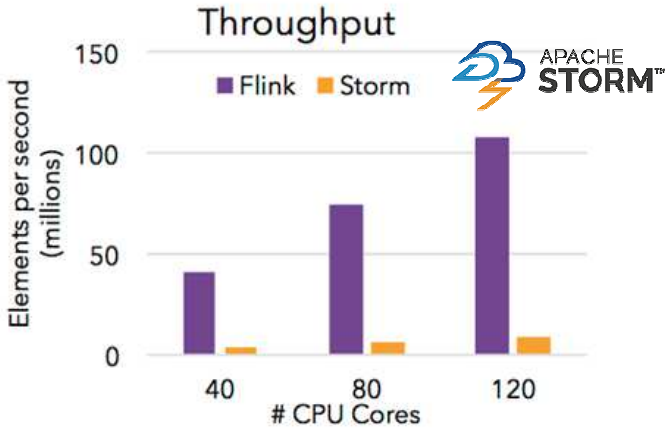
Flink includes **several APIs** for creating applications that use the Flink engine:

1. DataStream API for unbounded streams embedded in Java and Scala, and
2. DataSet API for static data embedded in Java, Scala, and Python,
3. Table API with a SQL-like expression language embedded in Java and Scala.

Flink also bundles **libraries for domain-specific use cases**:

1. CEP, a complex event processing library,
2. Machine Learning library, and
3. Gelly, a graph processing API and library.

You can **integrate** Flink easily with other well-known open source systems both for data input and output as well as deployment.



Batch on Streaming

Batch processing applications run efficiently as special cases of stream processing applications.

Streaming First

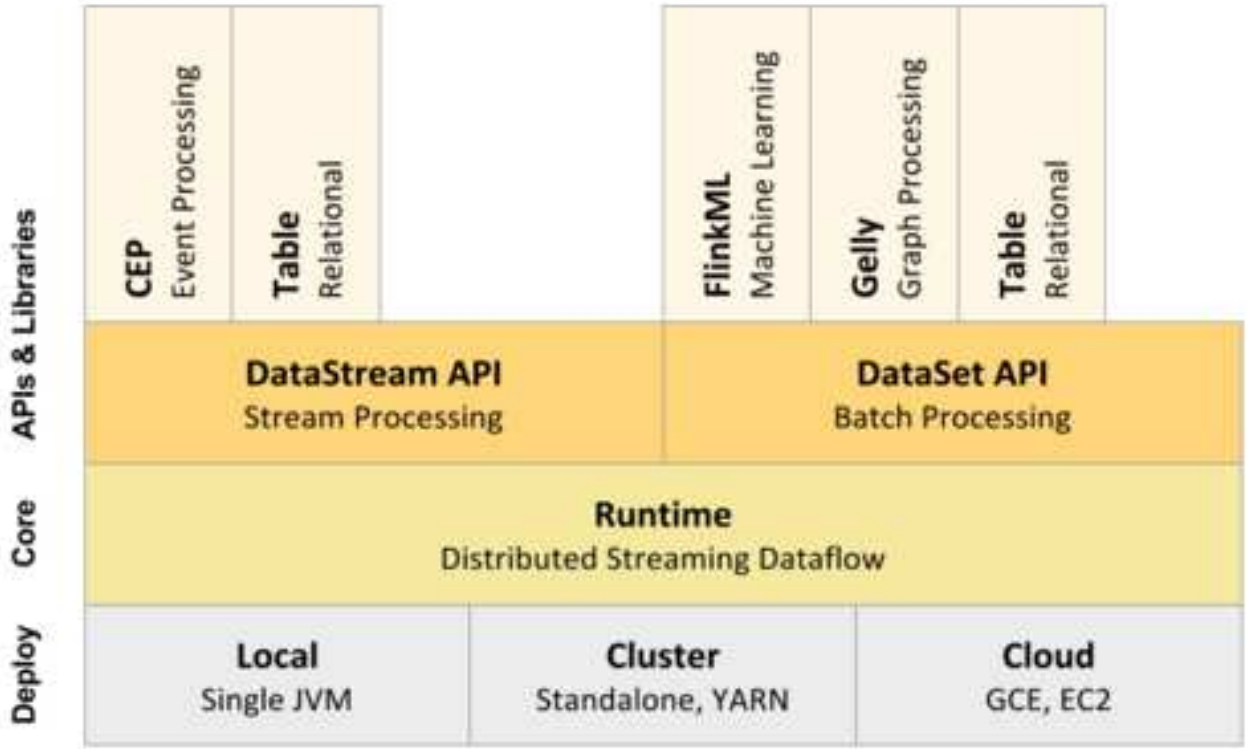
High throughput and low latency stream processing with exactly-once guarantees.

APIs, Libraries, and Ecosystem

DataSet, DataStream, and more. Integrated with the Apache Big Data stack.

Flink

https://flink.apache.org/



2001-2010

2010-2015

Big Data: Technology and Chronology

The Apache Software Foundation

BIG DATA

2001

3V's Gartner

Doug Laney

Google

Hadoop Ecosystem

2009-2013 Flink

TU Berlin

Flink Apache (Dec. 2014) Volker Markl

2004

MapReduce

Google

Jeffrey Dean

Flink

Big Data

YAHOO!

hadoop

2008

Hadoop

Yahoo!

Doug Cutting

2010 Spark

U Berckeley

Apache Spark Feb. 2014

Matei Zaharia

Spark

Lightning-Fast Cluster Computing

2010-2015:

Big Data Analytics: Mahout, MLLib, …

Hadoop Ecosystem

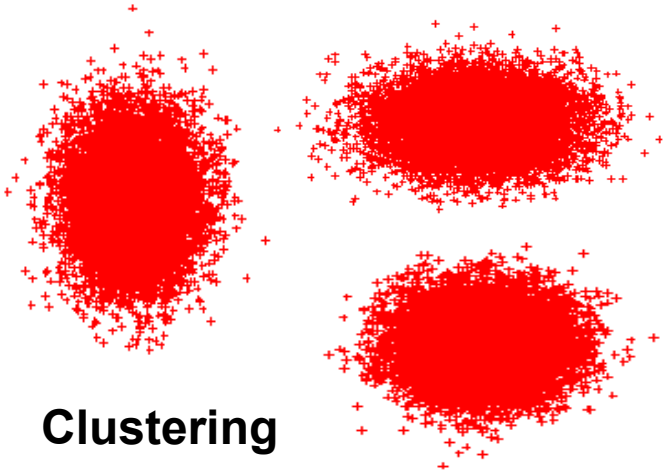
Applications New Technology

Outline

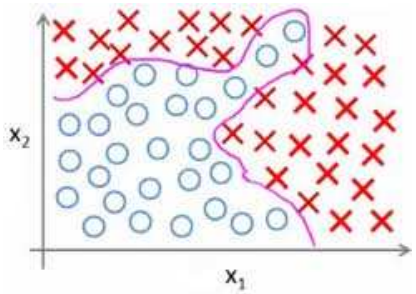
- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Big Data Analytics

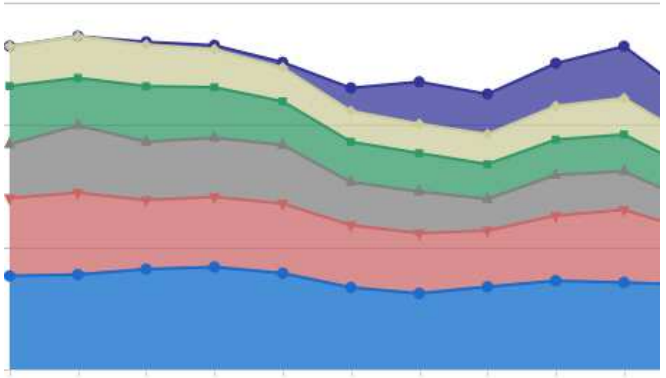
Potential scenarios:



Clustering

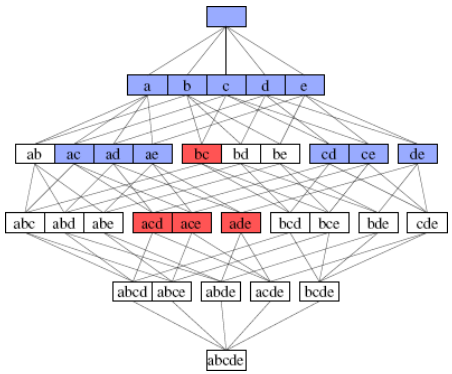


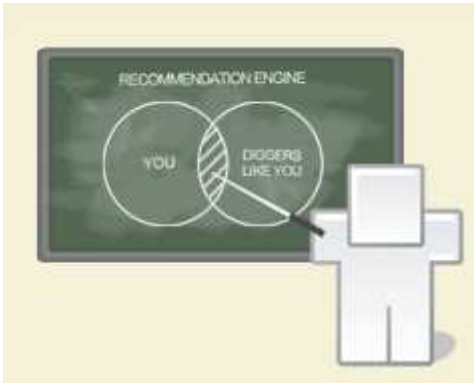
Classification



Real Time Analytics/
Big Data Streams

Association





Recommendation
Systems

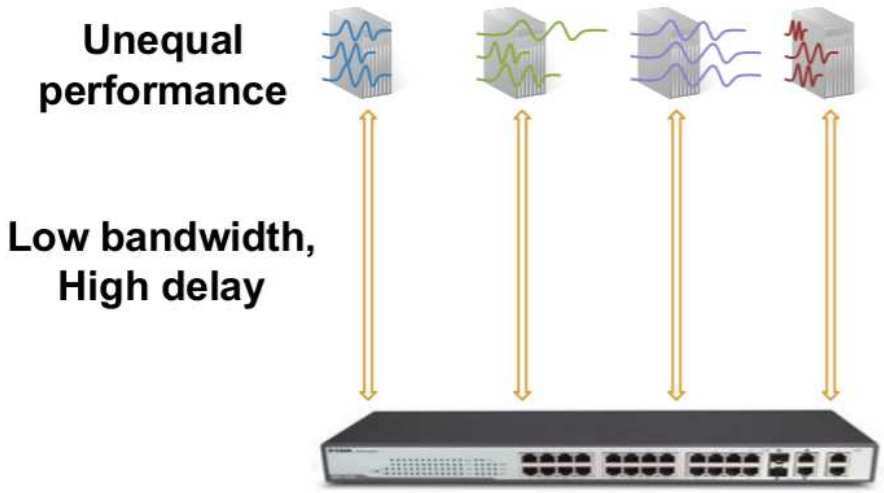
Social Media Mining
Social Big Data

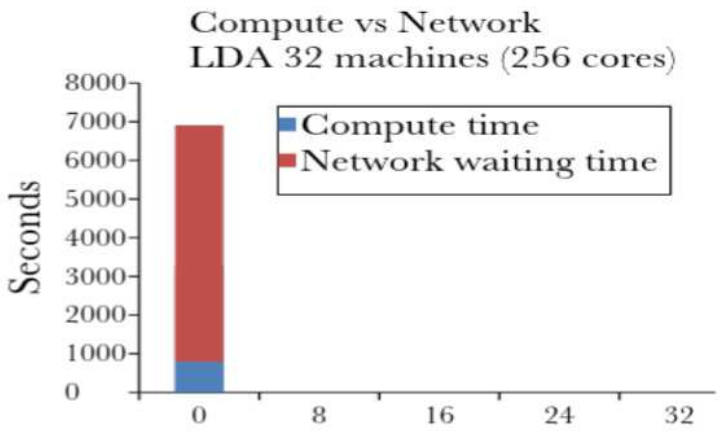
Machine learning for Big Data

- Data mining techniques have demonstrated to be very useful tools to extract new valuable knowledge from data.
- The knowledge extraction process from big data has become a very difficult task for most of the **classical** and **advanced** data mining tools.
- The **main challenges** are to deal with:
  - The increasing scale of data
    - at the level of **instances**
    - at the level of **features**
  - The **complexity** of the problem.
  - And many other points

Problems: No ideal Distributed System

- Two distributed challenges for ML:
  - Networks are slow
  - "Identical" machines rarely perform equally





Why do we need new Big ML systems?

ML practitioner's view

- Want correctness, fewer iters to converge
- … but assume an ideal system

```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```

- Oversimplify systems issues
  - e.g. machines perform consistently
  - e.g. can sync parameters any time

Systems view

- Want more iters executed per second
- … but assume ML algo is a black box
- … or assume ML algo "still works" under different execution models

Slow-but-correct
Bulk Sync. Parallel

Fast-but-unstable
Asynchronous Parallel

- Oversimplify ML issues
  - e.g. assume ML algo "works" without proof
  - e.g. ML algo "easy to rewrite" in chosen abstraction: MapR, vertex program, etc.

Why do we need new Big ML systems?

ML practitioner's view

- Want correctness, fewer iters to converge
- … but assume

```
for (t = 1 t
  doThings()
  paralle
  doOtherThi
}
```

Systems view

- Want more iters executed per second
- lack box
- orks" under

nstable

Asynchronous Parallel

Alone, neither side has full picture…

New opportunities exist in the middle

- Oversimplify systems issues
  - e.g. machines perform consistently
  - e.g. can sync parameters any time

- Oversimplify ML issues
  - e.g. assume ML algo "works" without proof
  - e.g. ML algo "easy to rewrite" in chosen abstraction: MapR, vertex program, etc.

The Science of ML for Big Data

- Apart from Hadoop and Spark, there are other frameworks and programming paradigms such as:
  - Giraph
  - GraphLab
  - Petuum
- Each one has distinct technical innovations

- **Key insight**: ML algorithms have special properties
  - Error-tolerance, dependency structures, uneven convergence
  - How to utilize for faster data/model-parallelism?

Big Data Analytics: A 3 generational view

Generation	1st Generation	2nd Generation	3rd Generation
Examples	SAS, R, Weka, SPSS, KEEL	Mahout, Pentaho, Cascading	Spark, Haloop, GraphLab, Pregel, Giraph, ML over Storm
Scalability	Vertical	Horizontal (over Hadoop)	Horizontal (Beyond Hadoop)
Algorithms Available	Huge collection of algorithms	Small subset: sequential logistic regression, linear SVMs, Stochastic Gradient Descendent, k-means clustering, Random forest, etc.	Much wider: CGD, ALS, collaborative filtering, kernel SVM, matrix factorization, Gibbs sampling, etc.
Algorithms Not Available	Practically nothing	Vast no.: Kernel SVMs, Multivariate Logistic Regression, Conjugate Gradient Descendent, ALS, etc.	Multivariate logistic regression in general form, k-means clustering, etc. – Work in progress to expand the set of available algorithms
Fault-Tolerance	Single point of failure	Most tools are FT, as they are built on top of Hadoop	FT: HaLoop, Spark Not FT: Pregel, GraphLab, Giraph

Machine learning interfaces for Hadoop

- Based on pure MapReduce:
  - Mahout 0.9
- Beyond MapReduce:
  - Mahout (>0.9)
  - MLlib
  - GraphLab
  - …

MLlib

&

Mahout 0.9

- **What is Mahout?**
  - The starting place (2009) for MapReduce-based ML algorithms.
  - ***Goal***: delivering scalable machine learning algorithm implementations
- **Why Mahout?**
  - Many open ML
    - Lack Commu
    - Lack Scalabi
- **Who use Mah**
  - Adobe
  - AOL
  - Twitter

Mahout

History

25 July 2013 - Apache Mahout 0.8 released

Visit our release notes page for details.

16 June 2012 - Apache Mahout 0.7 released

Visit our release notes page for details.

6 Feb 2012 - Apache Mahout 0.6 released

Visit our release notes page for details.

9 Oct 2011 - Mahout in Action released

The book Mahout in Action is available in print. Sean Owen, Robin Anil, Ted Dunning and Ellen Friedman thank the community (especially those who were reviewers) for input during the process and hope it is enjoyable.

Find it at your favorite bookstore, or order print and eBook copies from Manning -- use discount code "mahout37" for 37% off.

Mahout

History

1 February 2014 - Apache Mahout 0.9 released

Apache Mahout has reached version 0.9. All developers are encouraged to begin using version 0.9. Highlights include:

- New and improved Mahout website based on Apache CMS - MAHOUT-1245
- Early implementation of a Multi Layer Perceptron (MLP) classifier - MAHOUT-1265
- Scala DSL Bindings for Mahout Math Linear Algebra. See this blogpost and MAHOUT-1297
- Recommenders as Search. See [https://github.com/pferrel/solr-recommender] and MAHOUT-1288
- Support for easy functional Matrix views and derivatives - MAHOUT-1300
- JSON output format for ClusterDumper - MAHOUT-1343
- Enabled randomised testing for all Mahout modules using Carrot RandomizedRunner - MAHOUT-1345
- Online Algorithm for computing accurate Quantiles using 1-dimensional Clustering - See this pdf and MAHOUT-1361
- Upgrade to Lucene 4.6.1 - MAHOUT-1364

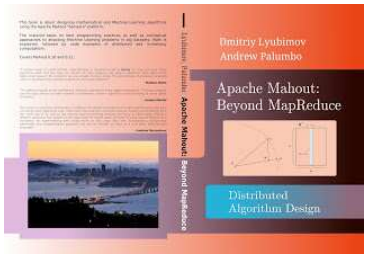
Changes in 0.9 are detailed in the release notes.

The following algorithms that were marked deprecated in 0.8 have been removed in 0.9:

- Switched LDA implementation from Gibbs Sampling to Collapsed Variational Bayes
- Meanshift - removed due to lack of actual usage and support
- MinHash - removed due to lack of actual usage and support
- Winnow - removed due to lack of actual usage and support
- Perceptron - removed due to lack of actual usage and support
- Slope One - removed due to lack of actual usage
- Distributed Pseudo recommender - removed due to lack of actual usage
- TreeClusteringRecommender - removed due to lack of actual usage

Mahout

History



12 March 2016 - Apache Mahout 0.11.2 released¶

23 February 2016 - New Apache Mahout Book - "Apache Mahout: Beyond MapReduce" by D.Lyubimov and A.Palumbo released. See Mahout "Samsara" Book Is Out

Mahout News

25 April 2014 - Goodbye MapReduce

The Mahout community decided to move its codebase onto modern data processing systems that offer a richer programming model and more efficient execution than Hadoop MapReduce. **Mahout will therefore reject new MapReduce algorithm implementations from now on**. We will however keep our widely used MapReduce algorithms in the codebase and maintain them.

We are building our future implementations on top of a DSL for linear algebraic operations which has been developed over the last months. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark.

Furthermore, there is an experimental contribution undergoing which aims to integrate the h20 platform into Mahout.

Mahout

Algorithms

A good library to use pure MapReduce applications under Hadoop

	Single Machine	MapReduce	Spark	H2O	Flink
Mahout Math-Scala Core Library and Scala DSL					
Mahout Distributed BLAS. Distributed Row Matrix API with R and Matlab like operators. Distributed ALS, SPCA, SSVD, thin-QR. Similarity Analysis.			x	x	*in development*
Mahout Interactive Shell					
Interactive REPL shell for Spark optimized Mahout DSL			x		
Collaborative Filtering					
User-Based Collaborative Filtering	x		x		
Item-Based Collaborative Filtering	x	x	x		
Matrix Factorization with ALS	x	x			
Matrix Factorization with ALS on Implicit Feedback	x	x			
Weighted Matrix Factorization, SVD++	x				
Classification					
Logistic Regression - trained via SGD	x				
Naive Bayes / Complementary Naive Bayes		x	x		
Random Forest		x			
Hidden Markov Models	x				
Multilayer Perceptron	x				

Mahout

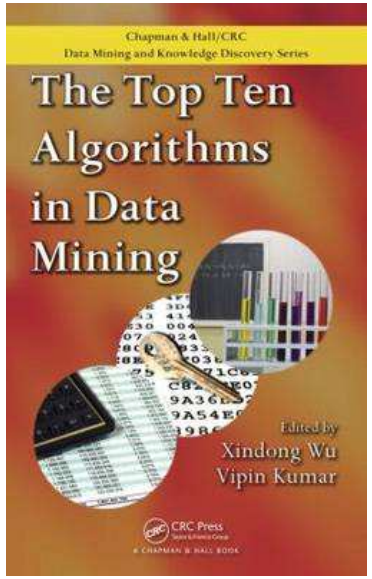
Algorithms

A good library to use pure MapReduce applications under Hadoop

	Single Machine	MapReduce	Spark	H2O	Flink
Mahout Math-Scala Core Library and Scala DSL					
Mahout Distributed BLAS. Distributed Row Matrix API with R and Matlab like operators. Distributed ALS, SPCA, SSVD, thin-QR. Similarity Analysis.			x	x	*in development*
Clustering					
Canopy Clustering	*deprecated*	*deprecated*			
k-Means Clustering	x	x			
Fuzzy k-Means	x	x			
Streaming k-Means	x	x			
Spectral Clustering		x			
Dimensionality Reduction *note: most scala-based dimensionality reduction algorithms are available through the* Mahout Math-Scala Core Library for all engines					
Singular Value Decomposition	x	x	x	x	
Lanczos Algorithm	*deprecated*	*deprecated*			
Stochastic SVD	x	x	x	x	
PCA (via Stochastic SVD)	x	x	x	x	
QR Decomposition	x	x	x	x	

Hadoop Mahout

What kind of algorithms can be found in Mahout?



Decision trees (C4.5, Cart)(MReC4.5)
K-Means (is a good implementation?)
SVM
Apriori
kNN
Naïve Bayes
EM (Expectation Maximization)
PageRank
Adaboost

Not available!

MapReduce limitations

Iterative algorithms!

Mahout

Version 0.11.2

- **Apache Mahout introduces a new math environment called Samsara,** including new implementations built for speed on Mahout-Samsara [Spark]
- They run on Spark 1.3+ and some on H2O, which means as much as a 10x speed increase.
- You'll find robust matrix decomposition algorithms as well as a Naive Bayes classifier and collaborative filtering.
- The new spark-item similarity enables the next generation of co-occurrence recommenders that can use entire user click streams and context in making recommendations.

Mahout

Version 0.11.2

Latest release version 0.11.2 has Mahout Samsara Environment
Distributed Algebraic optimizer
R-Like DSL Scala API
Linear algebra operations
Ops are extensions to Scala
IScala REPL based interactive shell
Integrates with compatible libraries like MLLib
Run on distributed Spark and H2O
fastutil to speed up sparse matrix and vector computations
Flink in progress

Mahout Samsara based Algorithms
Stochastic Singular Value Decomposition (ssvd, dssvd)
Stochastic Principal Component Analysis (spca, dspca)
Distributed Cholesky QR (thinQR)
Distributed regularized Alternating Least Squares (dals)
Collaborative Filtering: Item and Row Similarity
Naive Bayes Classification
Distributed and in-core

http://mahout.apache.org/

Spark Libraries

https://spark.apache.org/

Lightning-fast cluster computing

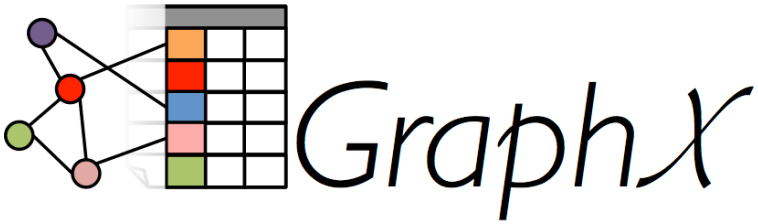
Download | Libraries | Documentation | Examples | Community | FAQ

- APIs: RDD, DataFrame and SQL
- Backend Execution: DataFrame and SQL
- Integrations: Data Sources, Hive, Hadoop, Mesos and Cluster Management
- R Language
- Machine Learning and Advanced Analytics
- Spark Streaming
- Deprecations, Removals, Configs, and Behavior Changes
  - Spark Core
  - Spark SQL & DataFrames
  - Spark Streaming
  - MLlib
- Known Issues
  - SQL/DataFrame
  - Streaming
- Credits

Spark Libraries

Spark MLlib

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



MLlib

Spark
Lightning-Fast Cluster Computing

MLlib types, algorithms and utilities

This lists functionality included in `spark.mllib`, the main MLlib API.

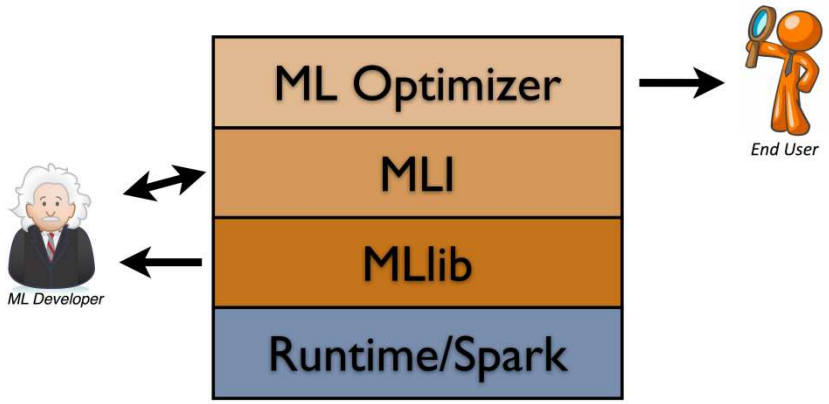
- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)
- PMML model export

https://spark.apache.org/mllib/

Mllib: Spark Machine learning library

- **MLlib (2010):** is a Spark implementation of some common machine learning functionality, as well associated tests and data generators.

- Includes:
  - Binary classification (SVMs and
  - Logistic Regression)
  - Random Forest
  - Regression (Lasso, Ridge, etc.)
  - Clustering (K-Means)
  - Collaborative Filtering
  - Gradient Descent Optimization
  - Primitive



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

MLlib

Spark
MLlib

http://spark.apache.org/mllib/

MLlib is Apache Spark's scalable machine learning library.

Ease of Use

Usable in Java, Scala and Python.

MLlib fits into Spark's APIs and interoperates with NumPy in Python (starting in Spark 0.9). You can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows.

Performance

High-quality algorithms, 100x faster than MapReduce.

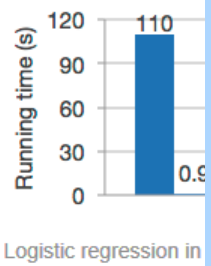
Spark excels at iterative computation, enabling MLlib to run fast. At the same time, we care about algorithmic performance: MLlib contains high-quality algorithms that leverage iteration, and can yield better results than the one-pass approximations sometimes used on MapReduce.

Easy to Deploy

Runs on existing Hadoop clusters and data.

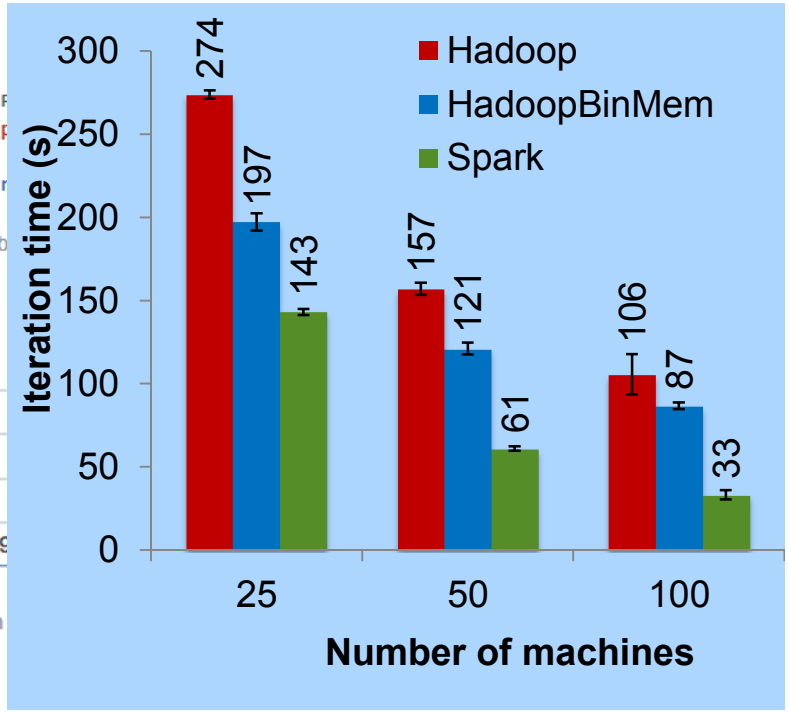
If you have a Hadoop 2 cluster, you can run Spark and MLlib without any pre-installation. Otherwise, Spark is easy to run standalone or on EC2 or Mesos. You can read from HDFS, HBase, or any Hadoop data source.

```
points = spark.textF
         .map(p

model = KMeans.train

Calling MLlib
```



Logistic regression in

K-Means



[Zaharia et. al, NSDI'12]

hadoop

MLlib

Spark
MLlib

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

Machine Learning Library (MLlib) Guide

MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives, as outlined below:

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
  - Gaussian mixture
  - power iteration clustering (PIC)
  - latent Dirichlet allocation (LDA)
  - streaming k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
  - FP-growth
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

MLlib is under active development. The APIs marked `Experimental/DeveloperApi` may change in future releases, and the migration guide below

http://spark-packages.org/

A community index of packages for Apache Spark.

MLlib

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

MLlib - Classification and Regression

MLlib supports various methods for binary classification, multiclass classification, and regression analysis. The table below outlines the supported algorithms for each type of problem.

Problem Type	Supported Methods
Binary Classification	linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes
Multiclass Classification	decision trees, random forests, naive Bayes
Regression	linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

More details for these methods can be found here:

- Linear models
  - binary classification (SVMs, logistic regression)
  - linear regression (least squares, Lasso, ridge)
- Decision trees
- Ensembles of decision trees
  - random forests
  - gradient-boosted trees
- Naive Bayes
- Isotonic regression

MLlib

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

MLlib - Clustering

Clustering is an unsupervised learning problem whereby we aim to group subsets of entities with one another based on some notion of similarity. Clustering is often used for exploratory analysis and/or as a component of a hierarchical supervised learning pipeline (in which distinct classifiers or regression models are trained for each cluster).

MLlib supports the following models:

- K-means
- Gaussian mixture
- Power iteration clustering (PIC)
- Latent Dirichlet allocation (LDA)
- Streaming k-means

K-means

k-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The MLlib implementation includes a parallelized variant of the k-means++ method called kmeans||. The implementation in MLlib has the following parameters:

- *k* is the number of desired clusters.
- *maxIterations* is the maximum number of iterations to run.
- *initializationMode* specifies either random initialization or initialization via k-means||.
- *runs* is the number of times to run the k-means algorithm (k-means is not guaranteed to find a globally optimal solution, and when run multiple times on a given dataset, the algorithm returns the best clustering result).
- *initializationSteps* determines the number of steps in the k-means|| algorithm.
- *epsilon* determines the distance threshold within which we consider k-means to have converged.

MLlib

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

MLlib - Feature Extraction and Transformation

- TF-IDF
- Word2Vec
  - Model
  - Example
- StandardScaler
  - Model Fitting
  - Example
- Normalizer
  - Example
- Feature selection
  - ChiSqSelector
    - Model Fitting
    - Example

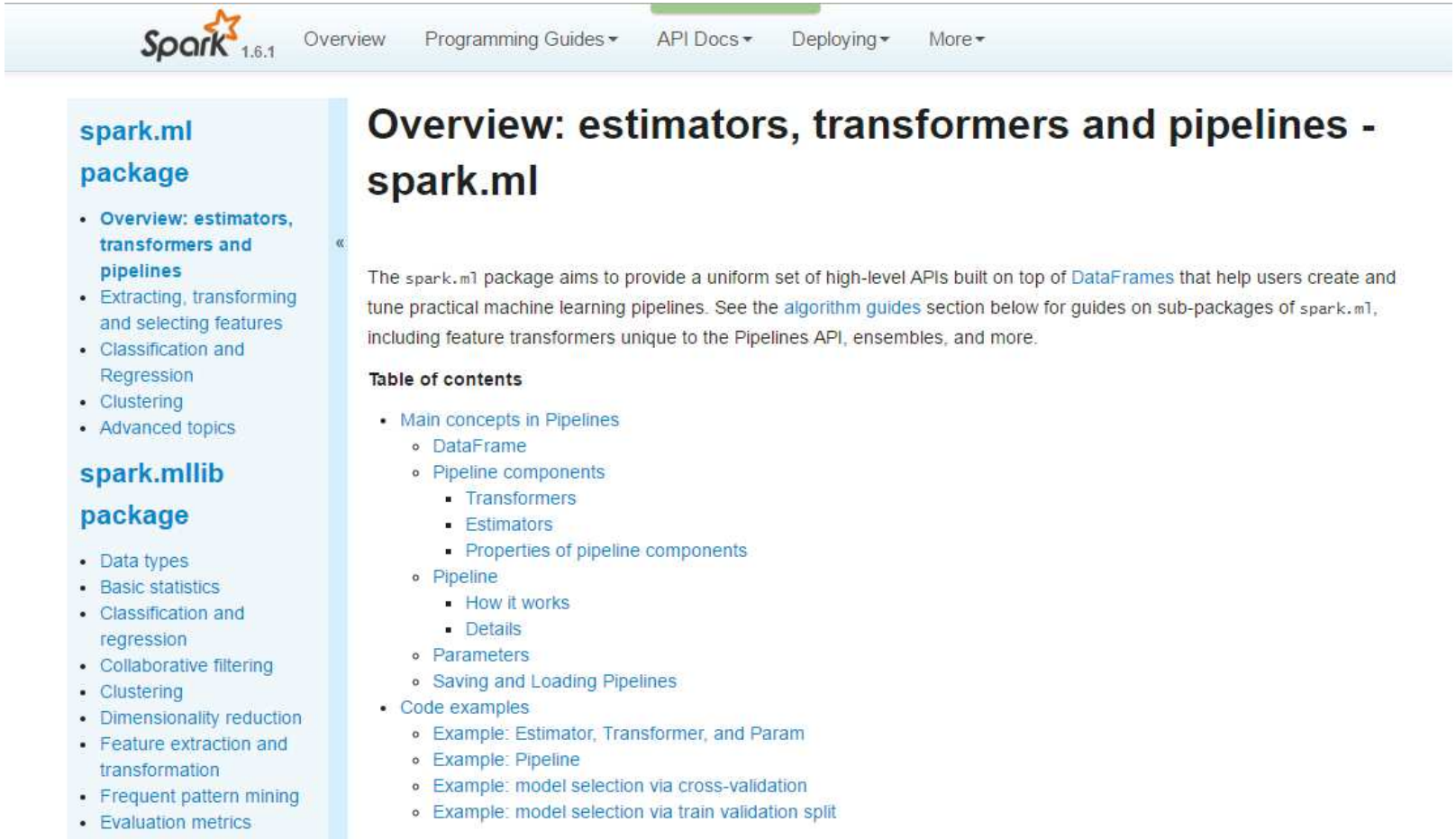
MLlib - Dimensionality Reduction

- Singular value decomposition (SVD)
  - Performance
  - SVD Example
- Principal component analysis (PCA)

Spark.ml

Spark
MLlib

http://spark.apache.org/docs/latest/ml-guide.html



FlinkML

https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/ml/

DataSet API
Transformations
Zipping Elements
Fault Tolerance
Iterations
Connectors
Python API
Examples
Libraries
Gelly
» Machine Learning
Table
Hadoop Compatibility

Important: Maven artifacts which depend on Scala are now suffixed with the Scala major version, e.g. "2.10" or "2.11". Please consult the migration guide on the project Wiki.

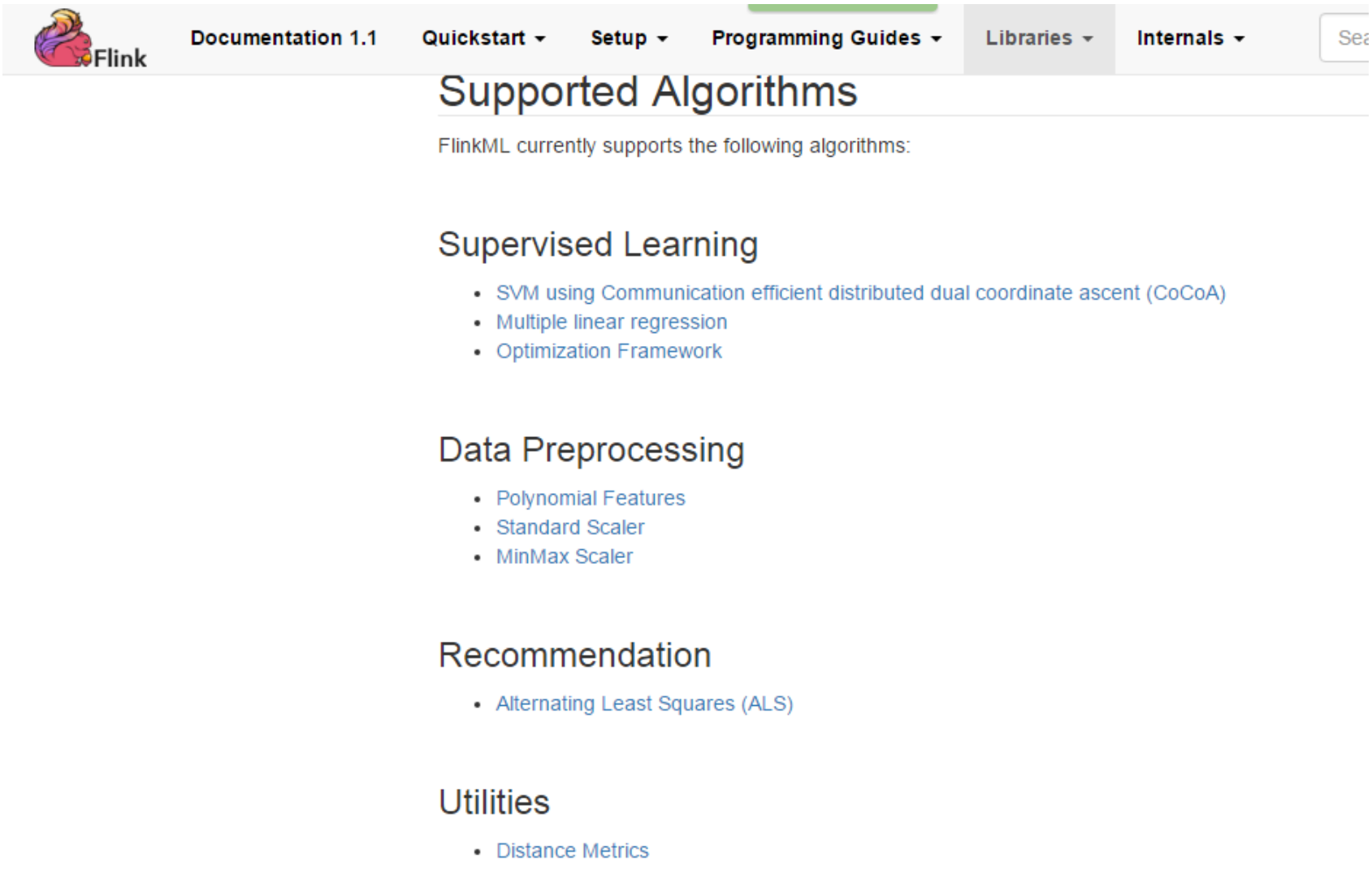
FlinkML - Machine Learning for Flink

FlinkML is the Machine Learning (ML) library for Flink. It is a new effort in the Flink community, with a growing list of algorithms and contributors. With FlinkML we aim to provide scalable ML algorithms, an intuitive API, and tools that help minimize glue code in end-to-end ML systems. You can see more details about our goals and where the library is headed in our vision and roadmap here.

Supported Algorithms
Supervised Learning
Data Preprocessing
Recommendation
Utilities
Getting Started
Pipelines
How to contribute

FlinkML

https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/ml/

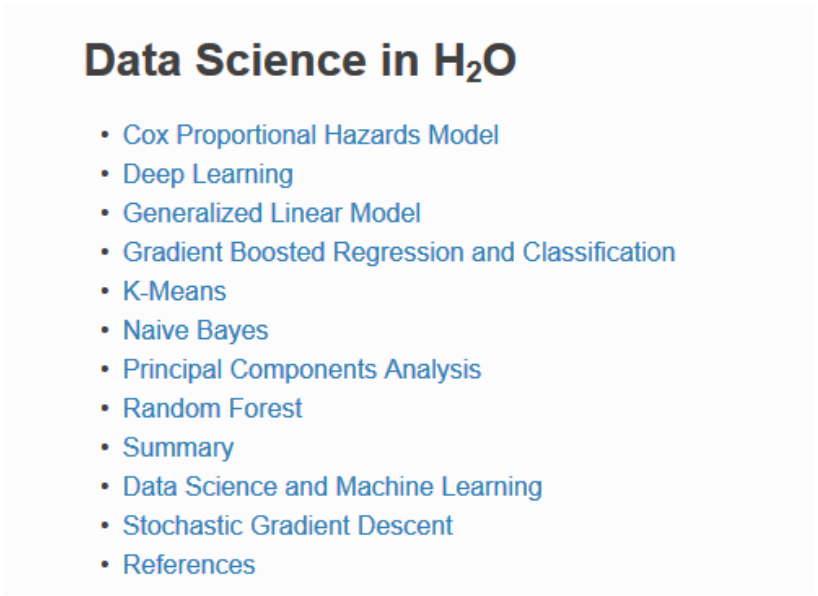


H_20 library

H_2O

H_2O

http://0xdata.com/



Support for R, Python, Hadoop y Spark

Way of working: It creates a new JVM that optimizes the paralellism of the algorithms

- It contains Deep Learning algorithms
  - World record to solve the MNIST problema without preprocessing

http://0xdata.com/blog/2015/02/deep-learning-performance/

H_2O Library

H_2O

H_2O APIs

Overview and walkthroughs for the different APIs to H_2O.

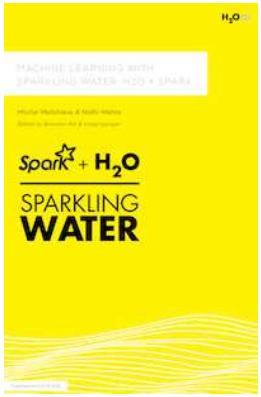
- R On H_2O
- Tableau on H_2O

http://www.h2o.ai/resources/

Spark + H_2O SPARKLING WATER

http://h2o-release.s3.amazonaws.com/h2o/rel-turan/4/docs-website/h2o-r/h2o_package.pdf

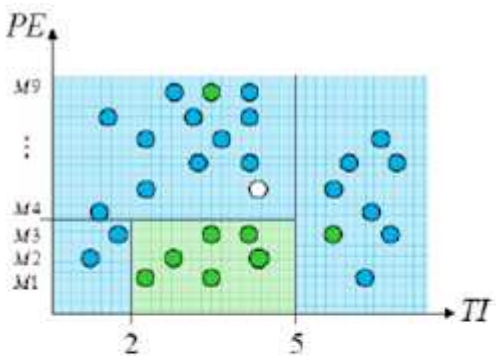
Machine Learning with Sparkling Water: H2O + Spark

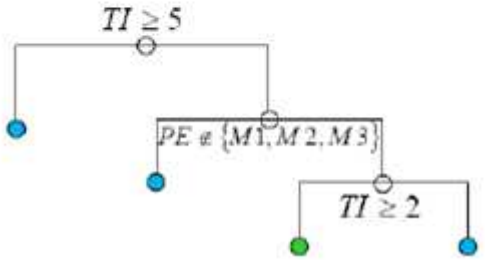


Sparkling Water allows users to combine the fast, scalable machine learning algorithms of H2O with the capabilities of Spark. With Sparkling Water, users can drive computation from Scala/R/Python and utilize the H2O Flow UI, providing an ideal machine learning platform for application developers.

Case of Study: Random Forests

- Random Forest is a very well-known machine learning technique for **classification** or regression.
  - Ensemble learning
  - Tree-based models
  - Random selection of features

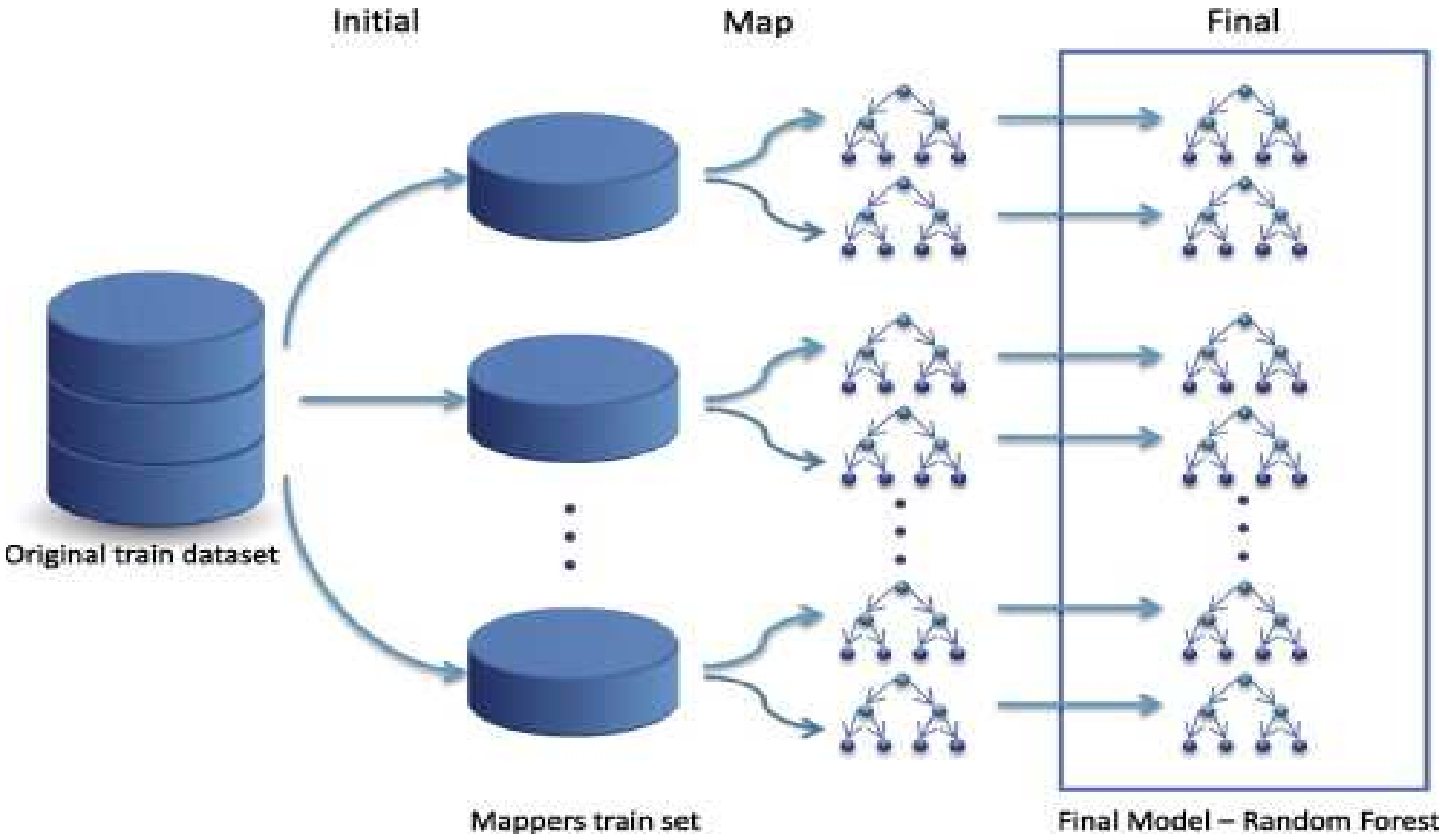




- Most promising characteristics:
  - Great generalization capabilities
  - Detect variable importance
  - Relatively efficient on large data bases.

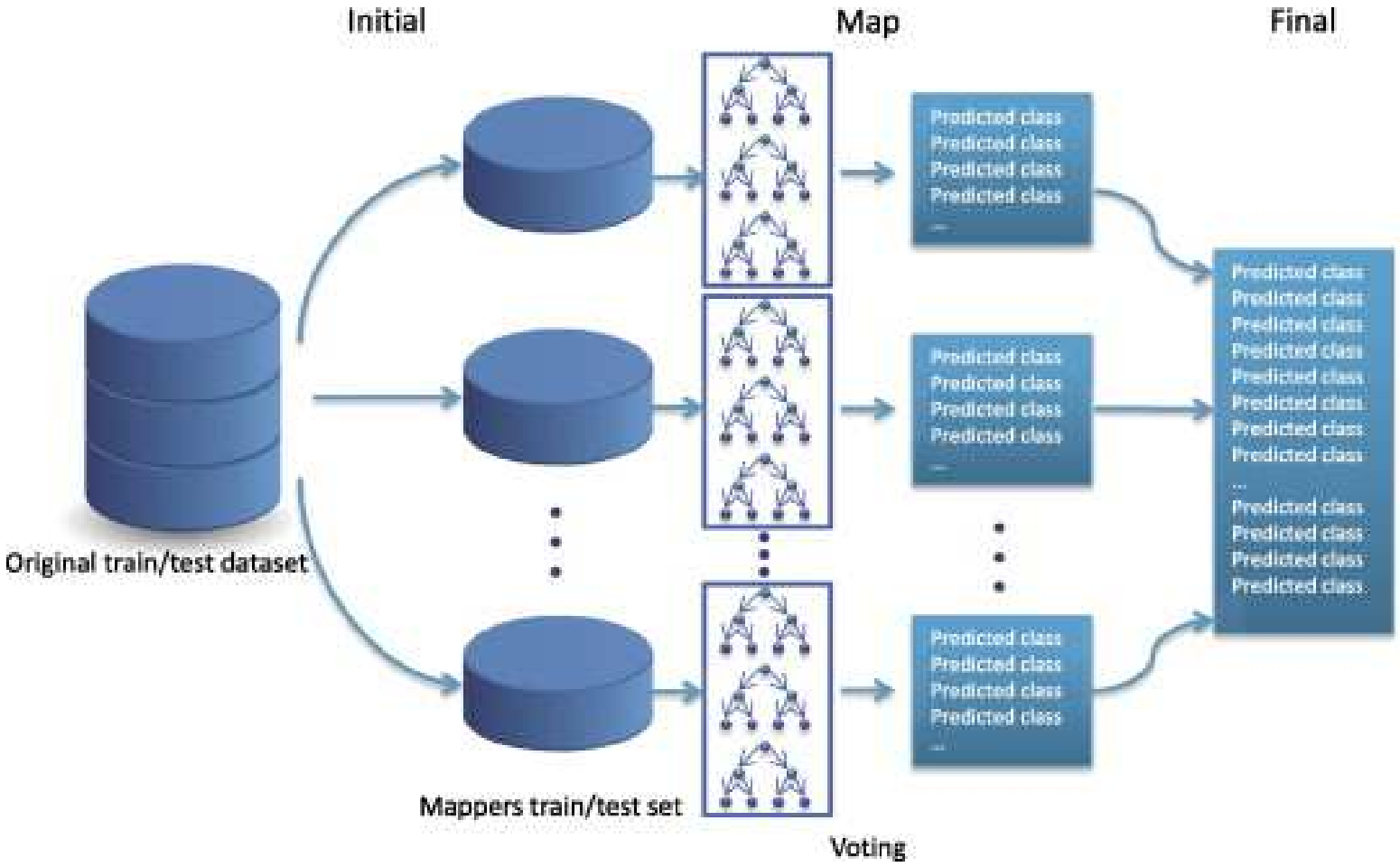
Random Forest under MapReduce

1) Building phase



Random Forest under MapReduce

2) Testing phase



Big Data Analytics

Case of Study: Random Forest for KddCup'99

Class	Instance Number
normal	972.781
DOS	3.883.370
PRB	41.102
R2L	1.126
U2R	52

Time elapsed (seconds) for sequential versions:

Datasets	RF		
	10%	50%	full
DOS_versus_normal	6344.42	49134.78	NC
DOS_versus_PRB	4825.48	28819.03	NC
DOS_versus_R2L	4454.58	28073.79	NC
DOS_versus_U2R	3848.97	24774.03	NC
normal_versus_PRB	468.75	6011.70	NC
normal_versus_R2L	364.66	4773.09	14703.55
normal_versus_U2R	295.64	4785.66	14635.36

Big Data Analytics

Case of Study: Random Forest for KddCup'99

Class	Instance Number
normal	972.781
DOS	3.883.370
PRB	41.102
R2L	1.126
U2R	52

	10%	50%	full
DOS_versus_normal	6344.42	49134.78	NC
DOS_versus_PRB	4825.48	28819.03	NC

Time elapsed (seconds) for Big data versions with 20 partitions:

Datasets	RF-BigData		
	10%	50%	full
DOS_versus_normal	98	221	236
DOS_versus_PRB	100	186	190
DOS_versus_R2L	97	157	136
DOS_versus_U2R	93	134	122
normal_versus_PRB	94	58	72
normal_versus_R2L	92	39	69
normal_versus_U2R	93	52	64

Cluster ATLAS: 16 nodes
-Microprocessors: 2 x Intel E5-2620 (6 cores/12 threads, 2 GHz)
- RAM 64 GB DDR3 ECC 1600MHz
- Mahout version 0.8

Case of Study: K-means

vs

- **Mahout: The K-Means algorithm**

- **Input**
  - Dataset (set of points in 2D) –Large
  - Initial centroids (K points) –Small

- **Map Side**
  - Each map reads the K-centroids + one block from dataset
  - Assign each point to the closest centroid
  - Output <centroid, point>

R. M. Esteves, C. Rong, R. Pais, **K-means Clustering in the Cloud – A Mahout Test**. IEEE Workshops of International Conference on Advanced Information Networking and Applications, pp.514,519, 22-25 March 2011.

Case of Study: K-means

vs

Mahout: K-means clustering

- **Reduce Side**
  - Gets all points for a given centroid
  - Re-compute a new centroid for this cluster
  - Output: <new centroid>
- **Iteration Control**
  - Compare the old and new set of K-centroids If similar or max iterations reached then Stop Else Start another Map-Reduce Iteration

- **THIS IS AN ITERATIVE MAP-REDUCE ALGORITHM**

Case of Study: K-means

vs

- **Map phase: assign cluster IDs**

(x1, y1) , centroid1, centroid2, …	→	<cluster_n, (x1, y1)>
(x2, y2) , centroid1, centroid2, …	→	<cluster_m, (x2, y2)>
(x3, y3) , centroid1, centroid2, …	→	<cluster_i, (x3, y3)>
(x4, y4) , centroid1, centroid2, …	→	<cluster_k, (x4, y4)>

- **Reduce phase: reset centroids**

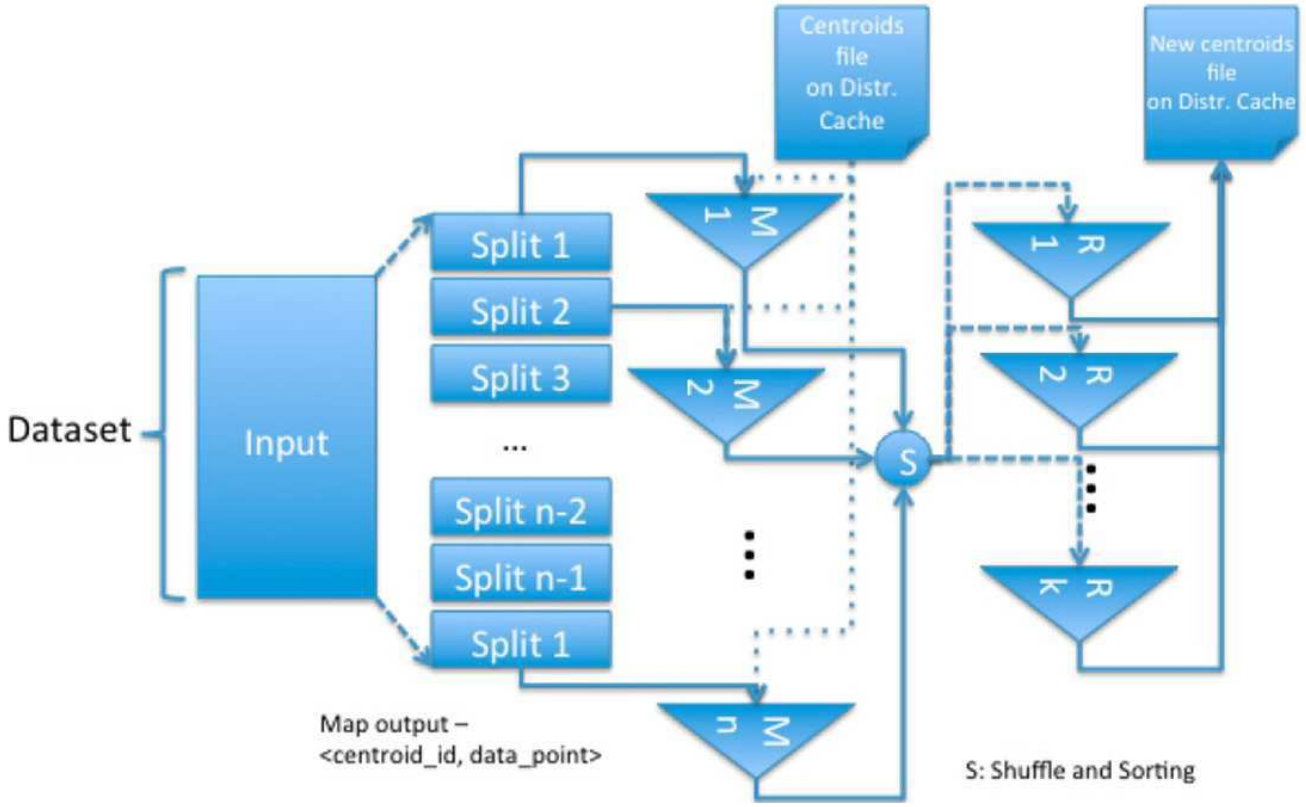
<cluster_n, (x1, y1)>	→	<cluster_n, centroid_n>
<cluster_m, (x2, y2)>	→	<cluster_m, centroid_m>
<cluster_i, (x3, y3)>	→	<cluster_i, centroid_i>
<cluster_k, (x4, y4)>	→	<cluster_k, centroid_k>

Case of Study: K-means

vs

Spark MLlib

Mahout: K-means clustering



R. M. Esteves, C. Rong, R. Pais, **K-means Clustering in the Cloud – A Mahout Test**. IEEE Workshops of International Conference on Advanced Information Networking and Applications, pp.514,519, 22-25 March 2011.

Case of Study: K-means

vs

K-Means: An example of limitation of MapReduce

- **What's wrong with these iterative approaches?**
  - Iterative algorithms in MapReduce chain multiple jobs together.
  - The standard MapReduce is not ready to do this.
  - Hadoop offers some snippets (Counters) to determine the stopping criteria.

- **Main issues:**
  - MapReduce jobs have high startup costs.
  - Repetitive Shuffle.
  - Results are serialized to HDFS.

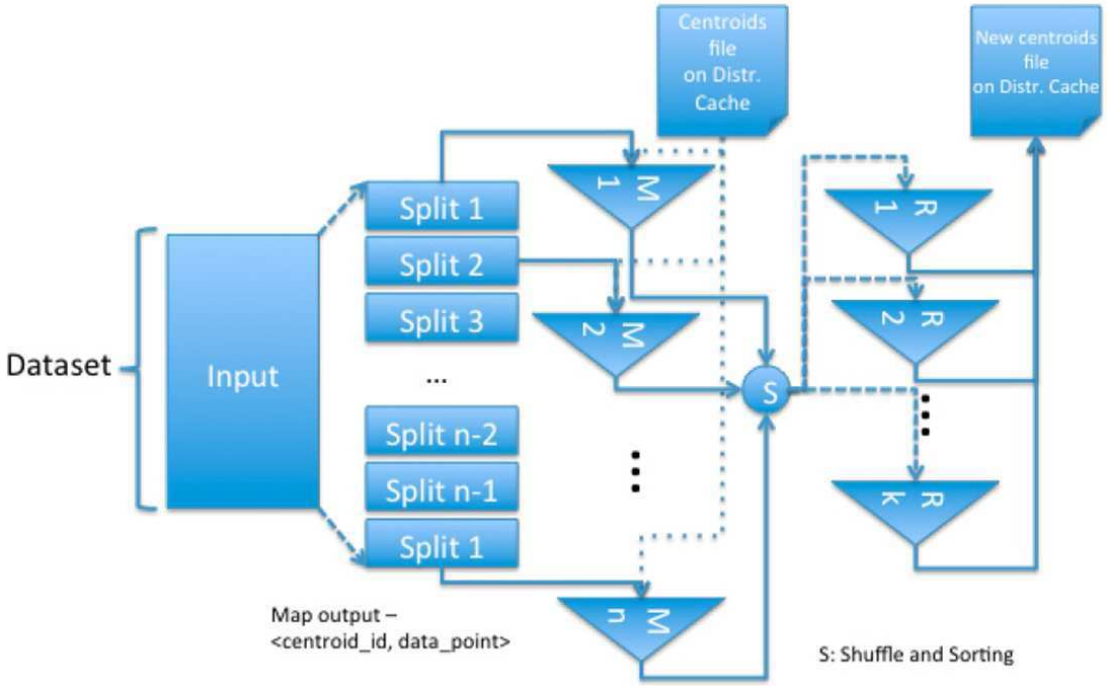
K-means

K-Means Clustering using Spark

Focus: Implementation and Performance

K-means

Mahout: K-means clustering



- Iterative algorithms in MapReduce chain multiple jobs together.
- The standard MapReduce is not ready to do this.

R. M. Esteves, C. Rong, R. Pais, **K-means Clustering in the Cloud – A Mahout Test**. IEEE Workshops of International Conference on Advanced Information Networking and Applications, pp.514,519, 22-25 March 2011.

K-means

MLlib: K-means clustering

k-means pseudo-code:

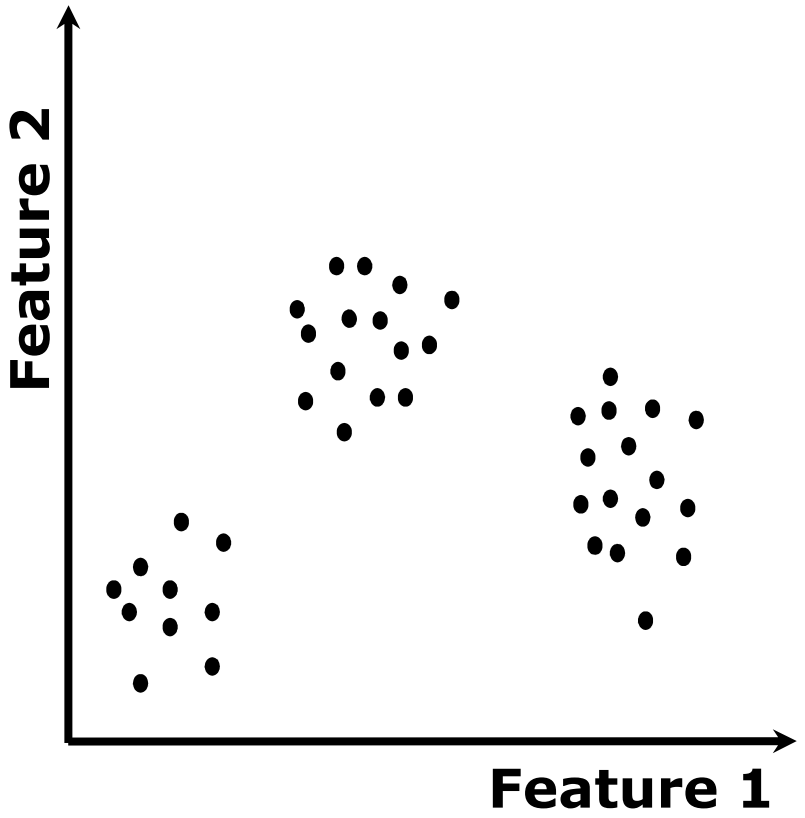
- Initialize K cluster centers
- Repeat until convergence:
  Assign each data point to the cluster with the closest center.
  Assign each cluster center to be the mean of its cluster's data points.

k-means MLlib source

```
centers = data.takeSample(
    false, K, seed)
while (d > ε)
{
 closest = data.map(p =>
     (closestPoint(p,centers),p))
 pointsGroup =
     closest.groupByKey()
 newCenters =pointsGroup.mapValues(
     ps => average(ps))
 d = distance(centers, newCenters)
 centers = newCenters.map(_)
}
```

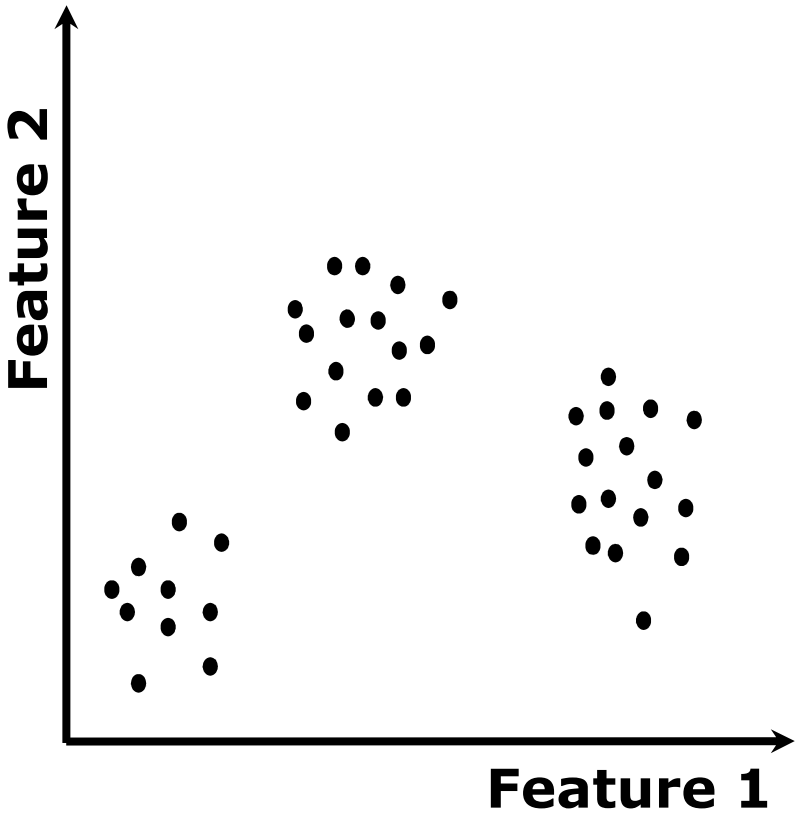
K-Means Algorithm

- Initialize K cluster centers
- Repeat until convergence:
  - Assign each data point to the cluster with the closest center.
  - Assign each cluster center to be the mean of its cluster's data points.



K-Means Algorithm

- **Initialize K cluster centers**
- **Repeat until convergence:**
  - **Assign each data point to the cluster with the closest center.**
  - **Assign each cluster center to be the mean of its cluster's data points.**

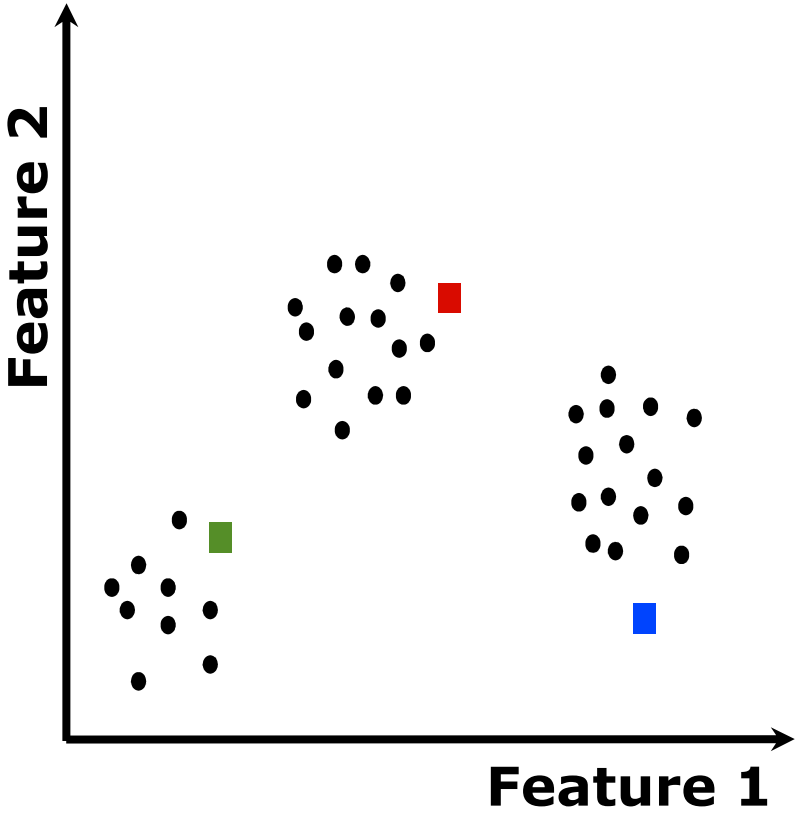


K-Means Algorithm

- **Initialize K cluster centers**

```
centers = data.takeSample(
    false, K, seed)
```

- **Repeat until convergence:**
  - **Assign each data point to the cluster with the closest center.**
  - **Assign each cluster center to be the mean of its cluster's data points.**

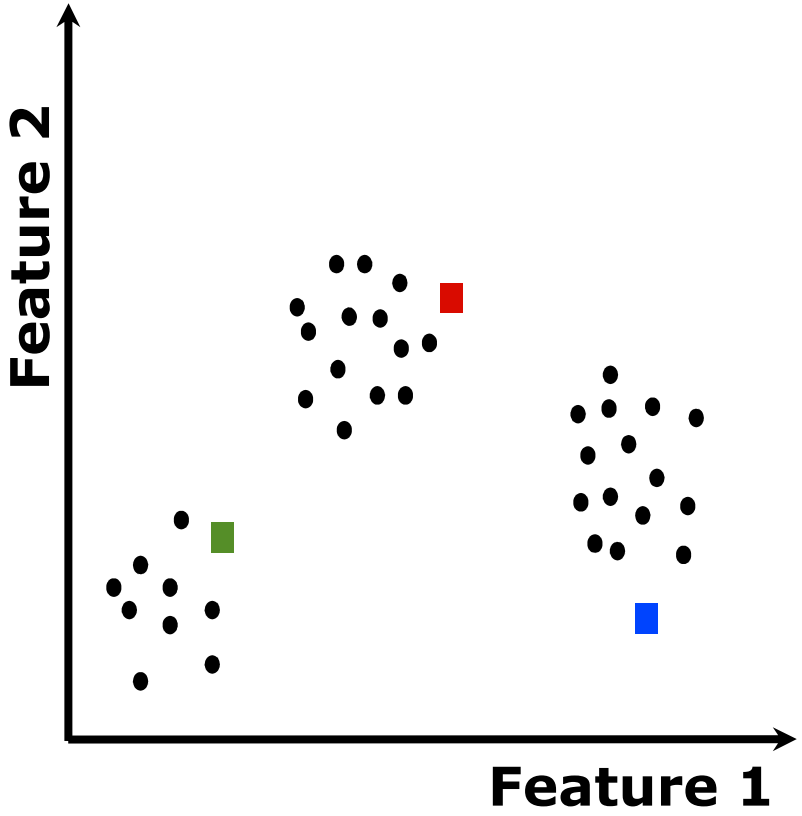


K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- **Repeat until convergence:**
  - **Assign** each data point to the cluster with the closest center.
  - Assign each cluster center to be the mean of its cluster's data points.

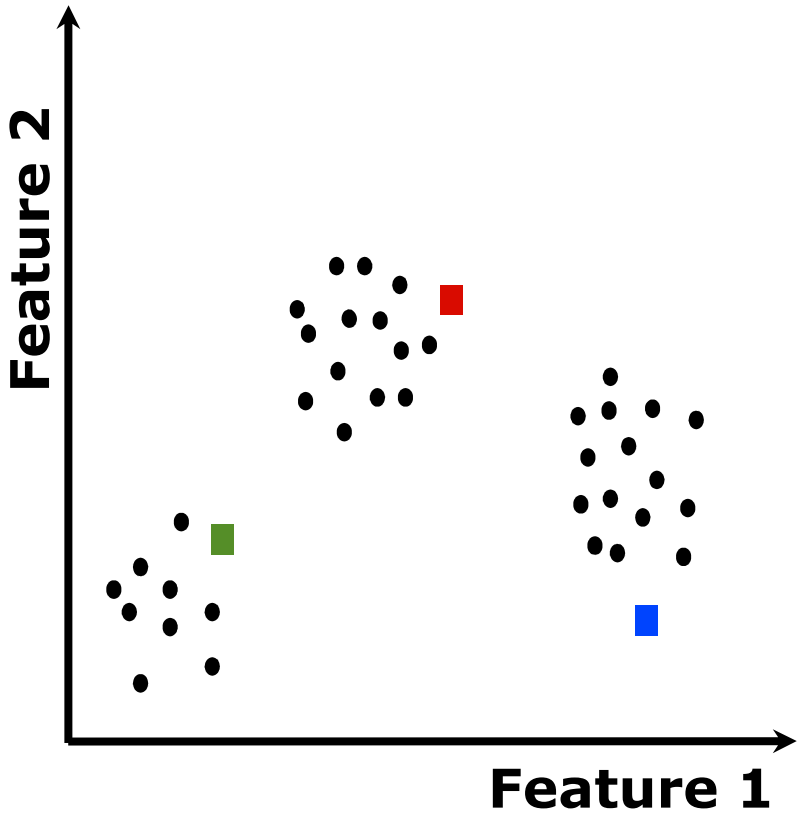


K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:
  - **Assign each data point to the cluster with the closest center.**
  - Assign each cluster center to be the mean of its cluster's data points.



K-Means Algorithm

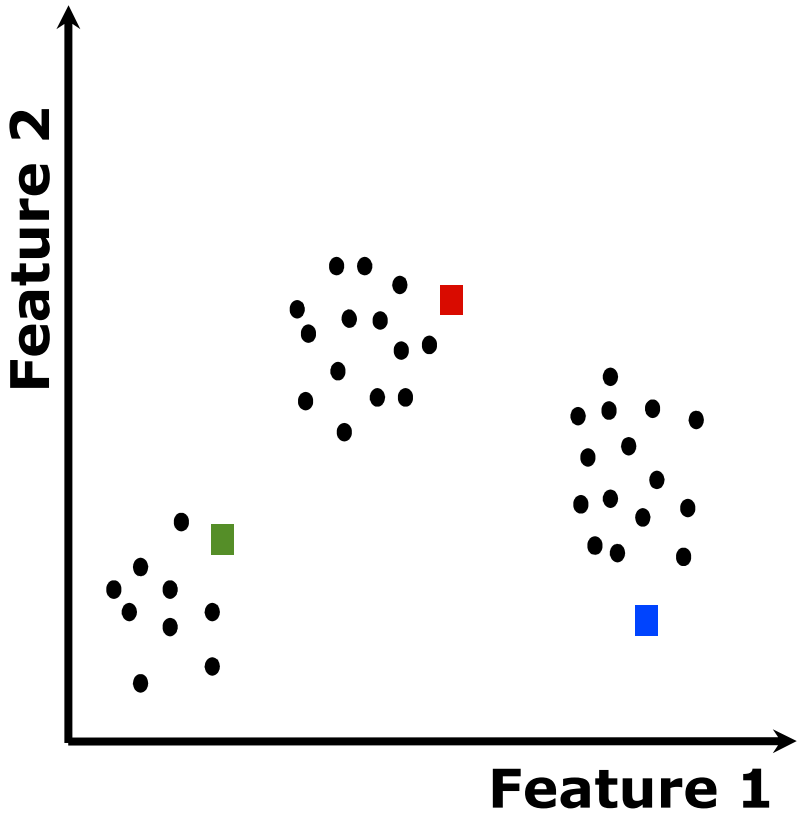
- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

(closestPoint(p,centers),p))
```

Assign each cluster center to be the mean of its cluster's data points.



K-Means Algorithm

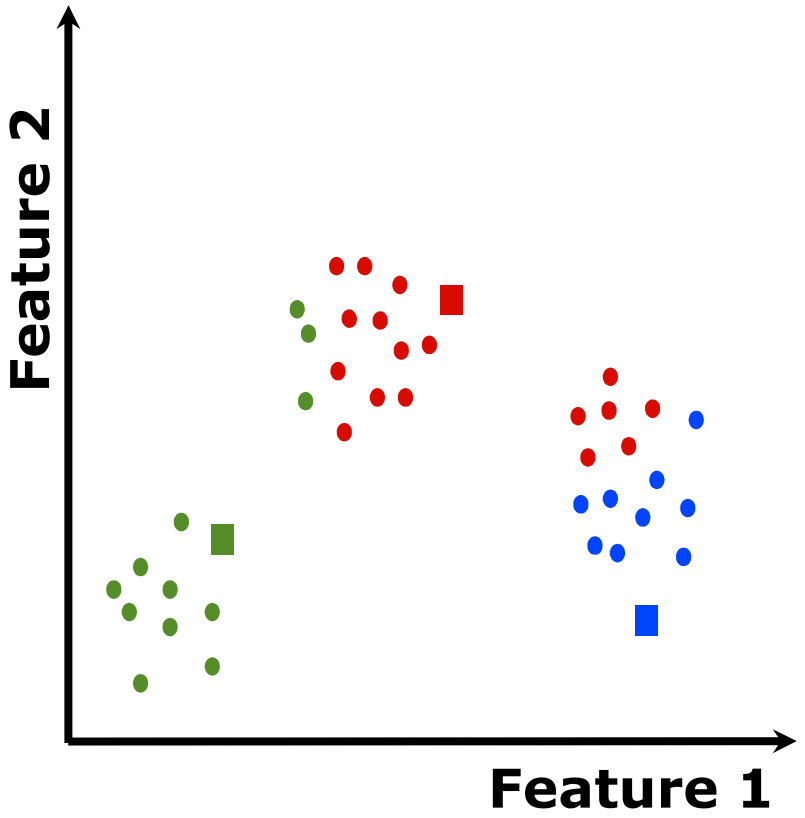
- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

(closestPoint(p,centers),p))
```

Assign each cluster center to be the mean of its cluster's data points.



K-Means Algorithm

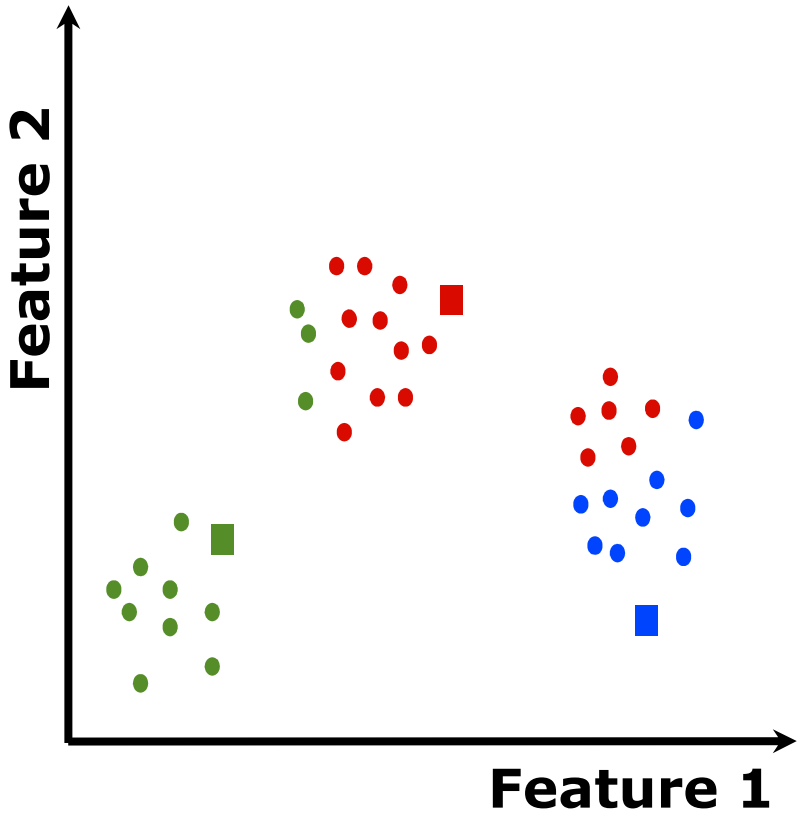
- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

(closestPoint(p,centers),p))
```

Assign each cluster center to be the mean of its cluster's data points.



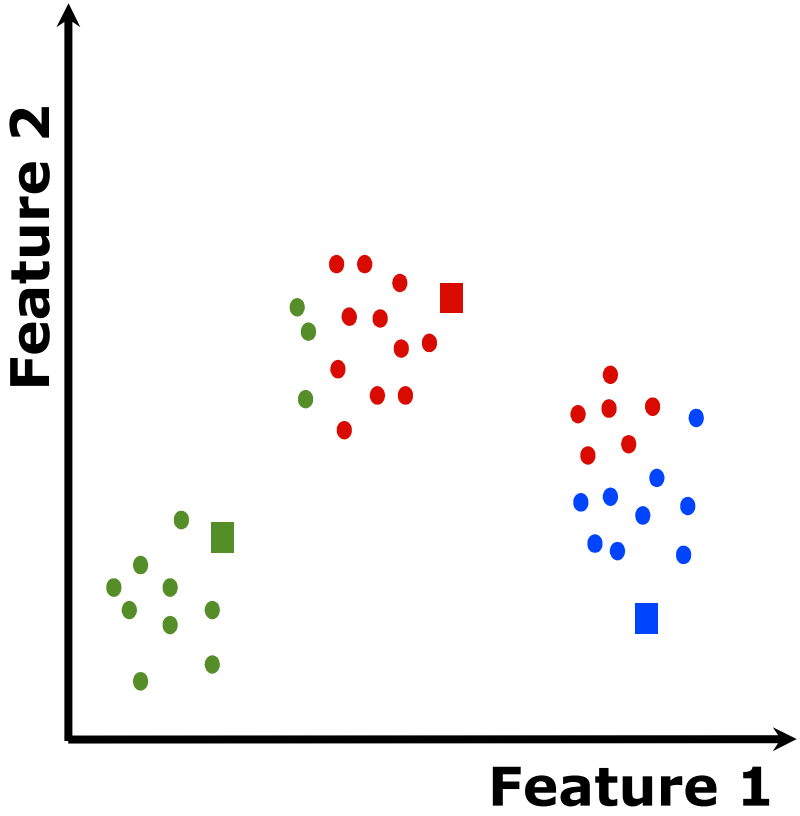
K-Means Algorithm

- Initialize K cluster centers

```
centers =data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

(closestPoint(p,centers),p))

pointsGroup =
    closest.groupByKey()
```



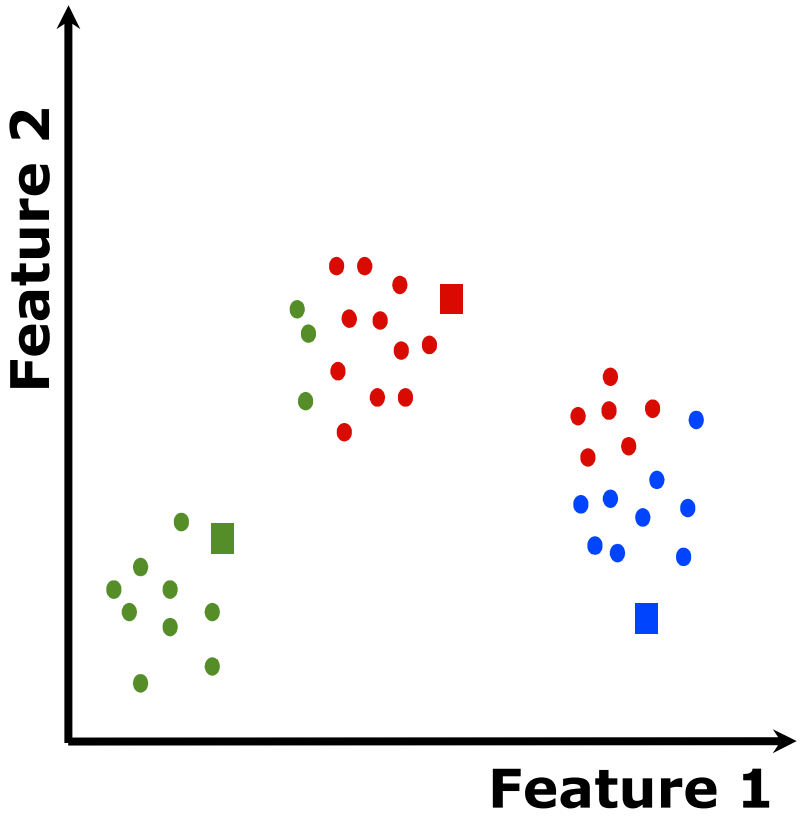
K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

 (closestPoint(p,centers),p))
pointsGroup =
    closest.groupByKey()

newCenters =
pointsGroup.mapValues(
    ps => average(ps))
```



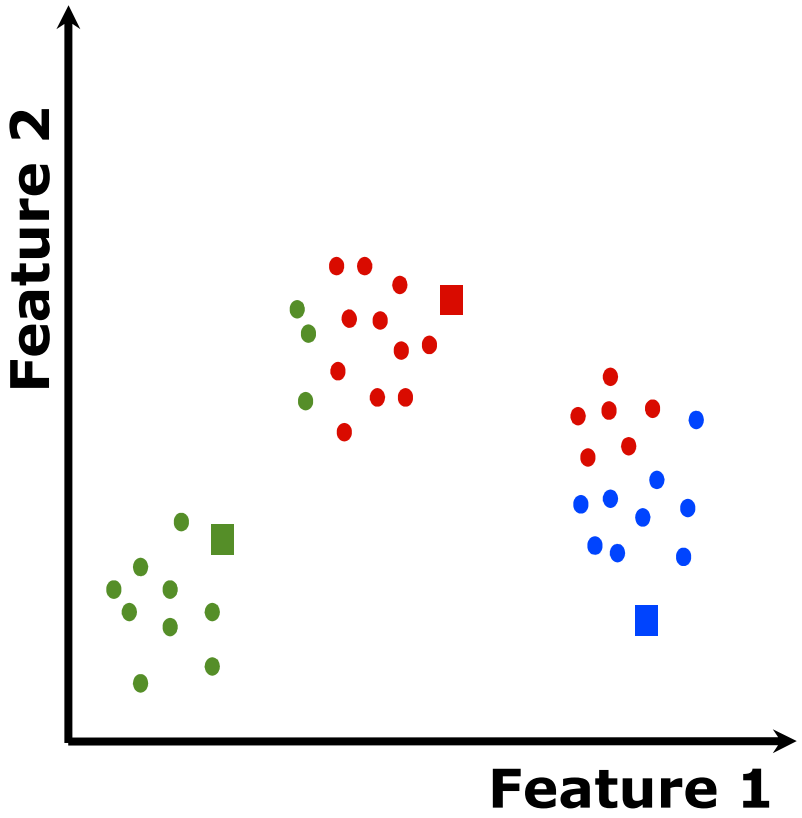
K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- Repeat until convergence:

```
closest = data.map(p =>

(closestPoint(p,centers),p))

pointsGroup =
    closest.groupByKey()

newCenters =
pointsGroup.mapValues(
    ps => average(ps))
```



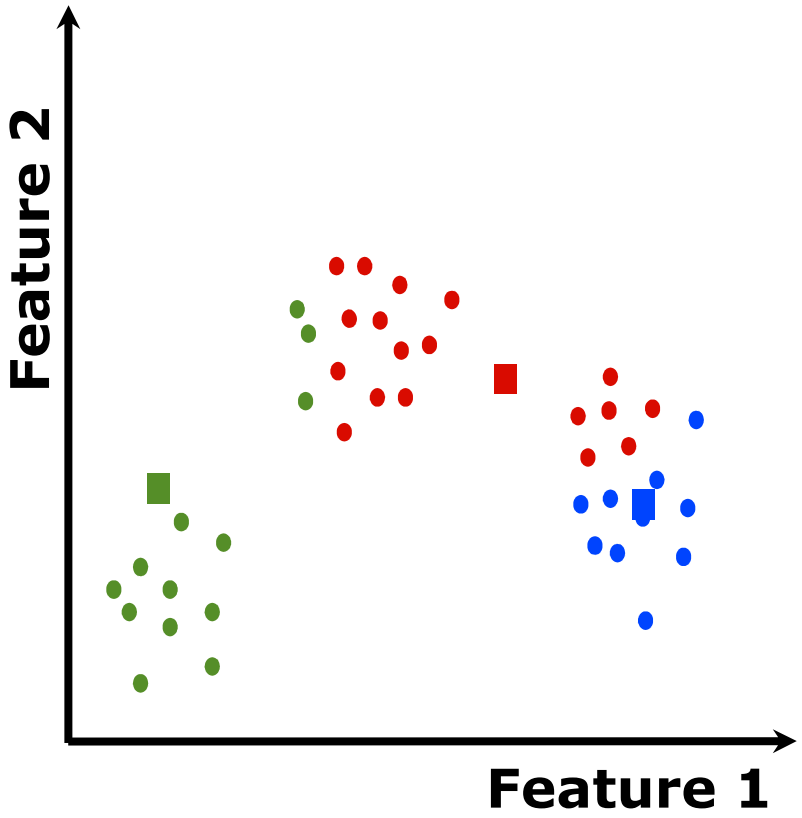
K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- **Repeat until convergence:**

```
closest = data.map(p =>

(closestPoint(p,centers),p))

pointsGroup =
    closest.groupByKey()

newCenters =
pointsGroup.mapValues(
    ps => average(ps))
```



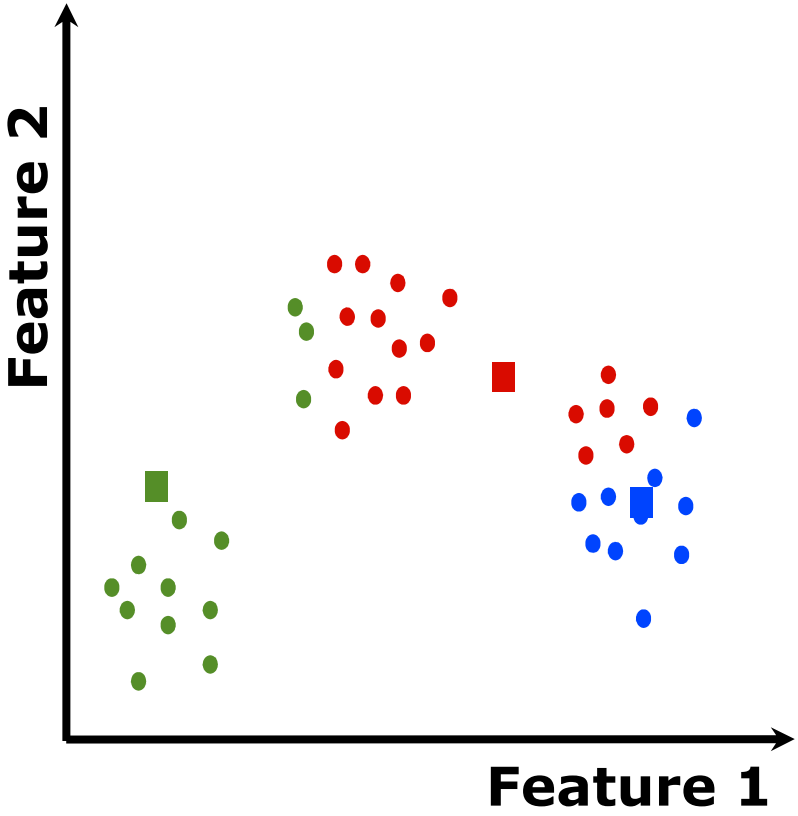
K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

- **Repeat until convergence:**

```
while (dist(centers, newCenters)
> ε)

closest = data.map(p =>

(closestPoint(p,centers),p))

pointsGroup =
    closest.groupByKey()

newCenters
=pointsGroup.mapValues(
    ps => average(ps))
```



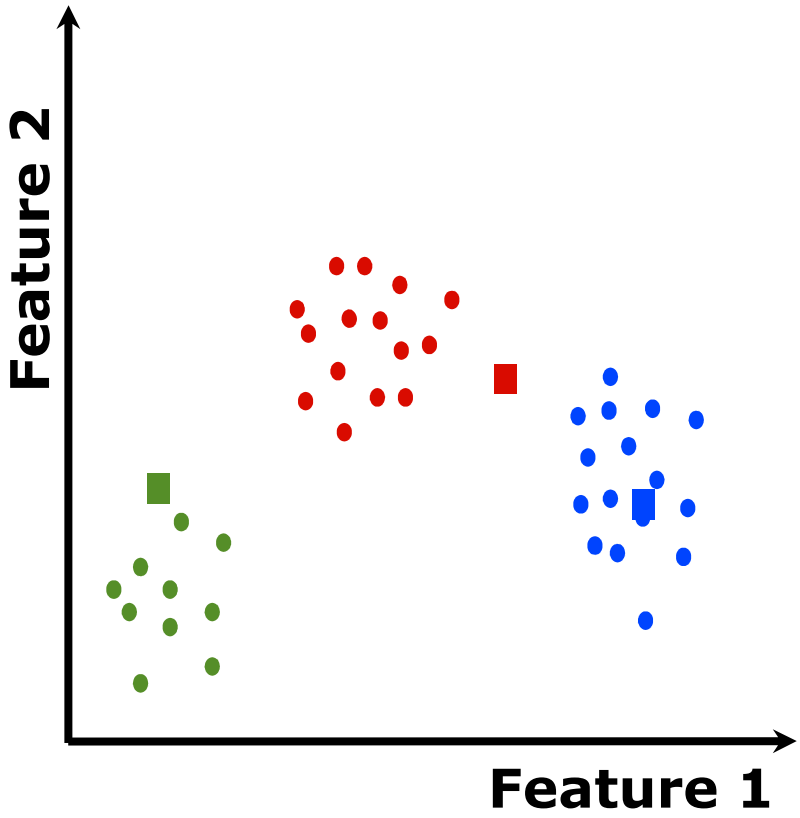
K-Means Algorithm

- Initialize K cluster centers

```
centers = data.takeSample(
    false, K, seed)
```

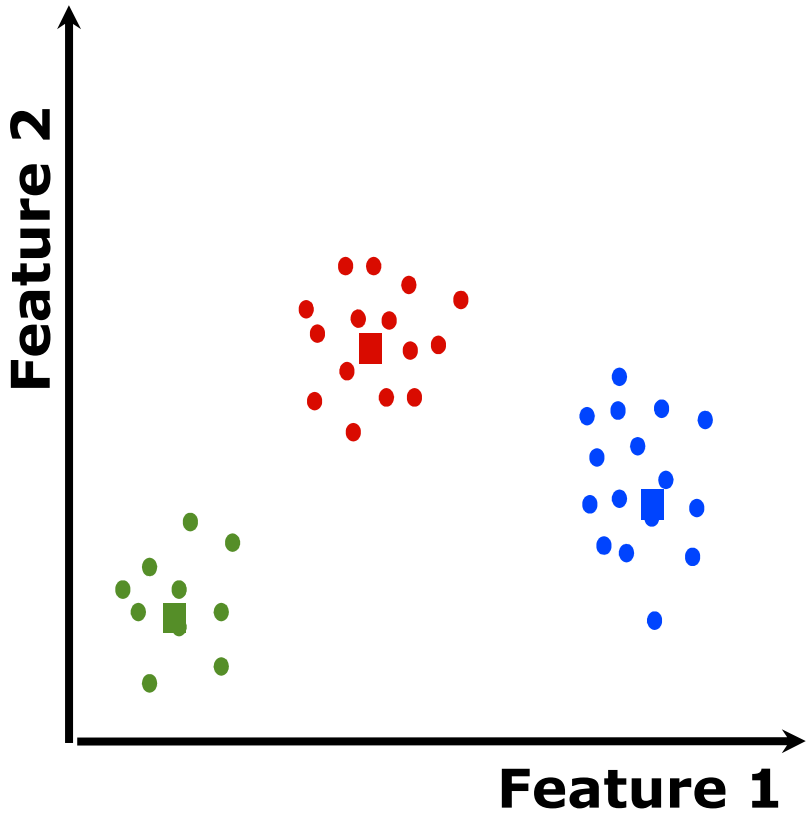
- **Repeat until convergence:**

```
while (dist(centers, newCenters)
> ε)

closest = data.map(p =>

(closestPoint(p,centers),p))

pointsGroup =
    closest.groupByKey()

newCenters
=pointsGroup.mapValues(
    ps => average(ps))
```



K-Means Source

```
centers = data.takeSample(
    false, K, seed)

while (d > ε)
{
  closest = data.map(p =>

   (closestPoint(p,centers),p))

  pointsGroup =
      closest.groupByKey()

  d = distance(centers, newCenters)

  newCenters
  =pointsGroup.mapValues(
      ps => average(ps))

  centers = newCenters.map(_)

 }
```

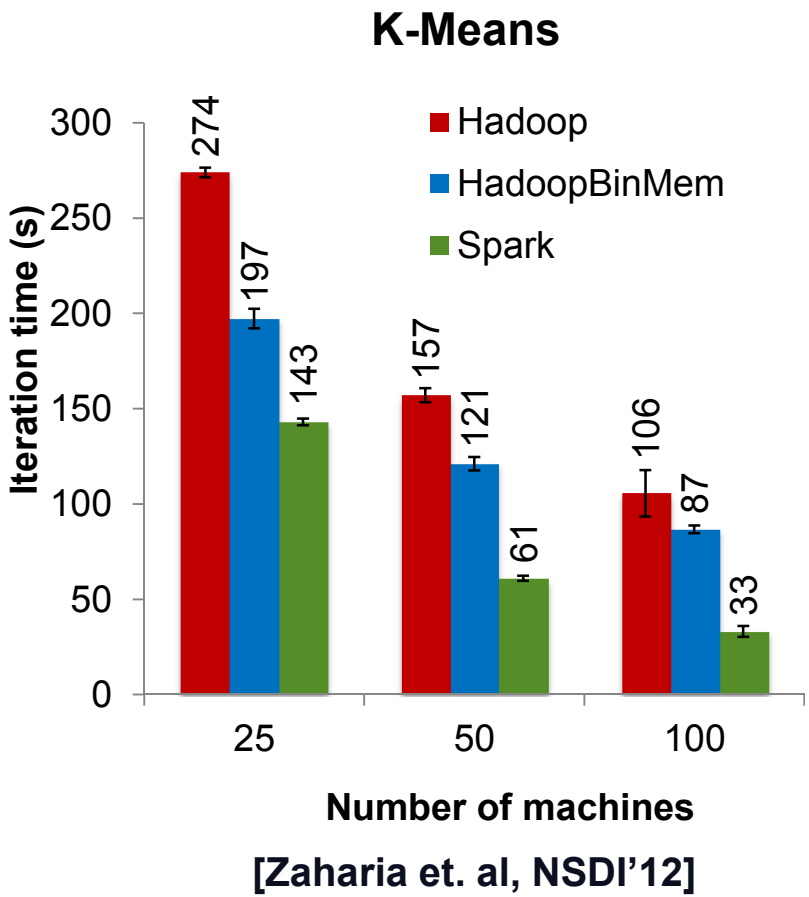


K-Means Performance

Lines of code for K-Means

Spark ~ 90 lines –

Hadoop ~ 4 files, > 300 lines



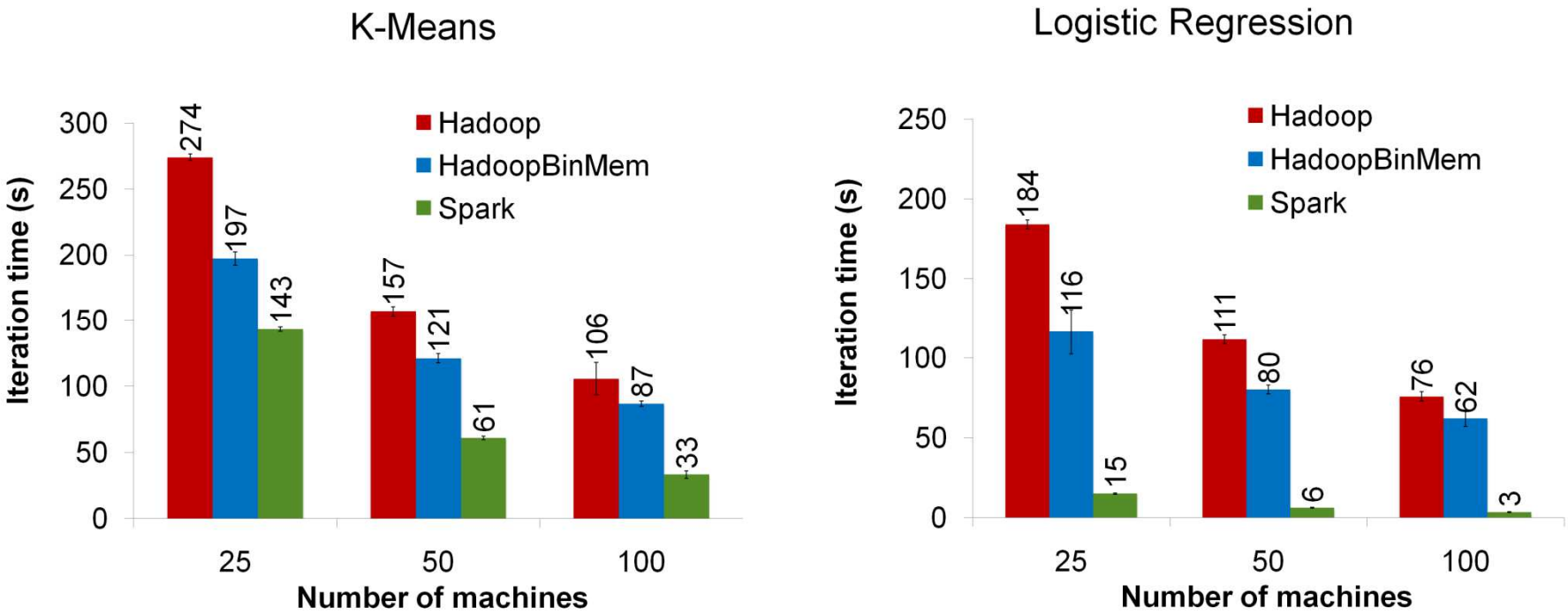
[Zaharia et. al, NSDI'12]

M. Zaharia et al. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**. NSDI 2012.

K-Means and Logistic Regression Performance

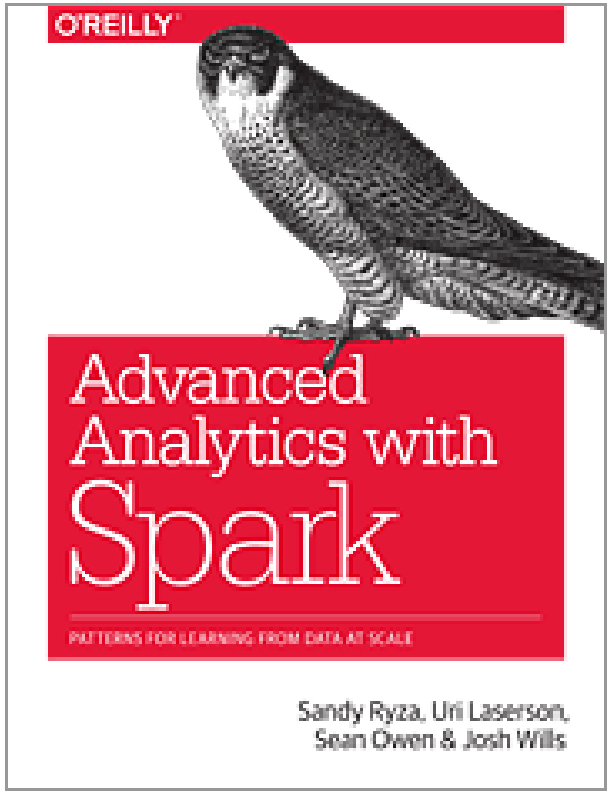
Spark MLlib

Performance

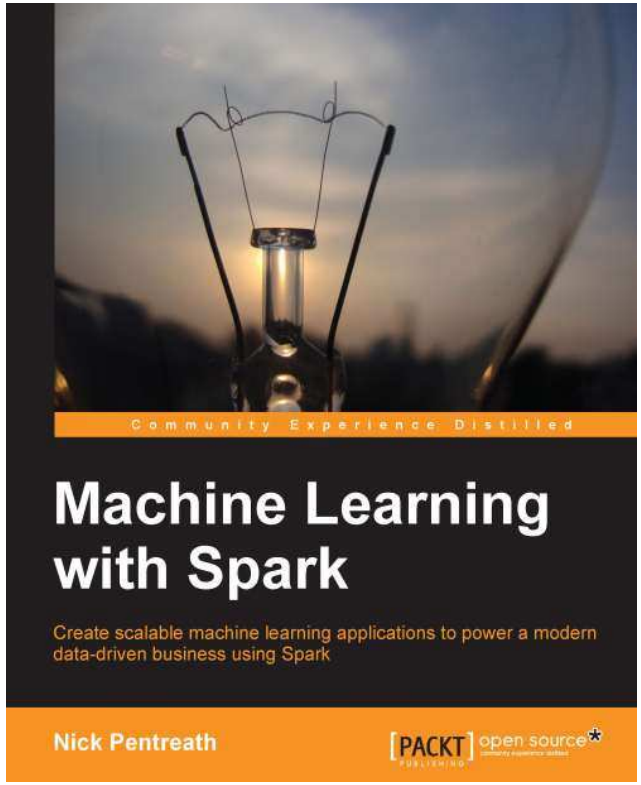


M. Zaharia et al. **Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing**. NSDI 2012.

Big Data Analytics: 2 books



9 cases of study



10 chapters giving a quick glance on Machine Learning with Spark

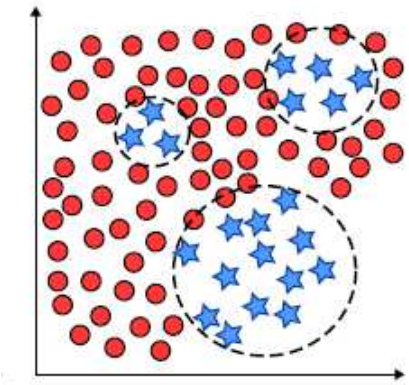
Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Uncertainty and Big Data

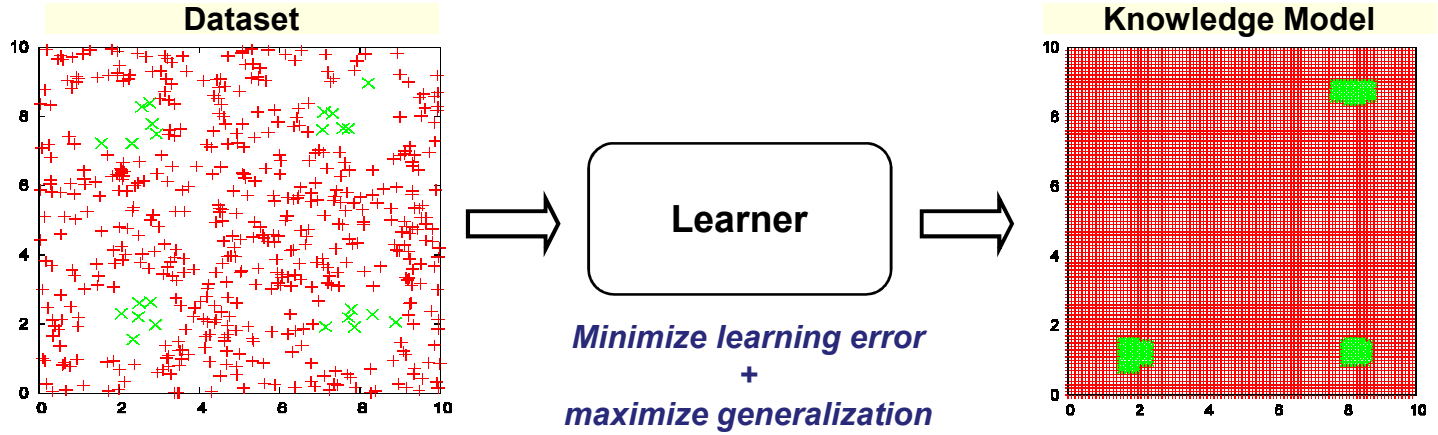
- Uncertainty is inherent to Big Data due to
  - Heterogeneous sources
  - Variety in data
  - Incomplete data
  - Veracity in question
- Fuzzy Rule Based Classification Systems can manage
  - Uncertainty
  - Vagueness
  - Lack of data/Data fragmentation

Appearance of small disjuncts with the MapReduce data fragmentation



(b) Small disjuncts

Rare cases or Small disjuncts are those disjuncts in the learned classifier that cover few training examples.

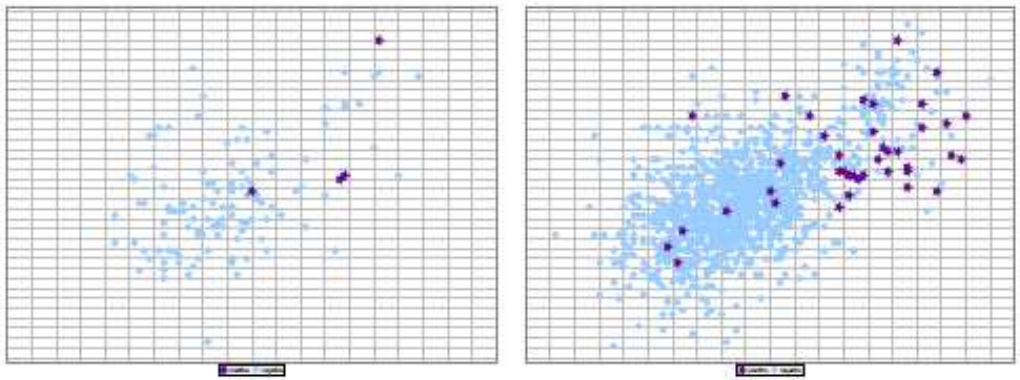


T. Jo, N. Japkowicz. Class imbalances versus small disjuncts. SIGKDD Explorations 6:1 (2004) 40-49

G.M. Weiss. Mining with Rarity: A Unifying Framework. SIGKDD Explorations 6:1 (2004) 7-19

Appearance of small disjuncts with the MapReduce data fragmentation

Data fragmentation - Lack of data



(a) 10 % of training instances (b) 100 % of training instances

Figure 11: Lack of density or small sample size on the yeast4 dataset

The lack of data in the training data may also cause the introduction of small disjuncts.

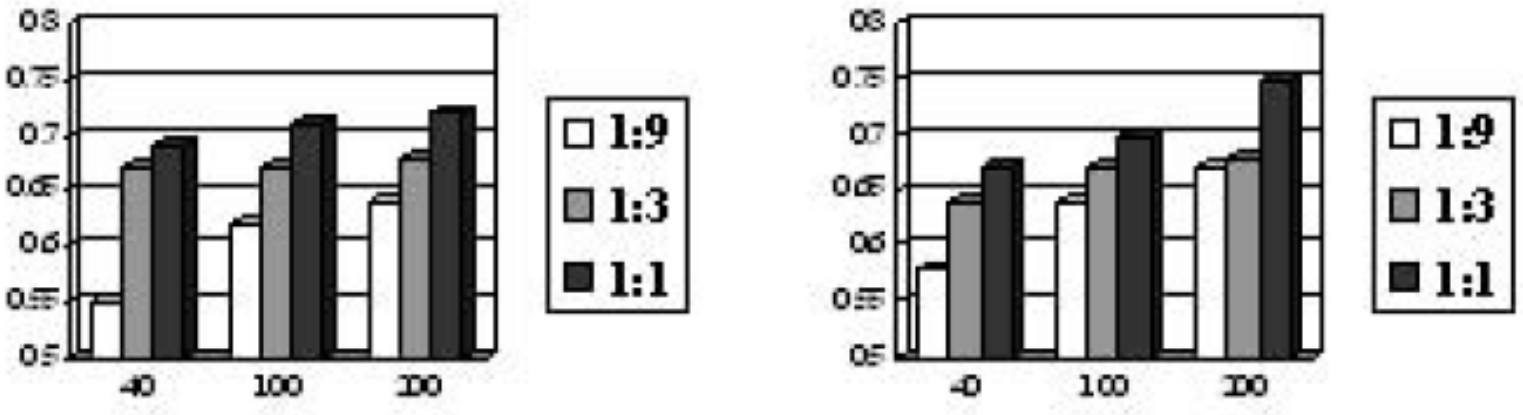
It becomes very hard for the learning algorithm to obtain a model that is able to perform a good generalization when there is not enough data that represents the boundaries of the problem.

And, what it is also most significant, when the concentration of minority examples is so low that they can be simply treated as noise.

Appearance of small disjuncts with the MapReduce data fragmentation

Lack of data

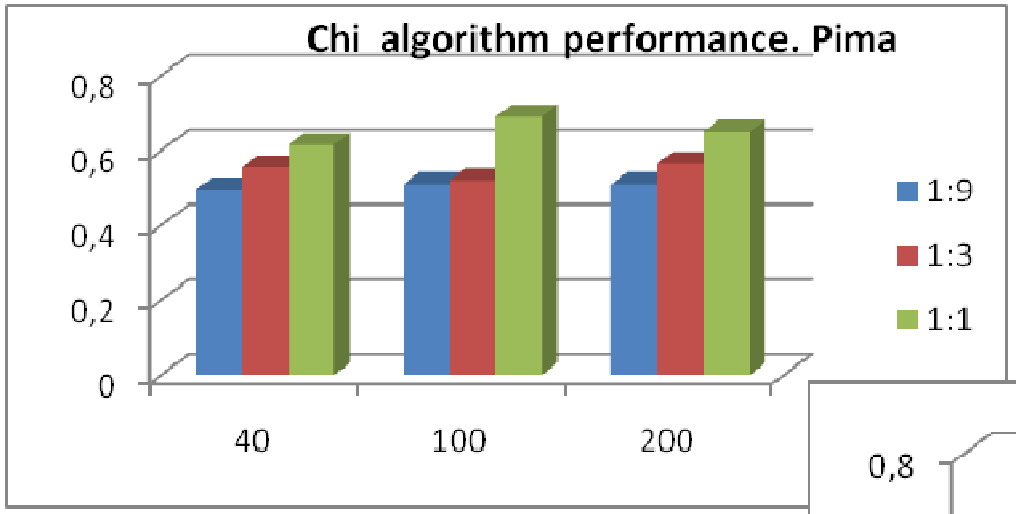
Left-C4.5, right-Backpropagation (Pima and Wisconsin Breast Cancer): These results show that the performance of classifiers, though hindered by class imbalances, is repaired as the training set size increases. This suggests that small disjuncts play a role in the performance loss of class imbalanced domains.

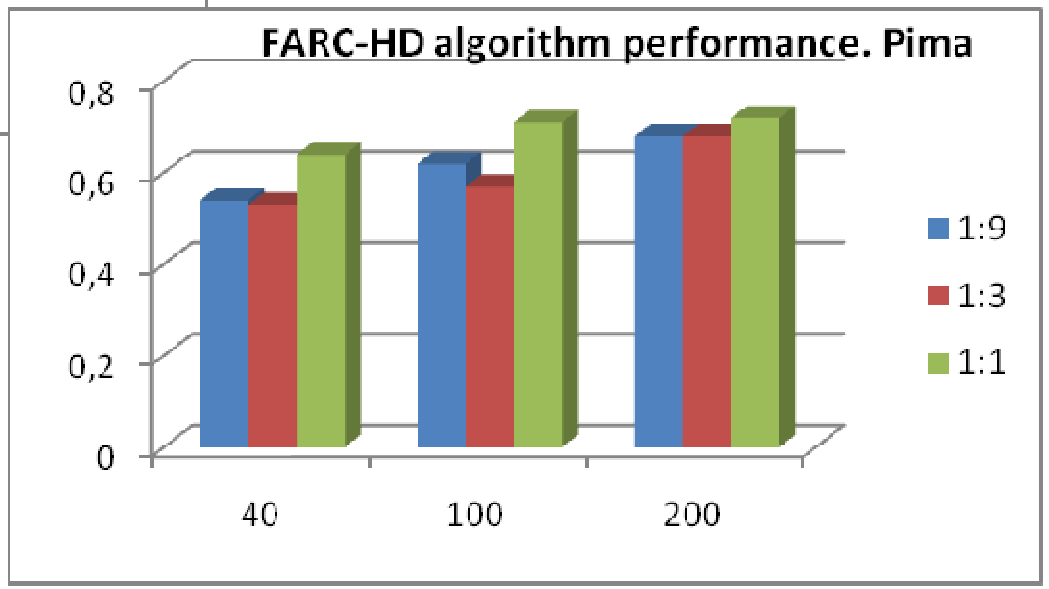


T. Jo, N. Japkowicz. Class imbalances versus small disjuncts. SIGKDD Explorations 6:1 (2004) 40-49

Appearance of small disjuncts with the MapReduce data fragmentation

Lack of data. Fuzzy models performance





Robustness to the lack of data?

Big Data: Selected Computational Intelligence approaches

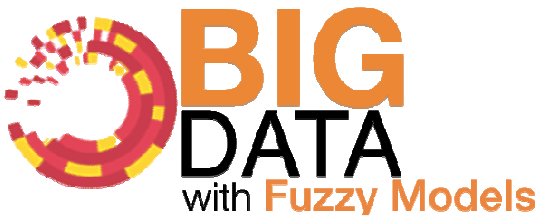
Chi-FRBCS-BigData: A Case of Study

We choose a simple Learning Methods to analyze the potential of FRBCSs for Big Data Classification

- MapReduce design based on the FRBCS algorithm (Chi et al).
- Uses two different MapReduce processes
  - Phase 1: Building the Fuzzy Rule Base
  - Phase 2: Estimating the class of samples belonging to big data sample sets
- Two versions which differ in the *Reduce function* of the building of the FRB have been produced
  - Chi-FRBCS-BigData-Max
  - Chi-FRBCS-BigData-Average

S. Río, V. López, J.M. Benítez, F. Herrera. *A MapReduce Approach to Address Big Data Classification Problems Based on the Fusion of Linguistic Fuzzy Rules.* International Journal of Computational Intelligence Systems 8:3 (2015) 422-437. doi: 10.1080/18756891.2015.1017377

Big Data: Selected Computational Intelligence approaches



Chi-FRBCS

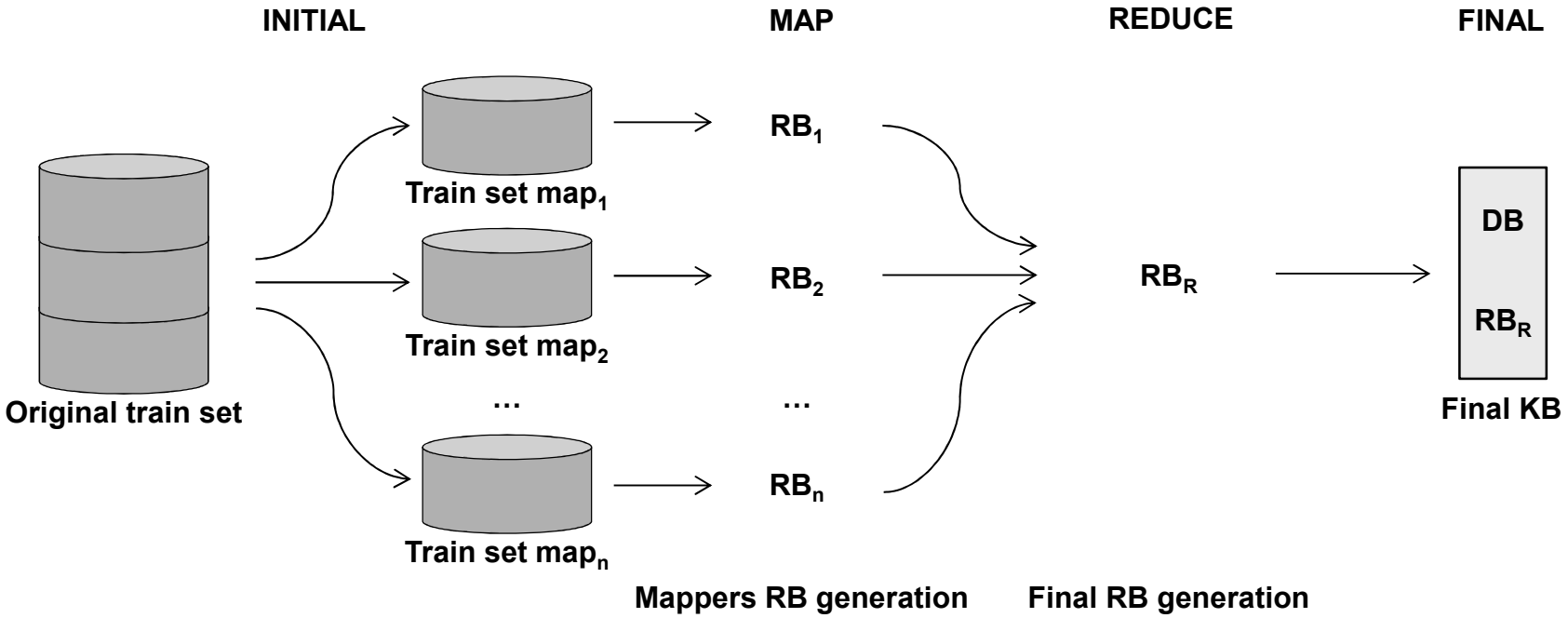
- Produces rules like "**Rule R_j:** IF x_1 IS A^1_j AND … AND x_n IS A^n_j THEN Class = C_j with RW_j"
- Builds the fuzzy partition using equally distributed triangular membership functions
- Builds the RB creating a fuzzy rule associated to each example
- Rules with the same antecedent may be created:
  - Same consequent → Delete duplicated rules
  - Different consequent → Preserve highest weight rule

Z. Chi, H. Yan and T. Pham, Fuzzy algorithms with applications to image processing and pattern recognition, World Scientific, 1996.

Big Data: Selected Computational Intelligence approaches

BIG DATA with Fuzzy Models

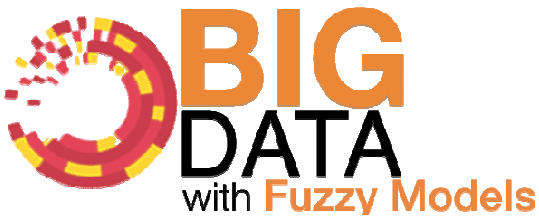
Building the RB with Chi-FRBCS-BigData: A Map Reduce approach



The key of a MapReduce data partitioning approach is usually on the reduce phase

Two alternative reducers (Max vs average weights)

Big Data: Selected Computational Intelligence approaches



Building the FRB with Chi-FRBCS-BigData-Max

RB_1:
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8743$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
…

RB_2:
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.9254$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
…

RB_3:
R_1: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.7142$
…

RB_4:
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_1 = 0.2143$
R_2: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
…

…

RB_n:
R_1: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_2 = 0.8215$
…

REDUCE

RB_R:
R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.9254$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_4: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_5: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
R_6: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
…

Final RB generation

RB_1, R_1, C_1, RW = 0.8743
RB_2, R_1, C_2, RW = 0.9254
RB_3, R_2, C_1, RW = 0.7142
RB_4, R_1, C_2, RW = 0.2143
RB_5, R_2, C_1, RW = 0.8215

Big Data: Selected Computational Intelligence approaches

BIG DATA with Fuzzy Models

Building the FRB with Chi-FRBCS-BigData-Ave

R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8743$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
…

RB_1

R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.9254$
R_2: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
…

RB_2

R_1: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.7142$
…

RB_3

R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_2; $RW_1 = 0.2143$
R_2: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
…

RB_4

…

R_1: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
R_2: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_2 = 0.8215$
…

RB_n

REDUCE

R_1: IF $A_1 = L_1$ AND $A_2 = L_1$ THEN C_1; $RW_1 = 0.8033$
R_2: IF $A_1 = L_2$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.9142$
R_3: IF $A_1 = L_1$ AND $A_2 = L_2$ THEN C_2; $RW_2 = 0.8842$
R_4: IF $A_1 = L_2$ AND $A_2 = L_1$ THEN C_2; $RW_3 = 0.6534$
R_5: IF $A_1 = L_3$ AND $A_2 = L_2$ THEN C_2; $RW_3 = 0.4715$
R_6: IF $A_1 = L_2$ AND $A_2 = L_3$ THEN C_2; $RW_3 = 0.7784$
…

RB_R

Final RB generation

RB_1, R_1, C_1, RW = 0.8743
RB_2, R_1, C_2, RW = 0.9254
RB_3, R_2, C_1, RW = 0.7142
RB_4, R_1, C_2, RW = 0.2143
RB_5, R_2, C_1, RW = 0.8215

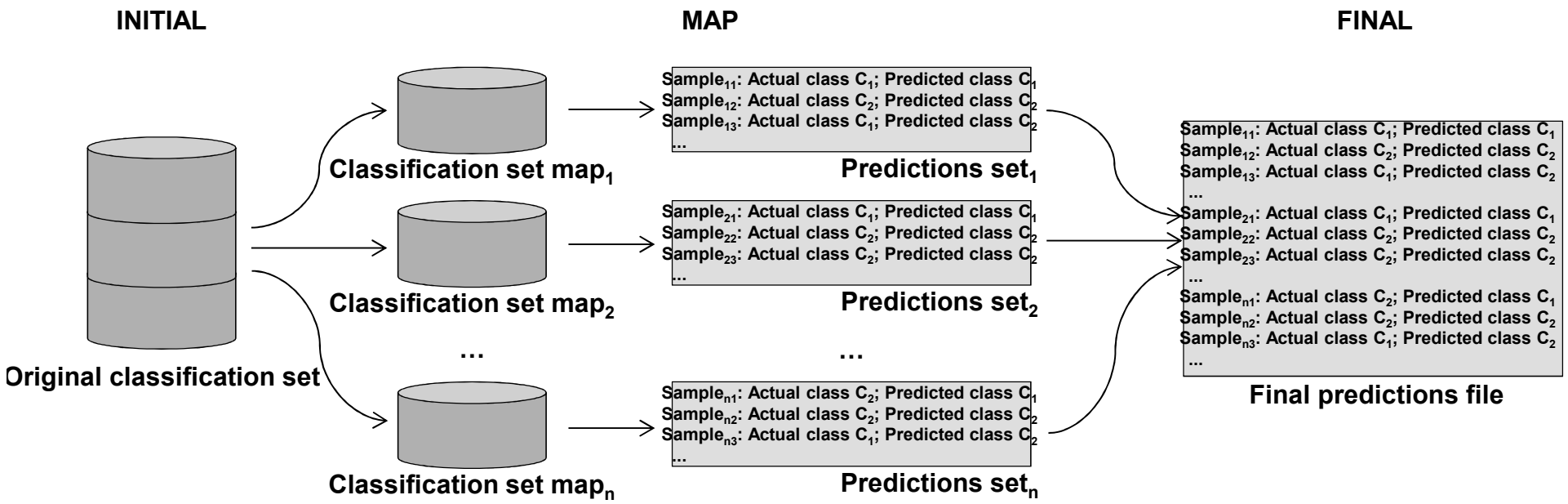
→

RC_1, C_1, $RW_{ave} = 0.8033$
RC_2, C_2, $RW_{ave} = 0.5699$

Big Data: Selected Computational Intelligence approaches

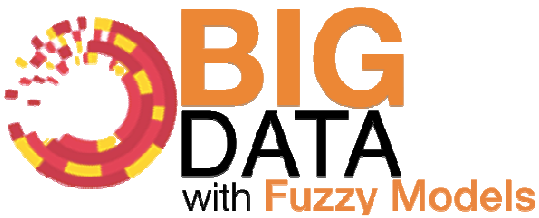
BIG DATA with Fuzzy Models

Estimating the class of a Big dataset with Chi-FRBCS-BigData



Mappers classification sets prediction

Big Data: Selected Computational Intelligence approaches



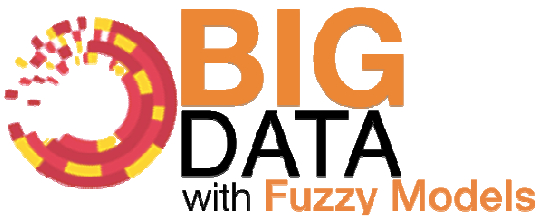
Experimental Analysis: Chi-FRBCS-BigData

- 6 Datasets with two classes problem
- Stratified 10 fold cross-validation
- Parameters:
  - Conjunction Operator: Product T-norm
  - Rule Weight: Penalized Certainty Factor
  - Fuzzy Reasoning Method: Winning Rule
  - Number of fuzzy labels per variable: 3 labels
  - Number of mappers: 16, 32, 64

Experimental Framework

Datasets	#Ex.	#Atts.	Selected classes	#Samples per class
RLCP	5749132	2	(FALSE; TRUE)	(5728201; 20931)
Kddcup_DOS_vs_normal	4856151	41	(DOS; normal)	(3883370; 972781)
Poker_0_vs_1	946799	10	(0; 1)	(513702; 433097)
Covtype_2_vs_1	495141	54	(2; 1)	(283301; 211840)
Census	141544	41	(-_50000.; 50000+.)	(133430; 8114)
Fars_Fatal_Inj_vs_No_Inj	62123	29	(Fatal_Inj; No_Inj)	(42116; 20007)

Big Data: Selected Computational Intelligence approaches

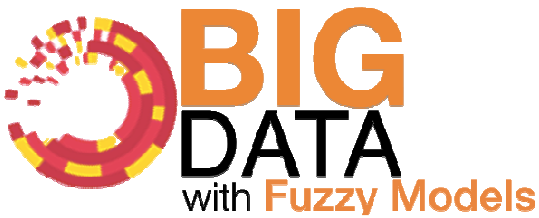


Analysis of the Performance, Precision

Datasets	8 maps					
	Chi-FRBCS		Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
Poker_0_vs_1	63.72	**61.77**	62.93	60.74	63.12	60.91
Covtype_2_vs_1	74.65	74.57	74.69	**74.63**	74.66	74.61
Census	96.52	86.06	97.12	**93.89**	97.12	93.86
Fars_Fatal_Inj_vs_No_Inj	99.66	89.26	97.01	95.07	97.18	**95.25**
Average	83.64	77.92	82.94	81.08	83.02	**81.16**

Good precision!

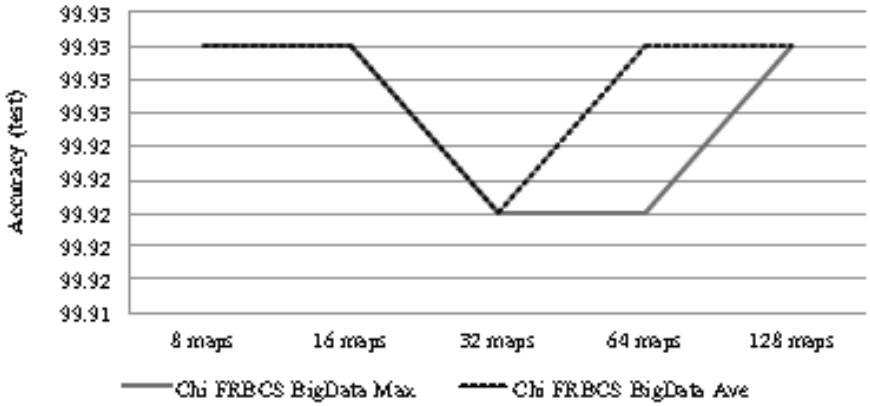
Big Data: Selected Computational Intelligence approaches



Analysis of the Performance, Number of rules

Kddcup_DOS_vs_normal dataset	
NumRules by map	Final numRules
RB_1 size: 211	RB_R size: 301
RB_2 size: 212	
RB_3 size: 221	
RB_4 size: 216	
RB_5 size: 213	
RB_6 size: 210	
RB_7 size: 211	
RB_8 size: 214	

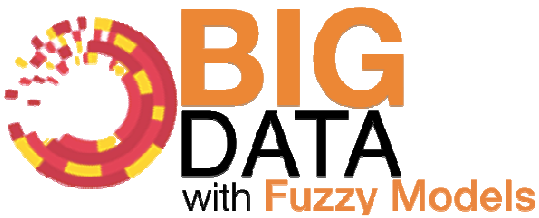
Robustness to the lack of data increasing the final number of rules



(b) Kddcup_DOS_vs_normal dataset

Class	Instance Number
normal	972.781
DOS	3.883.370

Big Data: Selected Computational Intelligence approaches



Analysis of the Performance, Number of rules

Datasets		8 maps	
	Chi-FRBCS	Chi-BigData-Max	Chi-BigData-Ave
	Average NumRules	Average NumRules	Average NumRules
Census	**31518.3**	34278.0	34278.0
Covtype_2_vs_1	**6962.7**	7079.1	7079.1
Fars_Fatal_Inj_vs_No_Inj	**16843.3**	17114.9	17114.9
Poker_0_vs_1	**51265.4**	52798.1	52798.1

Robustness to the lack of data for the data fragmentation, increasing the final number of rules

This may cause a improvement in the performance

Big Data: Selected Computational Intelligence approaches

Analysis of the Performance, Precision

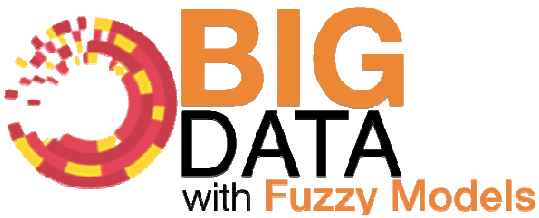
Datasets	16 mappers			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	99.63	99.63	99.63
Kddcup_DOS_vs_normal	99.93	99.93	99.93	99.93
Poker_0_vs_1	62.18	59.88	62.58	60.35
Covtype_2_vs_1	74.77	74.72	74.77	74.69
Census	97.14	93.75	97.15	93.52
Fars_Fatal_Inj_vs_No_Inj	96.69	94.75	97.06	95.01
Average	88.39	87.11	88.52	87.19

Datasets	32 mappers			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	99.63	99.63	99.63
Kddcup_DOS_vs_normal	99.92	99.92	99.92	99.92
Poker_0_vs_1	61.27	58.93	61.82	59.30
Covtype_2_vs_1	74.69	74.62	74.88	74.85
Census	97.11	93.48	97.12	93.32
Fars_Fatal_Inj_vs_No_Inj	96.49	94.26	96.87	94.63
Average	88.19	86.81	88.37	86.94

Datasets	64 mappers			
	Chi-BigData-Max		Chi-BigData-Ave	
	Acc_{tr}	Acc_{tst}	Acc_{tr}	Acc_{tst}
RLCP	99.63	99.63	99.63	99.63
Kddcup_DOS_vs_normal	99.92	99.92	99.93	99.93
Poker_0_vs_1	60.45	57.95	60.88	58.12
Covtype_2_vs_1	74.67	74.52	75.05	74.96
Census	97.07	93.30	97.13	93.11
Fars_Fatal_Inj_vs_No_Inj	96.27	93.98	96.76	94.56
Average	88.00	86.55	88.23	86.72

- **Performance improves slightly with less maps** (alleviate the small sample size problem)
- Chi-BigData-Ave obtains slightly better classification results

Big Data: Selected Computational Intelligence approaches



Analysis of the Performance, Runtime (Chi-BigData-Ave)

Datasets	8 maps		
	Chi-FRBCS	Chi-BigData-Max	Chi-BigData-Ave
	Runtime (s)	Runtime (s)	Runtime (s)
Census	38655.60	**1102.45**	1343.92
Covtype_2_vs_1	86247.70	**2482.09**	2512.16
Fars_Fatal_Inj_vs_No_Inj	8056.60	**241.96**	311.95
Poker_0_vs_1	114355.80	**5672.80**	7682.19
Average	61828.93	**2374.82**	2962.56

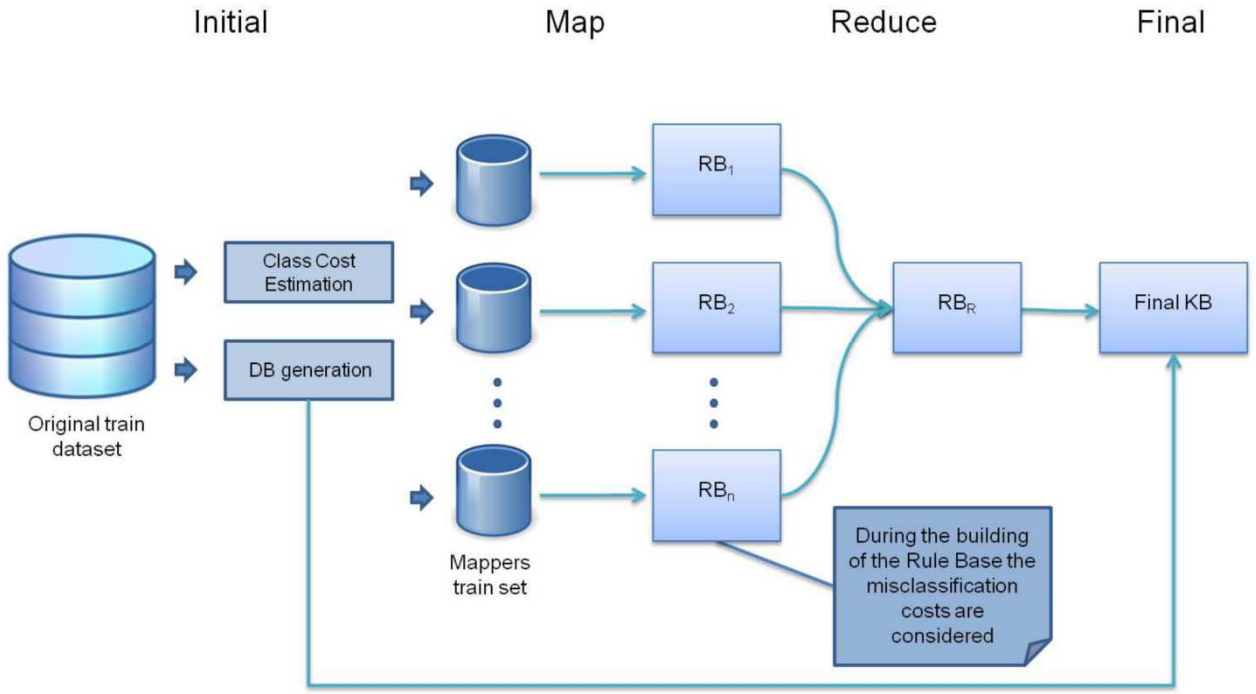
KddCUP'99	
Class	Instance Number
normal	**972.781**
DOS	**3.883.370**

Maps number	Seconds
8	**116.218,26**
16	**29.820,01**
32	**7.708,96**
64	**2.096,34**
132	**1.579,77**

Big Data: Selected Computational Intelligence approaches

FRBCS for Big Data: Model for Imbalanced classes

- Chi-FRBCS-BigDataCS: **algorithm for imbalanced bigdata**



V. López, S. Río, J.M. Benítez, F. Herrera. *Cost-Sensitive Linguistic Fuzzy Rule Based Classification Systems under the MapReduce Framework for Imbalanced Big Data*. Fuzzy Sets and Systems 258 (2015) 5-38.

Big Data Classification with Fuzzy Models

Code for our approaches:

Sara Del Río

Fuzzy Rule Based System for classification

Fuzzy Rule Based System with cost sensitive for imbalanced data sets

https://github.com/saradelrio

Popular repositories

Chi-FRBCS-BigData-Ave
Chi-FRBCS-BigData-Ave: MapReduce implementation of the Chi et al.'s approach.

Chi-FRBCS-BigData-Max
Chi-FRBCS-BigData-Max: MapReduce implementation of the Chi et al.'s approach.

Chi-FRBCS-BigDataCS
Chi-FRBCS-BigDataCS: MapReduce implementation of the basic Chi et al.'s algorithm

hadoop-imbalanced-preprocessing
MapReduce implementations of random oversampling, random undersampling and "Sy
Echnique" (SMOTE) algorithms using Hadoop

RF-BigDataCS
RF-BigDataCS: A cost-sensitive approach for Random Forest MapReduce algorithm to

Big Data Classification with Fuzzy Models

- **Linguistic fuzzy models for Big Data** under the MapReduce framework:
  - Manages big datasets
  - Without damaging the classification accuracy
  - Fast response times (increasing with the number of Maps)

- **Some challenges:**
  - Reduce phase for approximate fuzzy models
  - Deep analysis "ensembles vs fusion of rules"
  - Deep analysis on the small disjuncts preprocessing for fuzzy models
  - New fuzzy models based on accurate algorithms

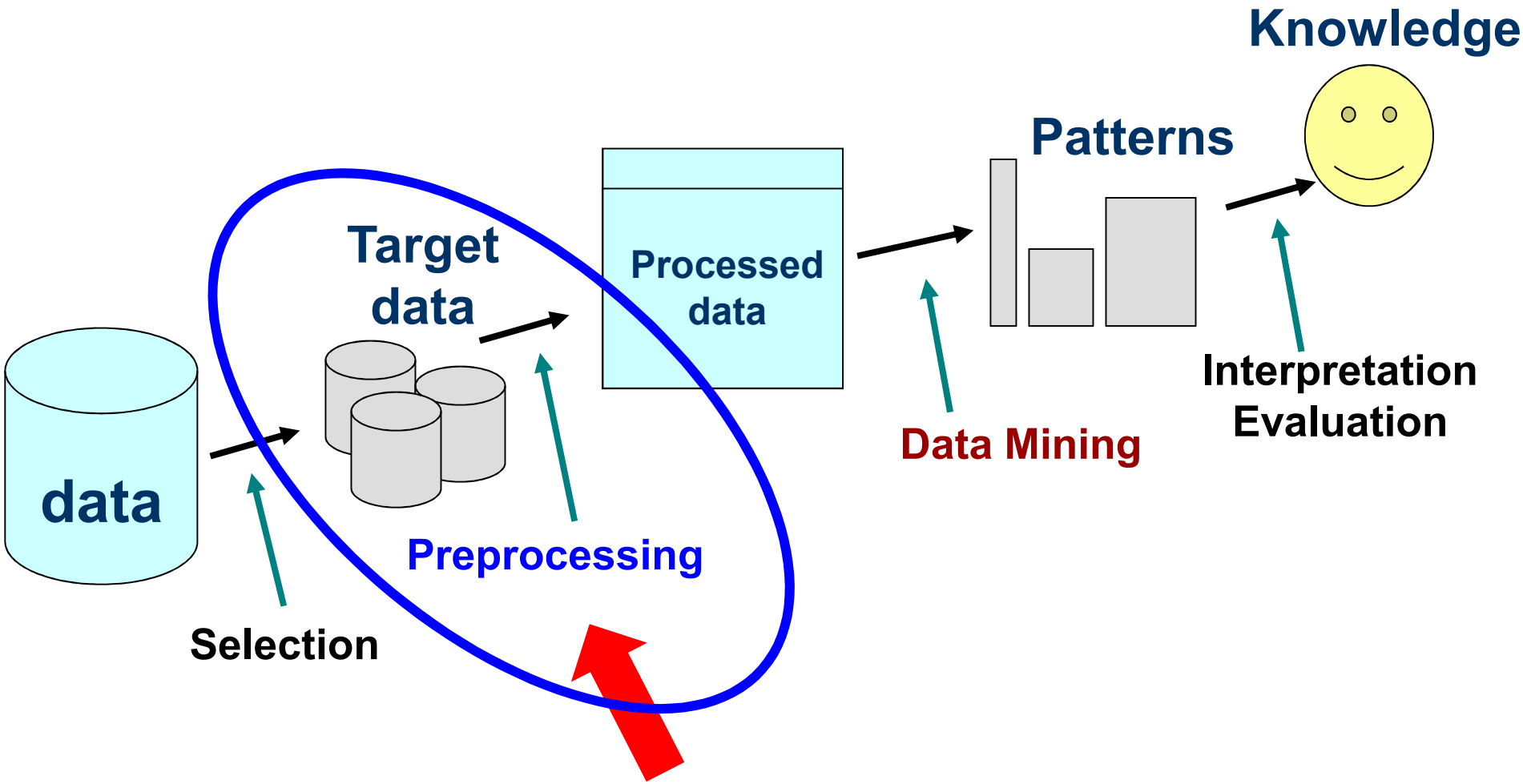
- **A promising line of work for the design of high performance Fuzzy Models for Big Data**

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competitic
- Final Comments

Data Preprocessing for Big Data

Data Preprocessing: Tasks to discover quality data prior to use knowledge extraction algorithms.



Data Preprocessing

D. Pyle, 1999, pp. 90:

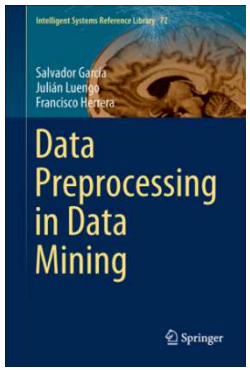
"The fundamental purpose of *data preparation* is to manipulate and transform raw data so that the information content enfolded in the data set can be exposed, or made more easily accesible."

Dorian Pyle
Data Preparation for Data Mining Morgan Kaufmann Publishers, 1999

Data Preprocessing

S. García, J. Luengo, F. Herrera, 2015, Preface vii:

"Data preprocessing is an often neglected but major step in the data mining process."



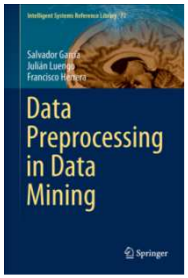
S. García, J. Luengo, F. Herrera
Data Preprocessing in Data Mining
Springer, January 2015
Website:
http://sci2s.ugr.es/books/data-preprocessing

Data Preprocessing

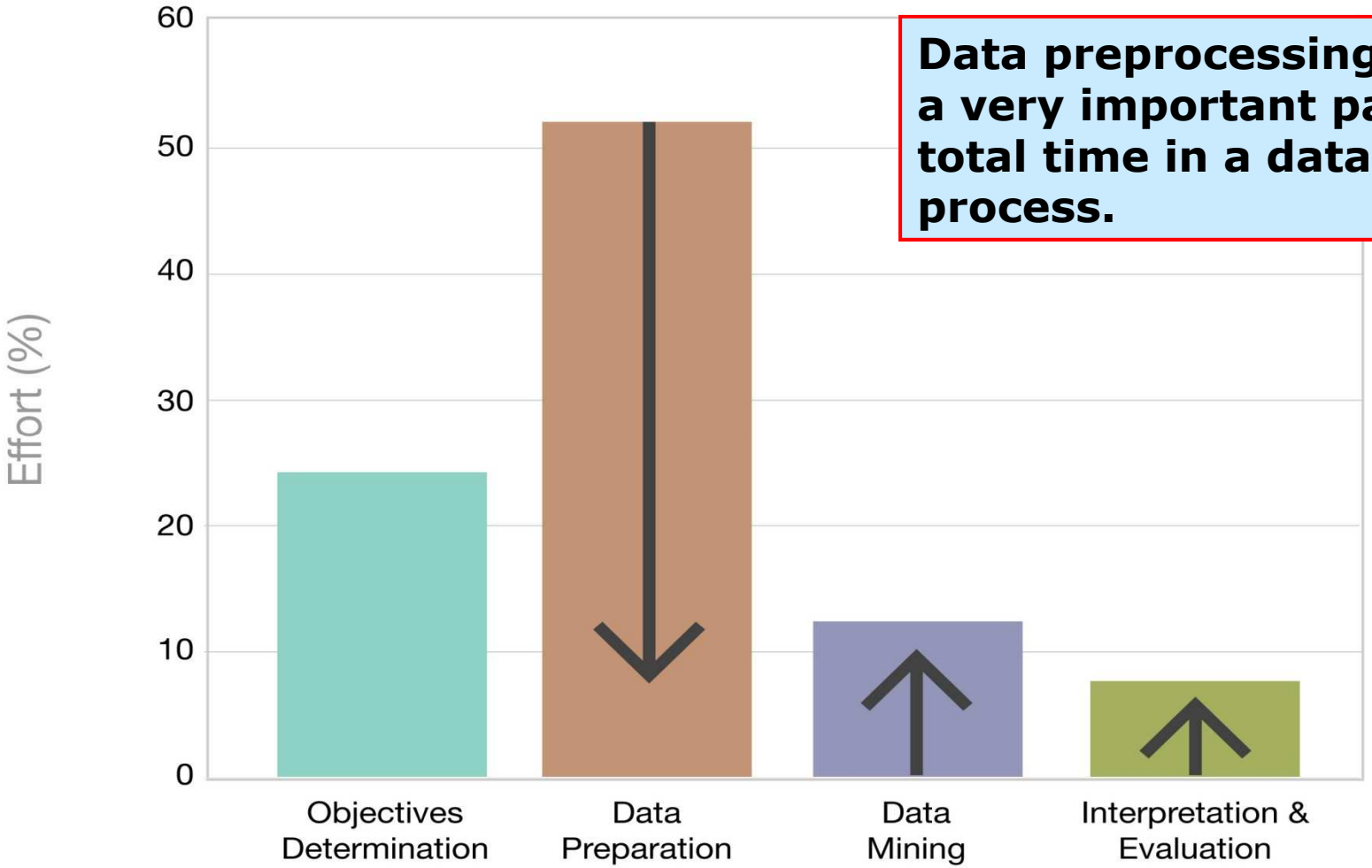
S. García, J. Luengo, F. Herrera, 2015, Preface viii:

There are many advantages that data preprocessing provides:

I. To adapt and particularize the data for each data mining algorithm.
II. To increase the effectiveness and accuracy in predictive tasks.
III. To reduce the amount of data required for a suitable learning task, also decreasing its time-complexity.
IV. To make possible the impossible with raw data, allowing data mining algorithms to be applied over high volumes of data.
V. To support to the understanding of the data.
VI. Useful for various tasks, such as classification, regression and unsupervised learning…



Data Preprocessing

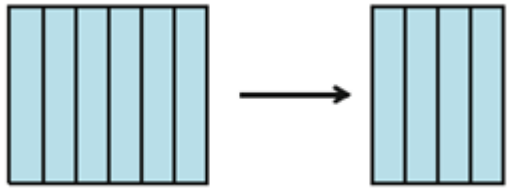


Data preprocessing spends a very important part of the total time in a data mining process.

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
  - Feature Selection
  - Instance Reduction
  - Undersampling for Imbalanced problems
- A case of study: the ECBDL'14 competition
- Final Comments

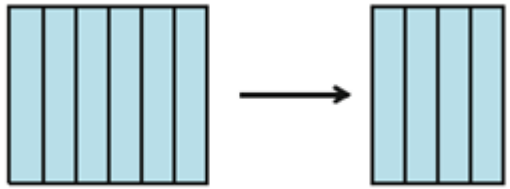
Feature Selection



The problem of *Feature Subset Selection (FSS*)

consists of finding a subset of the attributes/features/variables of the data set that optimizes the probability of success in the subsequent data mining taks.

Feature Selection



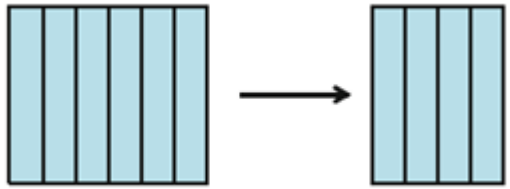
Var. 1.

Var. 5

Var. 13

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
E	0	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0
F	1	1	1	0	1	1	0	0	1	0	1	0	0	1	0	0

Feature Selection

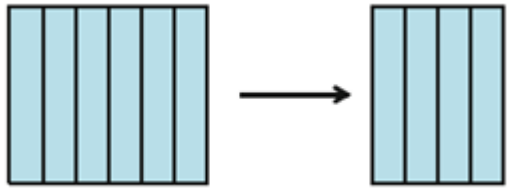


The problem of *Feature Subset Selection (FSS*) consists of finding a subset of the attributes/features/variables of the data set that optimizes the probability of success in the subsequent data mining taks.

Why is feature selection necessary?

- More attributes do not mean more success in the data mining process.
- Working with less attributes reduces the complexity of the problem and the running time.
- With less attributes, the generalization capability increases.
- The values for certain attributes may be difficult and costly to obtain.

Feature Selection



- **The outcome of FS would be:**
  - **Less data → algorithms could learn quicker**
  - **Higher accuracy → the algorithm generalizes better**
  - **Simpler results → easier to understand them**
- **FS has as extension the extraction and construction of attributes.**

Big Data Preprocessing: MR-EFS

Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach

D. Peralta, S. del Río, S. Ramírez-Gallego, I. Triguero, J.M. Benítez, F. Herrera. Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach. Mathematical Problems in Engineering, Vol. 2015, Article ID 246139, 11 pages, 2015, doi: 10.1155/2015/246139

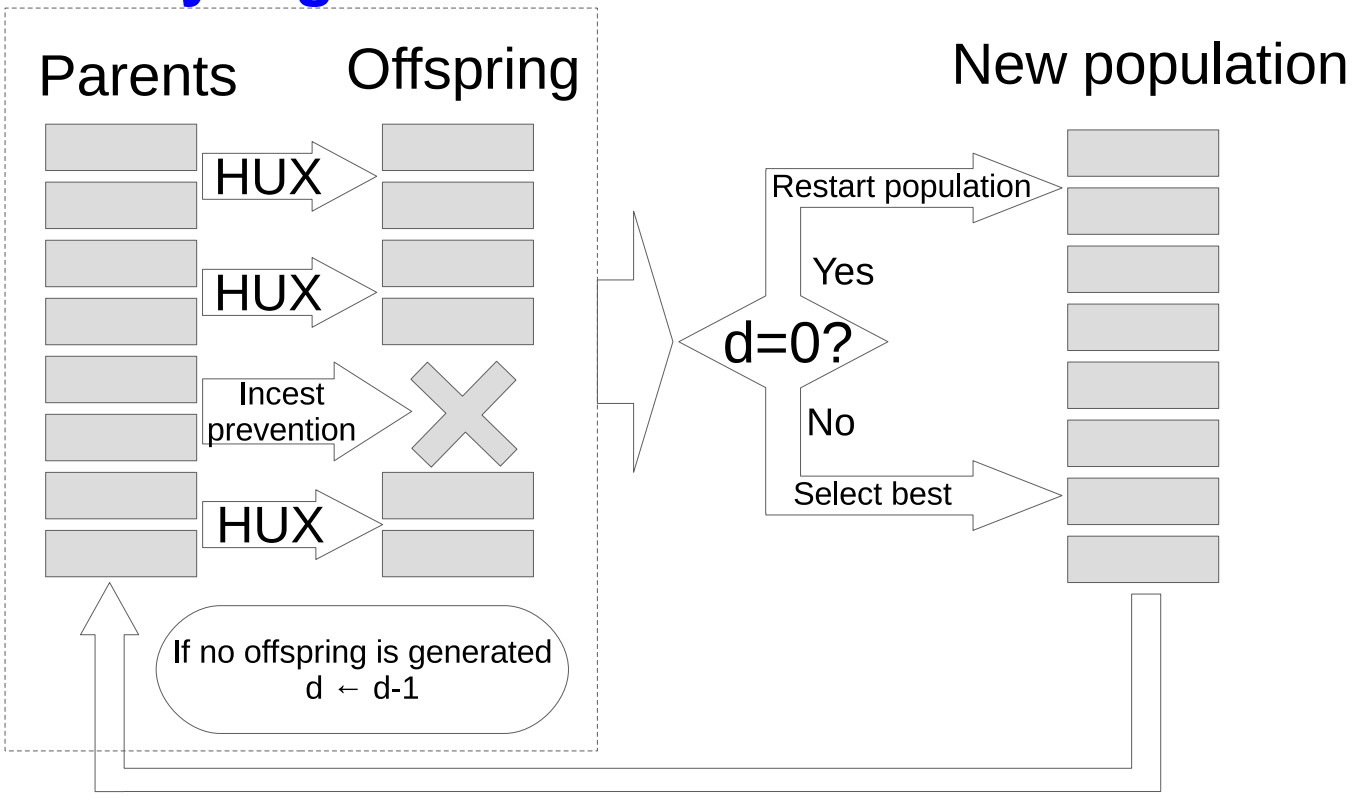
Big Data Preprocessing: MR-EFS

Evolutionary Feature Selection (EFS)

- Each individual represents a set of selected features (binary vector).
- The individuals are crossed and mutated to generate new candidate sets of features.
- Fitness function:
  - Classification performance in the training dataset using only the features in the corresponding set.

Big Data Preprocessing: MR-EFS

Evolutionary Algorithm: CHC



L. J. Eshelman, The CHC adaptative search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in: G. J. E. Rawlins (Ed.), Foundations of Genetic Algorithms, 1991, pp. 265--283.

Big Data Preprocessing: MR-EFS

Parallelizing FS with MapReduce

Map phase

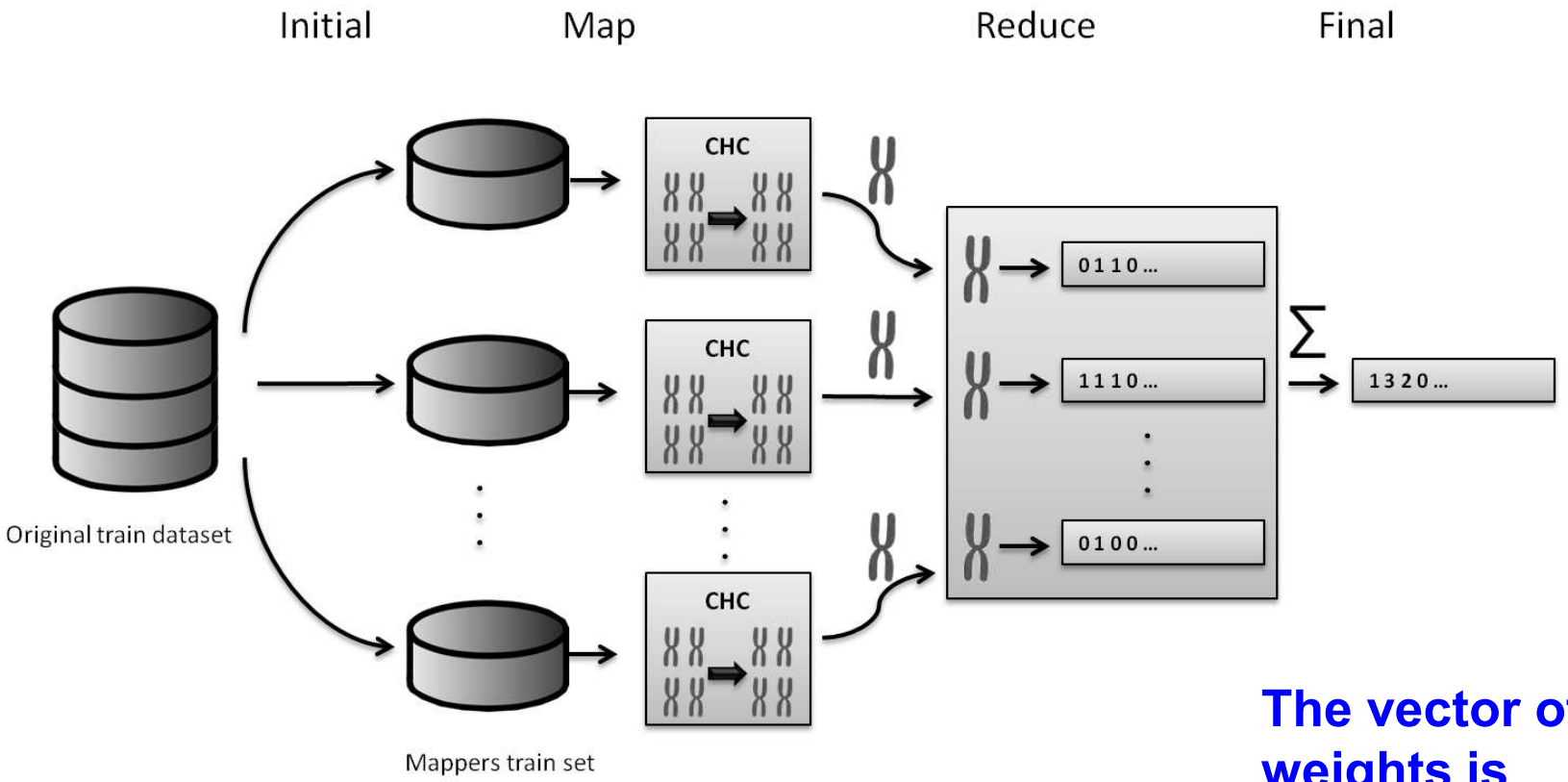
- Each map task uses a subset of the training data.
- It applies an EFS algorithm (CHC) over the subset.
- A k-NN classifier is used for the evaluation of the population.
- Output (best individual):
  - Binary vector, indicating which features are selected.

Reduce phase

- One reducer.
- It sums the binary vectors obtained from all the map tasks.
- The output is a vector of integers.
  - **Each element is a weight for the corresponding feature.**

Big Data Preprocessing: MR-EFS

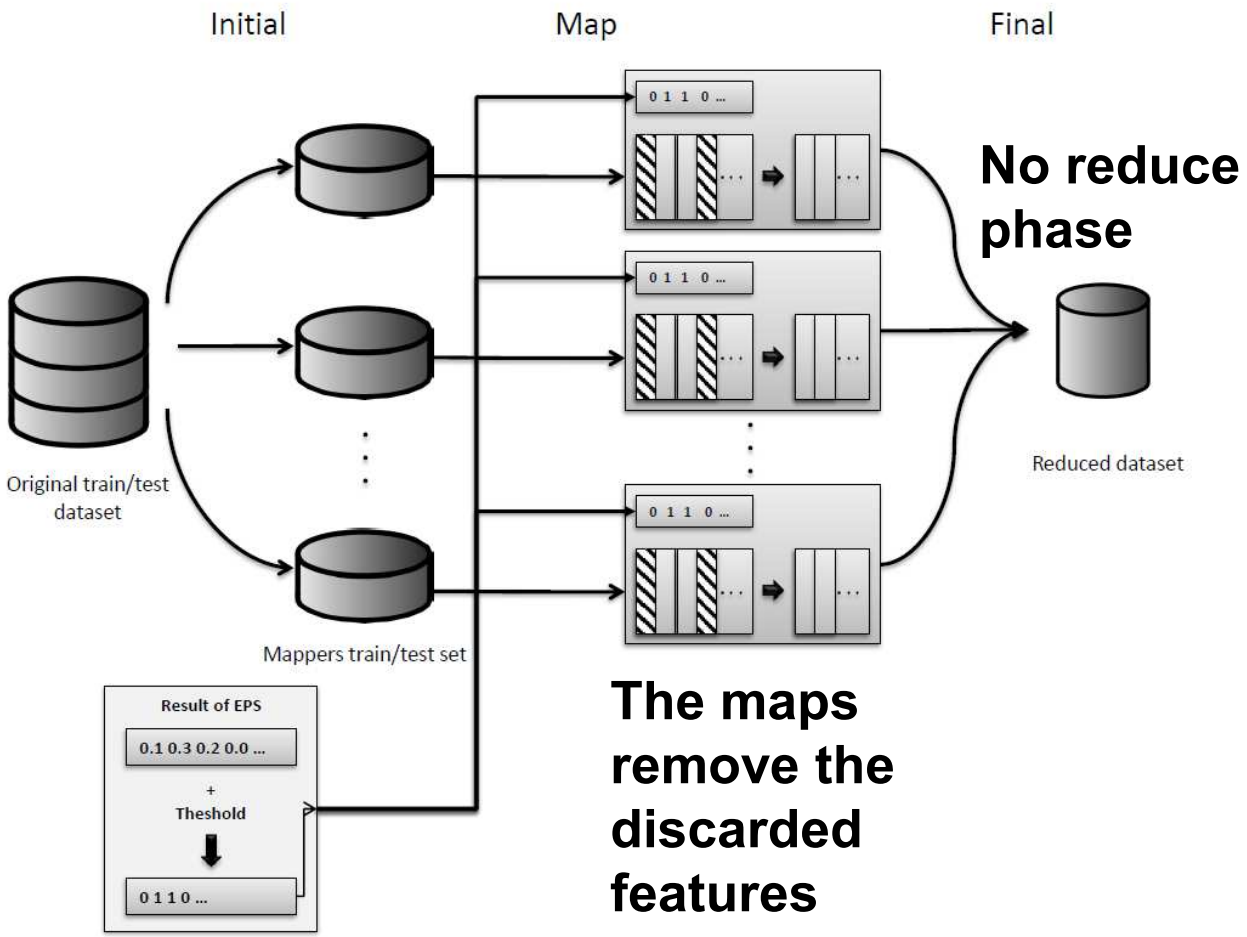
MapReduce EFS process



The vector of weights is binarized with a threshold

Big Data Preprocessing: MR-EFS

Dataset reduction

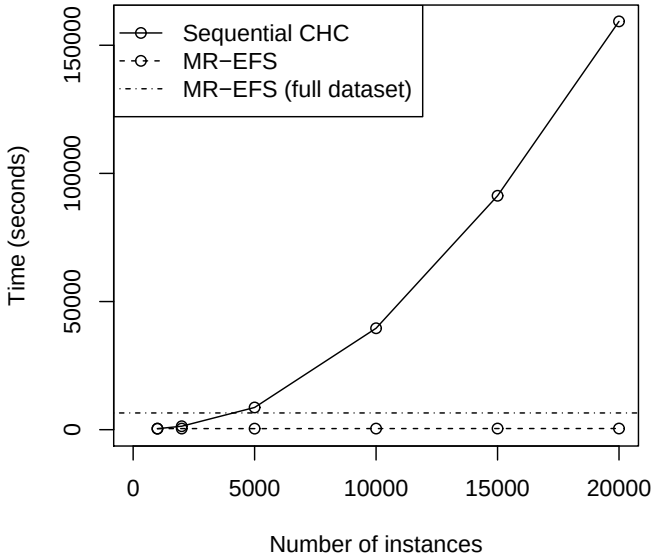


Big Data Preprocessing: MR-EFS

Experimental Study: EFS scalability in MapReduce

Table 3: Execution times (in seconds) over the epsilon subsets

Instances	Sequential CHC	MR-EFS	Splits
1000	391	419	1
2000	1352	409	2
5000	8667	413	5
10 000	39 576	431	10
15 000	91 272	445	15
20 000	159 315	455	20
400 000	–	6531	512



- CHC is quadratic w.r.t. the number of instances
- Splitting the dataset yields nearly quadratic acceleration

Big Data Preprocessing: MR-EFS

Experimental Study: Classification

- Two datasets
  - epsilon
  - ECBDL14, after applying Random Oversampling
- The reduction rate is controlled with the weight threshold
- Three classifiers in Spark
  - SVM
  - Logistic Regression
  - Naïve Bayes
- Performance measures
  - $\text{AUC} = \frac{\text{TPR} + \text{TNR}}{2}$
  - Training runtime

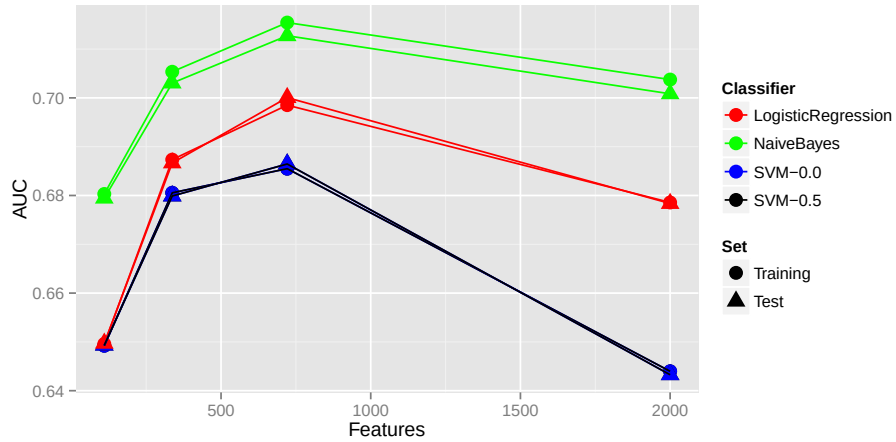
Dataset	Training instances	Test instances	Features	Splits	Instances per split
epsilon	400 000	100 000	2000	512	~780
ECBDL14	31 992 921	2 897 917	631	–	–
ECBDL14-ROS	65 003 913	2 897 917	631	32 768	~1984

Big Data Preprocessing: MR-EFS

Experimental Study: results

Table 4: AUC results for the Spark classifiers using epsilon

		Logistic Regression		Naive Bayes		SVM ($\lambda = 0.0$)		SVM ($\lambda = 0.5$)	
Threshold	**Features**	Training	Test	Training	Test	Training	Test	Training	Test
0.00	2000	0.6786	0.6784	0.7038	0.7008	0.6440	0.6433	0.6440	0.6433
0.55	721	**0.6985**	**0.7000**	**0.7154**	**0.7127**	**0.6855**	**0.6865**	**0.6855**	**0.6865**
0.60	337	0.6873	0.6867	0.7054	0.7030	0.6805	0.6799	0.6805	0.6799
0.65	110	0.6496	0.6497	0.6803	0.6794	0.6492	0.6493	0.6492	0.6493



Big Data Preprocessing: MR-EFS

Experimental Study: Feature selection scalability

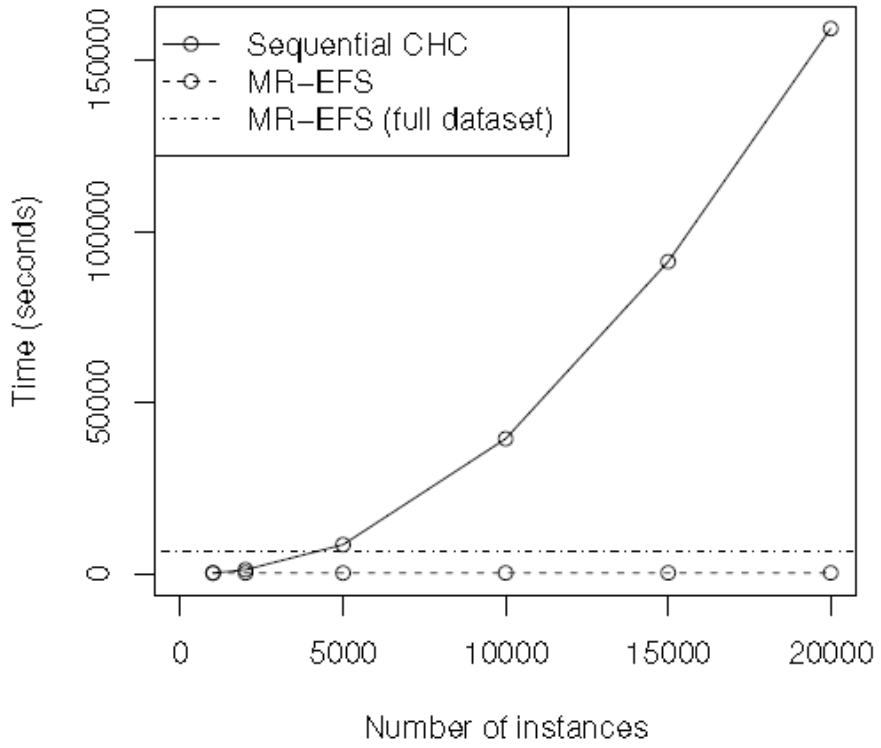


Figure 5: Execution times of the sequential CHC and MR-EFS.

Big Data Preprocessing: MR-EFS

EFS for Big Data: Final Comments

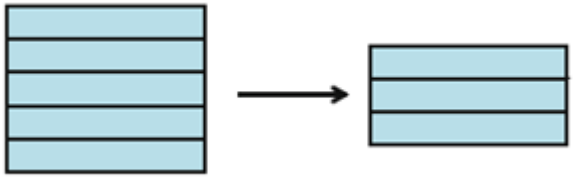
- The splitting of CHC provides several advantages:
  - It enables tackling Big Data problems
  - The speedup of the map phase is nearly quadratic
  - The feature weight vector is more flexible than a binary vector
- The data reduction process in MapReduce provides a scalable and flexible way to apply the feature selection
- Both the accuracy and the runtime of the classification were improved after the preprocessing.

https://github.com/triguero/MR-EFS

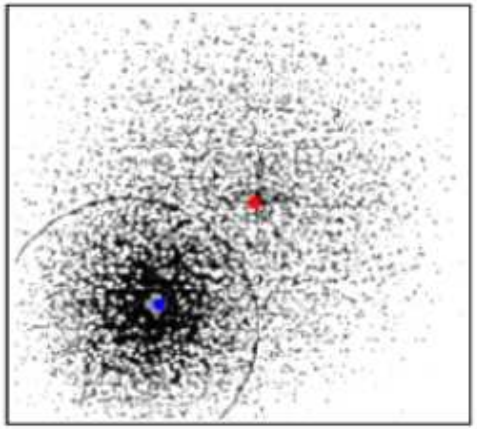
Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
  - Feature Selection
  - Instance Reduction
  - Undersampling for Imbalanced problems
- A case of study: the ECBDL'14 competition
- Final Comments

Instance Reduction

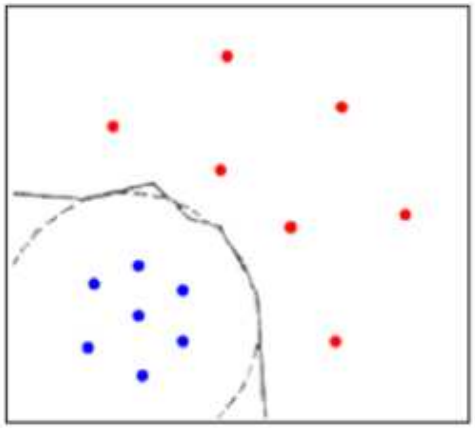


- Instance Reduction methods aim to reduce the number of training samples to find better and smoother decision boundaries between classes, by **selecting** relevant training samples or artificially **generating** new ones.



- Advantages:

  - ✓ ***Reduce Storage Requirements***
  - ✓ ***Remove noisy samples***
  - ✓ ***Speed up learning process***



Big Data Preprocessing: MRPR

Prototype Generation: properties

- The NN classifier is one of the most used algorithms in machine learning.
- Prototype Generation (PG) processes learn new representative examples if needed. It results in more accurate results.
- Advantages:
  - PG reduces the computational costs and high storage requirements of NN.
  - Evolutionary PG algorithms highlighted as the best performing approaches.
- *Main issues:*
  - Dealing with big data becomes impractical in terms of *Runtime* and *Memory consumption.* **Especially for Evolutionary PG models**.

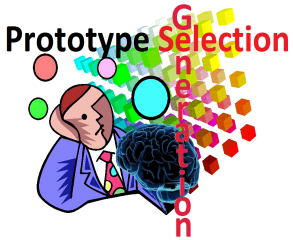
Big Data Preprocessing: MRPR

Evolutionary Prototype Generation

- Evolutionary PG algorithms are typically based on adjustment of the positioning of the prototypes.
- Each individual encodes a single prototype or a complete generated set with real codification.
- The fitness function is computed as the classification performance in the training set using the Generated Set.
- Currently, best performing approaches use differential evolution.

I. Triguero, S. García, F. Herrera, IPADE: Iterative Prototype Adjustment for Nearest Neighbor Classification. *IEEE Transactions on Neural Networks 21 (12) (2010) 1984-1990*

More information about Prototype Reduction can be found in the SCI2S thematic website: http://sci2s.ugr.es/pr



Big Data Preprocessing:MRPR

MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation

I. Triguero, D. Peralta, J. Bacardit, S.García, F. Herrera. A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation. Evolutionary Computation (CEC), 2014 IEEE Congress on, 3036-3043

Big Data Preprocessing: MRPR

Parallelizing PG with MapReduce

Map phase:

- Each map constitutes a subset of the original training data.
- It applies a Prototype Generation step.
- For evalution, it uses Windowing: Incremental Learning with Alternating Strata (ILAS)
- As output, it returns a Generated Set of prototypes.

Reduce phase:

- We established a single reducer.
- It consists of an iterative aggregation of all the resulting generated sets.
- As output, it returns the final Generated Set.

Big Data Preprocessing: MRPR

Parallelizing PG with MapReduce

The key of a MapReduce data partitioning approach is usually on the reduce phase.

Two alternative reducers:

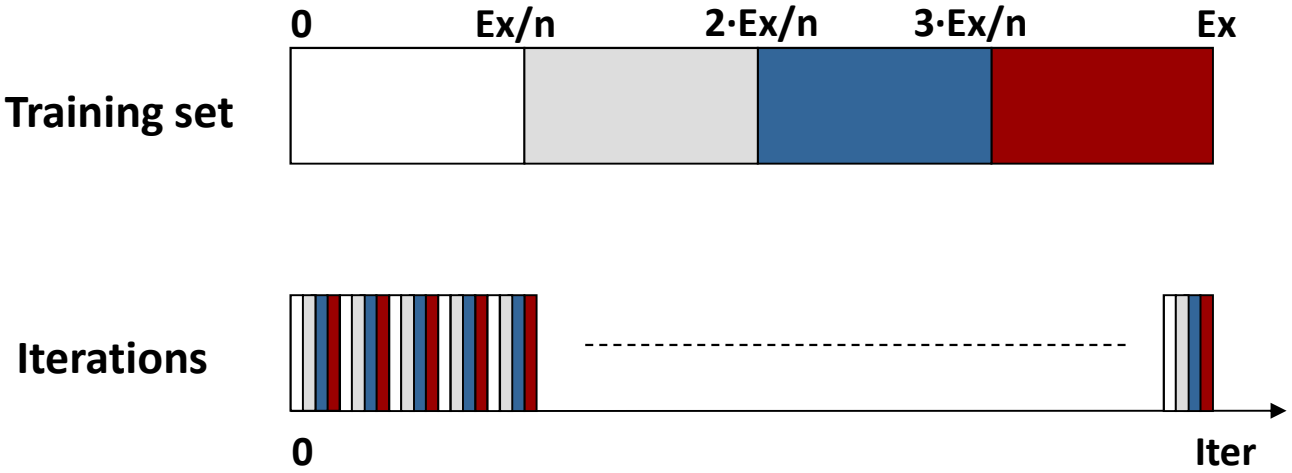
- **Join:** Concatenates all the resulting generated sets.
  - This process does not guarantee that the final generated set does not contain irrelevant or even harmful instances
- **Fusion**: This variant eliminates redundant prototypes by fusion of prototypes. Centroid-based PG methods: ICPL2 (Lam et al).

W. Lam et al, Discovering useful concept prototypes for classification based on filtering and abstraction. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 8, pp. 1075-1090, 2002

Big Data Preprocessing: MRPR

Windowing: Incremental Learning with Alternating Strata (ILAS)

- Training set is divided into strata, each iteration just uses one of the stratum.



Main properties:

✓Avoids a (potentially biased) static prototype selection

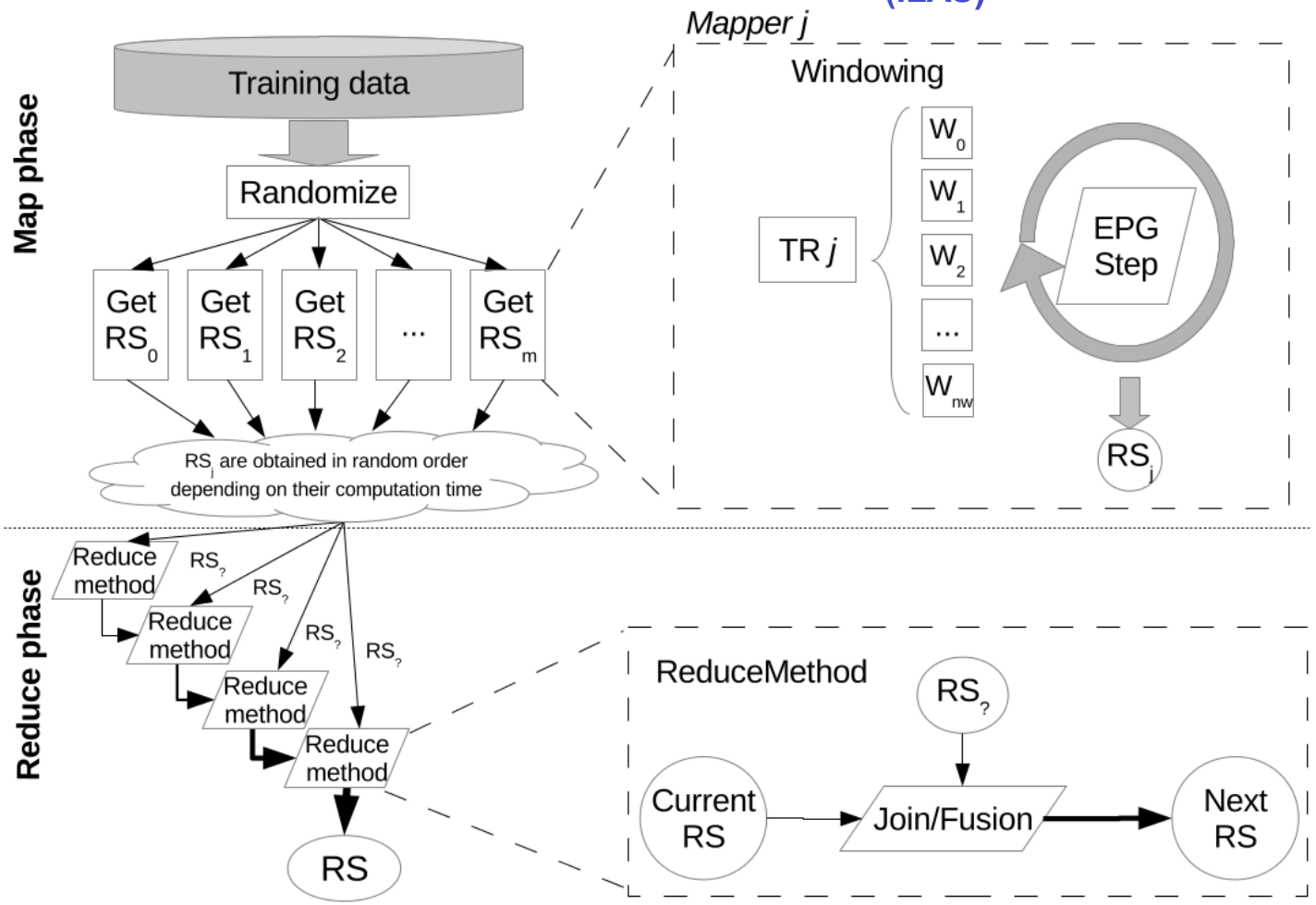
✓This mechanism also introduces some generalization pressure

J. Bacardit et al, Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy In Parallel Problem Solving from Nature - PPSN VIII, ser. LNCS, vol. 3242, 2004, pp. 1021–1031

Big Data Preprocessing: MRPR

The MRW-EPG scheme

Windowing: Incremental Learning with Alternating Strata (ILAS)



Big Data Preprocessing: MRPR

Experimental Study

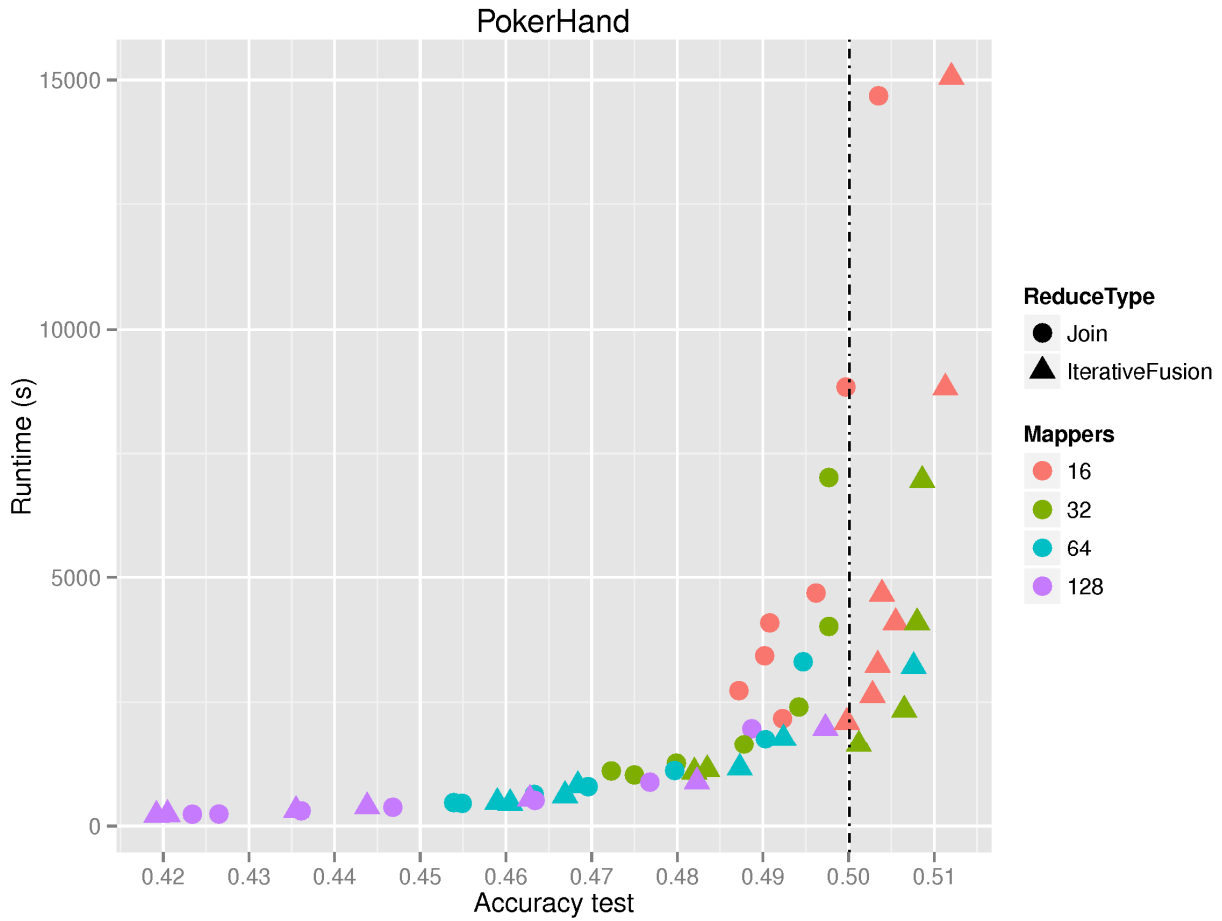
- **PokerHand data set. 1 million of instances, 3x5 fcv.**
- **Performance measures: Accuracy, reduction rate, runtime, test classification time and speed up.**
- **PG technique tested: IPADECS.**

TABLE I: Parameter specification for all the methods

Algorithm	*Parameters*
MRW-EPW	Mappers = 16/32/64/128, Reducers= 1 Windows = [1-7], ReduceType = Join/Fusion.
IPADECS ICLP2 (Fusion)	PopulationSize = 10, iterations of Basic DE = 500 iterSFGSS =8, iterSFHC=20, Fl=0.1, Fu=0.9 Filtering method = RT2
NN	Number of neighbors = 1, Euclidean distance.

Big Data Preprocessing: MRPR

Results



PokerHand: Accuracy Test vs. Runtime results obtained by MRW-EPG

Big Data Preprocessing: MRPR

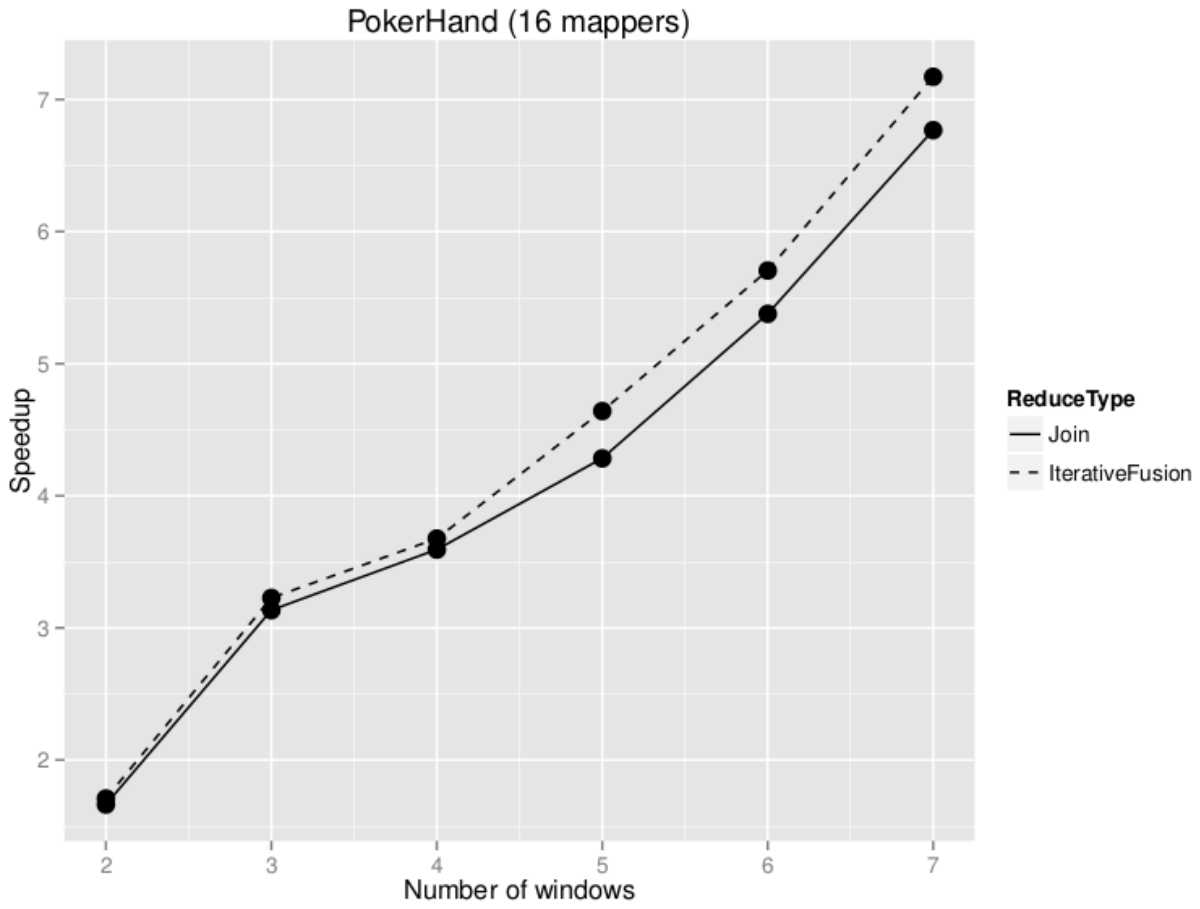
Results

TABLE III: Results obtained incorporating the windowing scheme with MRW-EPG and fusion reducer.

#Windows nw	#Mappers	Training		Test		Runtime		Reduction rate		Classification time (TS)
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	
1	16	0.5121	0.0028	0.5120	0.0031	15058.4740	1824.6586	99.9863	0.0007	26.2472
2	16	0.5115	0.0035	0.5113	0.0036	8813.7134	678.1335	99.9875	0.0007	23.8804
3	16	0.5038	0.0032	0.5039	0.0033	4666.5424	412.5351	99.9883	0.0010	26.5612
4	16	0.5052	0.0060	0.5055	0.0057	4095.8610	941.5737	99.9890	0.0011	25.8442
5	16	0.5041	0.0024	0.5034	0.0022	3244.0716	534.8720	99.9899	0.0015	25.0526
6	16	0.5031	0.0042	0.5028	0.0041	2639.4266	360.3121	99.9905	0.0011	26.6988
7	16	0.5000	0.0067	0.4998	0.0069	2099.5182	339.7356	99.9895	0.0010	25.8770
1	32	0.5089	0.0031	0.5086	0.0029	6963.5734	294.3580	99.9772	0.0018	28.1252
2	32	0.5084	0.0045	0.5080	0.0041	4092.5484	855.7351	99.9789	0.0016	30.6644
3	32	0.5067	0.0025	0.5065	0.0024	2343.1542	104.7222	99.9794	0.0012	33.6744
4	32	0.5012	0.0045	0.5012	0.0039	1639.0032	335.6036	99.9785	0.0015	26.8272
5	32	0.5012	0.0045	0.5012	0.0039	1639.0032	335.6036	99.9785	0.0015	26.8272
6	32	0.4824	0.0104	0.4820	0.0101	1083.1116	143.9288	99.9768	0.0019	35.1896
7	32	0.4838	0.0072	0.4835	0.0065	1129.8838	173.9482	99.9757	0.0024	35.4692

Big Data Preprocessing: MRPR

Results: Speed-up



Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

- There is a good synergy between the windowing and MapReduce approaches. They complement themselves in the proposed two-level scheme.
- Without windowing, evolutionary prototype generation could not be applied to data sets larger than approximately ten thousands instances
- The application of this model has resulted in a very big reduction of storage requirements and classification time for the NN rule.

Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

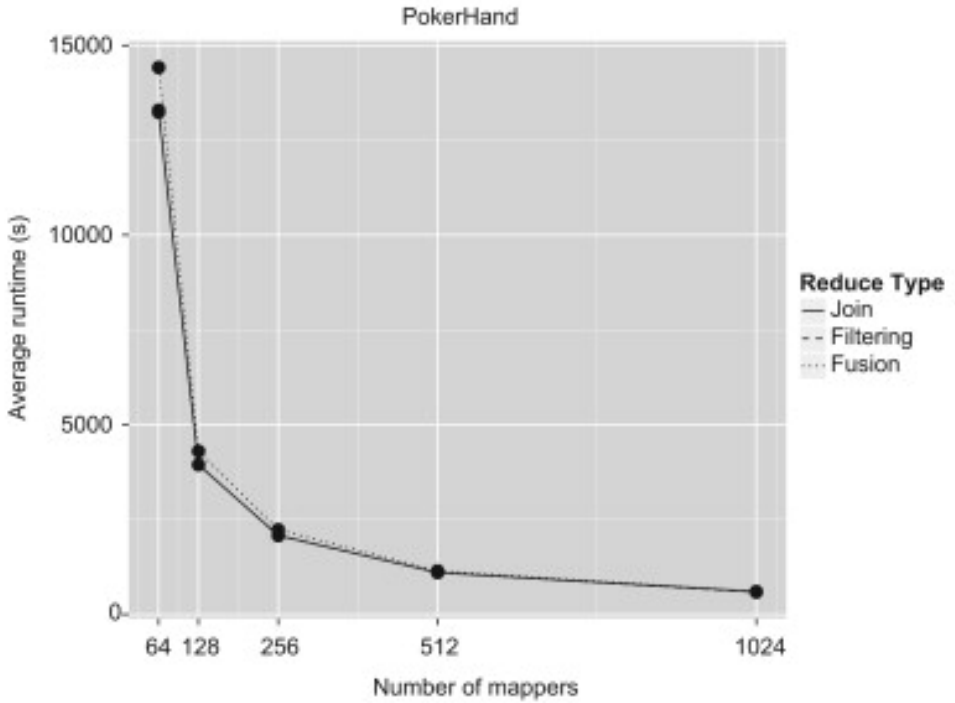


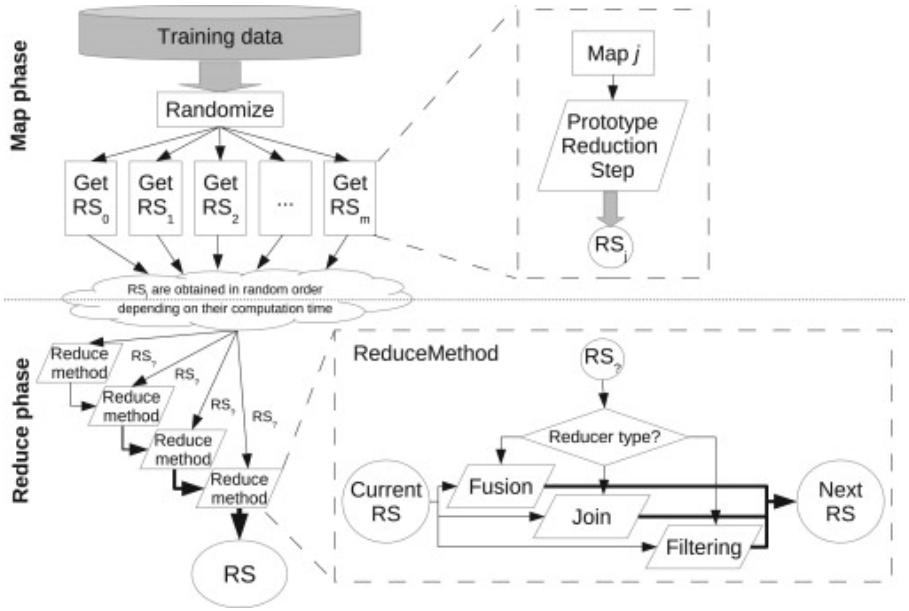
Fig. 6 Average runtime obtained by MRPR. (a) PokerHand

Complete study: I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera. MRPR: A MapReduce solution for prototype reduction in big data classification. ***Neurocomputing 150 (2015) 331–345.***

Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

- **Complete study:** I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera. MRPR: A MapReduce solution for prototype reduction in big data classification. *Neurocomputing 150 (2015) 331–345.*



https://github.com/triguero/MRPR

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
  - Feature Selection
  - Instance Reduction
  - Undersampling for Imbalanced problems
- A case of study: the ECBDL'14 competition
- Final Comments

Class imbalance problem

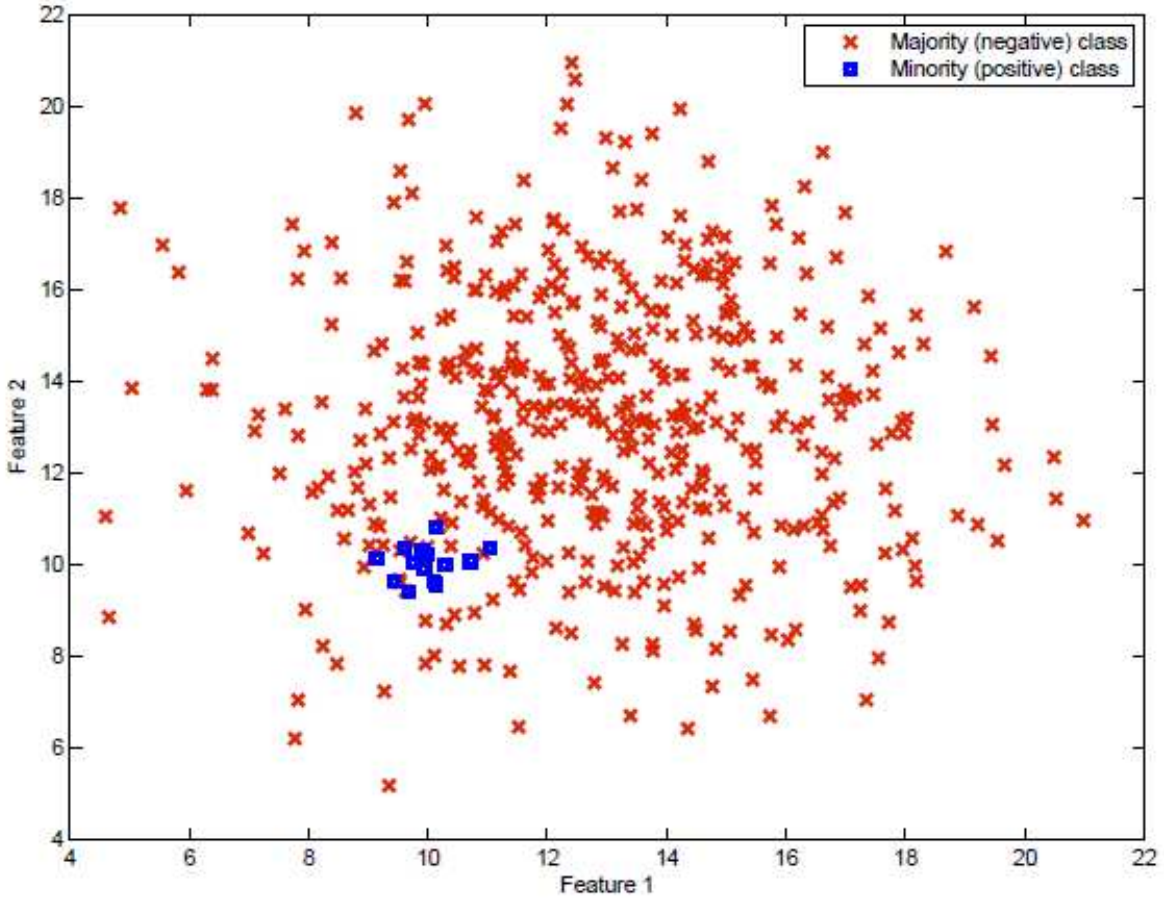
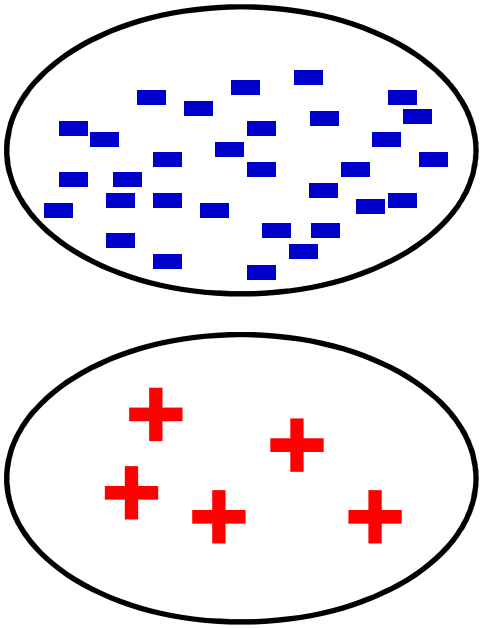


FIGURE : An example of an imbalanced data-set

Class imbalance problem

- Two main approaches to tackle this problem:
- Data sampling
- Undersampling
- Oversampling
- Hybrid approaches
- Algorithmic modifications



V. López et al, An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics, Information Sciences 250 (2013) 113–141

Evolutionary Undersampling

- Evolutionary undersampling (EUS) aims to select the best subset of instances from the original training set.
- EUS not only intends to balance the training set, but also to increase the overall performance on both classes of the problem.
- To do so, a genetic algorithm is used to search for an optimal subset of instances.
- This resulting set can be used by any standard classification model.

S. Garcia and F. Herrera, "Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy", Evolutionary Computation, 17 (3) (2009). 275–306

Evolutionary Undersampling in the big data context

- The application of EUS in big data problems is interesting because it does not generate more data, as opposed to oversampling methods.
- However, the increasing number of instances would lead to obtain an excessive chromosome size that can limit their application.
- The required runtime increases not only with the number of examples but also with the imbalance ratio (IR).

Imbalanced Big Data

Evolutionary Undersampling for Imbalanced Big Data

I Triguero, M Galar, S Vluymans, C Cornelis, H Bustince, F Herrera, I. Saeys.
Evolutionary undersampling for imbalanced big data classification.
IEEE Congress onEvolutionary Computation (CEC), 2015, 715-722.

Evolutionary Undersampling

- Evolutionary undersampling (EUS) aims to select the best subset of instances from the original training set.
- EUS not only intends to balance the training set, but also to increase the overall performance on both classes of the problem.
- To do so, a genetic algorithm is used to search for an optimal subset of instances.
- This resulting set can be used by any standard classification model.

S. Garcia and F. Herrera, "Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy", Evolutionary Computation, 17 (3) (2009). 275–306

Evolutionary Undersampling in the big data context

- The application of EUS in big data problems is interesting because it does not generate more data, as opposed to oversampling methods.
- However, the increasing number of instances would lead to obtain an excessive chromosome size that can limit their application.
- The required runtime increases not only with the number of examples but also with the imbalance ratio (IR).

Evolutionary undersampling

- **Representation of the solution**

$$V = (v_{x_1}, v_{x_2}, v_{x_3}, v_{x_4}, \ldots, v_{x_{n^-}}), \; v_{x_i} \in \{0,1\} \text{ for all } i = 1, \ldots, n^-$$

- **Performance**: g-mean, 1NN hold-one-out

$$\text{g-mean} = \sqrt{\text{TP}_{rate} \cdot \text{TN}_{rate}}$$

- **Fitness Function**

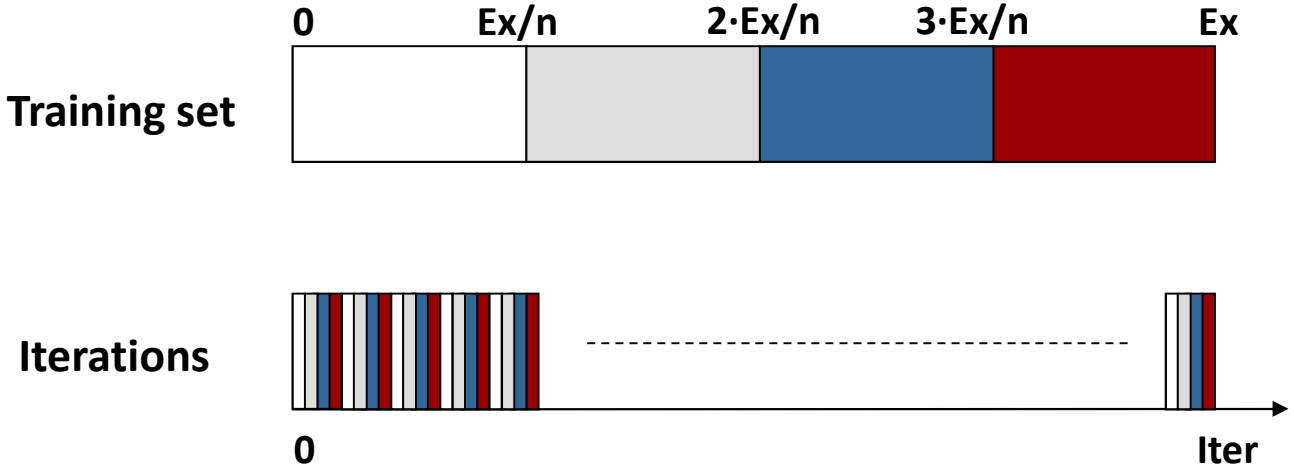
$$\text{fitness}_{\text{EUS}} = \begin{cases} \text{g-mean} - \left|1 - \frac{n^+}{N^-} \cdot P\right| & \text{if } N^- > 0 \\ \text{g-mean} - P & \text{if } N^- = 0, \end{cases}$$

- We use the CHC algorithm and GM as performance measure.

L. J. Eshelman, **The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination**, in Foundations of Genetic Algorithms, G. J. E. Rawlins, Ed. San Francisco, CA: Morgan Kaufmann, 265-283, 1991.

Windowing: Incremental Learning with Alternating Strata (ILAS)

- Training set is divided into strata, each iteration just uses one of the stratum.



Main properties:

✓Avoids a (potentially biased) static prototype selection
✓This mechanism also introduces some generalization pressure

J. Bacardit et al, **Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy** In Parallel Problem Solving from Nature - PPSN VIII, ser. LNCS, vol. 3242, 2004, pp. 1021–1031

EUS-BD: A two-level parallelization model for EUS

- A two-level parallelization scheme:
  - The MapReduce phase will allow us to divide the computational effort over different machines.

    Goal: Memory limitations and runtime.

  - The windowing scheme will be applied to reduce the computational time required by EUS.

    Goal: Runtime.

Parallelizing EUS with MapReduce

Map phase:

- Each map constitutes a subset of the original training data.
- Then, it applies a EUS step (with/without windowing).
- It builds a model with the corresponding base classifier (Decision tree).
- As output, it returns the built model (a tree).

Reduce phase:

- We established a single reducer.
- It consists of an iterative aggregation of all the resulting models.
- As output, it returns a set of trees.

Parallelizing EUS with windowing: Motivation

- The MapReduce algorithm allows us to use EUS with big data problems. However, the runtime required for each mapper can still be too long and highly depending on the ratio of imbalance.

- For this reason, we design a second level parallelization based on a windowing scheme.

Windowing for Class Imbalance

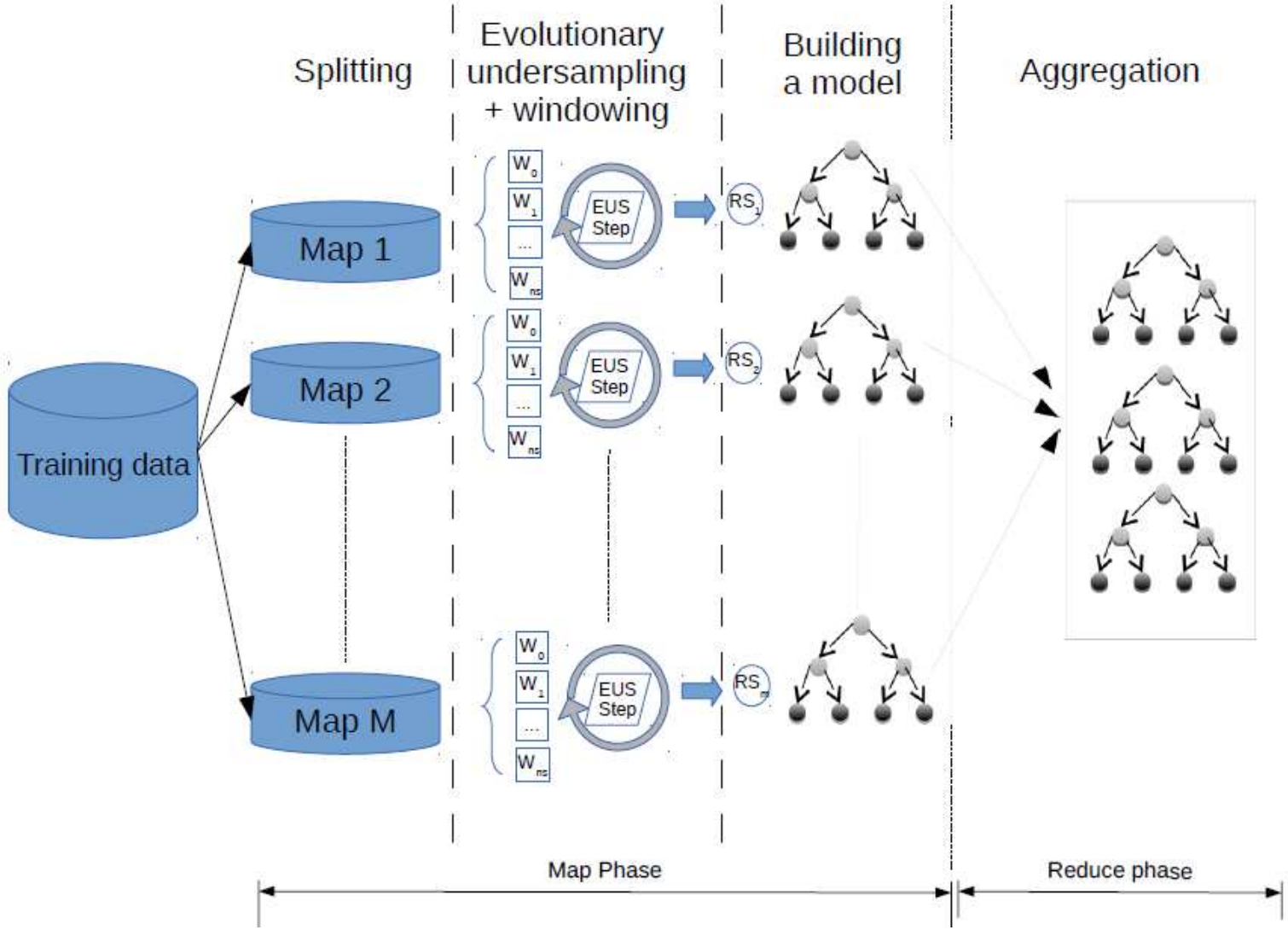
- Disjoint windows with equal class distribution may lead to information loss of the positive class.
- The minority class set will be always used to evaluate a chromosome.
- The majority class set is divided into several disjoint strata. The size of each subset will correspond to the number of minority class instances.
  - It means: Fixed number of strata.

Parallelizing EUS-BD with windowing: Properties

Properties:

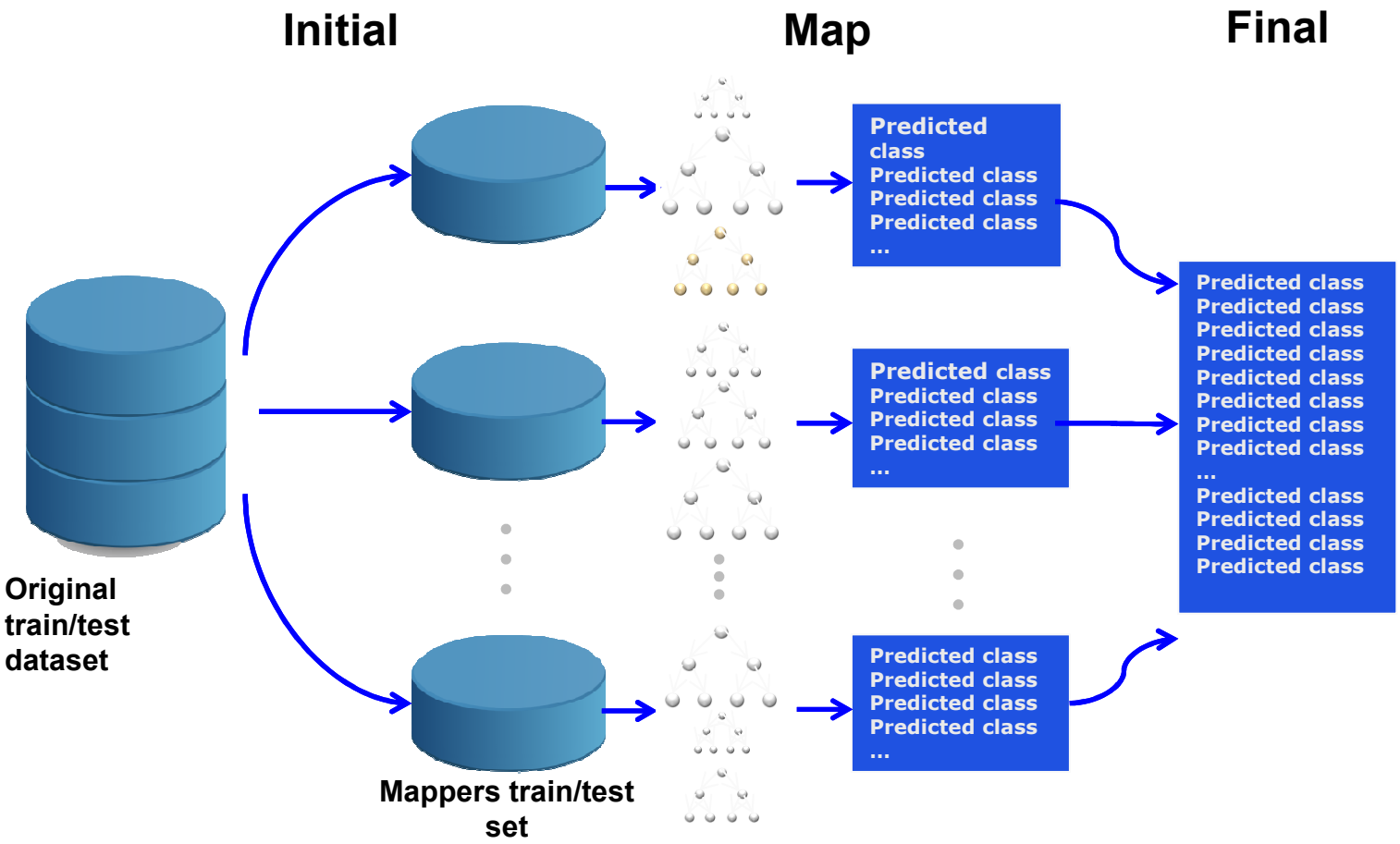
- Within this scheme, the algorithm disposes of the whole information although it is accessed in successive iterations.
- This model itself aims to improve the runtime requirements of EUS. But it does not deal with the memory consumption problem.
- This is why we use this strategy as a second level parallelization scheme after a previous distribution of the processing in a cluster of computing elements.

The EUS-BD scheme: learning phase



The EUS-BD scheme: testing phase

Estimating the class for a Big Dataset



Experimental Framework

- Different versions of the Kdd Cup 1999 data set with more than 4 million of instances, 3x5 fcv.
- Performance measures: AUC, g-mean, building time, and classification time.
- Different number of mappers: 128,256 and 512.

Data set	#negative	#positive	IR
kddcup DOS vs. normal	3883370	972781	3.99
kddcup DOS vs. PRB	3883370	41102	94.48
kddcup DOS vs. R2L	3883370	1126	3448.82
kddcup DOS vs. U2R	3883370	52	74680.25

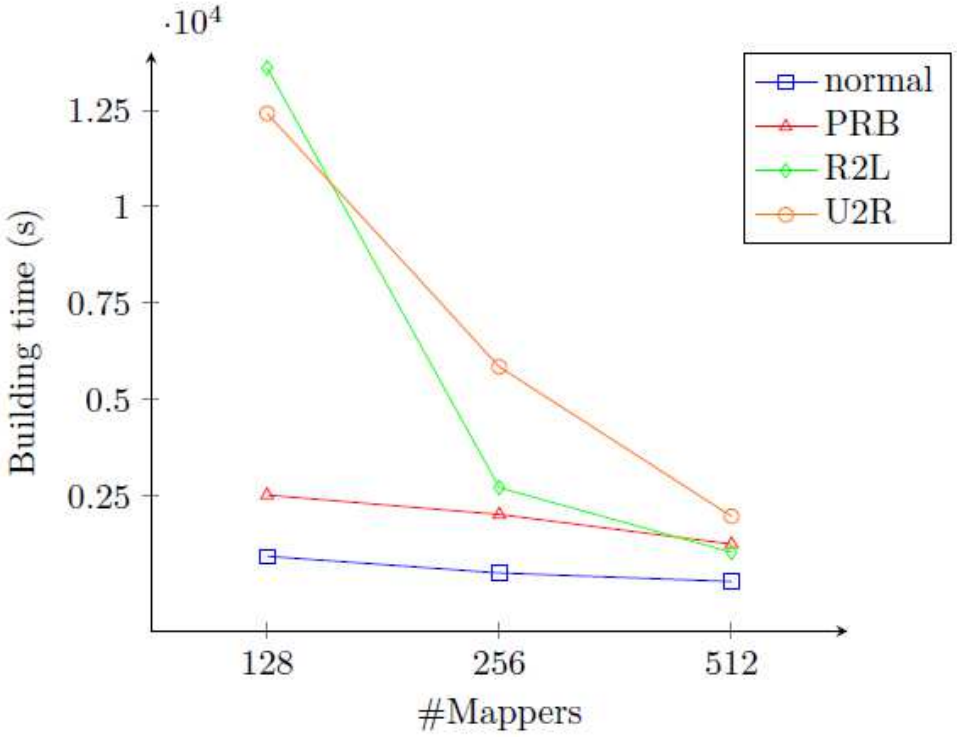
Results obtained without using the windowing mechanism

Data set	#Mappers	AUC	g-mean	Building Time	Classification Time
kddcup DOS vs. normal	128	0.99962397	0.99962395	942.2014	35.8988
	256	0.99924212	0.99924194	509.8122	38.4968
	512	0.99904700	0.99904674	287.2052	52.7572
kddcup DOS vs. PRB	128	0.99942829	0.99942822	2525.5332	33.4956
	256	0.99808006	0.99807901	2025.4140	41.9696
	512	0.99595677	0.99595641	1258.4924	48.9682
kddcup DOS vs. R2L	128	0.99817501	0.99817073	13595.0602	32.0616
	256	0.99817501	0.99817073	2720.0972	35.9038
	512	0.99817501	0.99817073	1045.7074	46.0528
kddcup DOS vs. U2R	128	0.97429702	0.97393535	12414.7948	31.2804
	256	0.98306267	0.98280252	5850.2702	35.2638
	512	0.98365571	0.98339482	1978.1212	46.0796

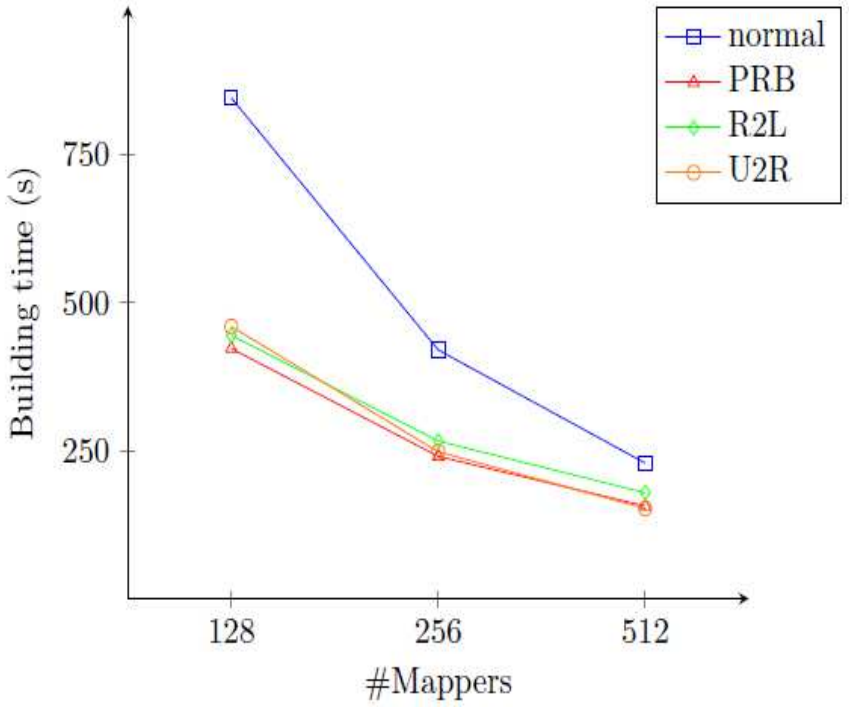
Results obtained using the windowing mechanism

Data set	Mappers	AUC	g-mean	Building Time	Classification Time
kddcup DOS vs. normal	128	0.99986345	0.99986345	845.5972	36.9734
	256	0.99979807	0.99979806	419.9624	31.3188
	512	0.99906136	0.99906110	228.9790	52.6386
kddcup DOS vs. PRB	128	0.99941760	0.99941754	422.4786	34.2640
	256	0.99778456	0.99778390	240.4662	36.7934
	512	0.99513122	0.99513099	156.4354	48.4240
kddcup DOS vs. R2L	128	0.99817501	0.99817073	444.7252	31.7255
	256	0.99817501	0.99817073	266.2424	36.1147
	512	0.99817501	0.99817073	178.8536	42.0057
kddcup DOS vs. U2R	128	0.98750466	0.98728379	459.6002	31.8436
	256	0.97617662	0.97583158	248.1038	35.5862
	512	0.97656950	0.97624880	152.3752	46.6194

Results: Building time comparison



Building time against the number of mappers, without using the windowing scheme.



Building time against the number of mappers, using the windowing scheme.

Final comments

- Good synergy between the windowing and MapReduce approaches.
- It enables EUS to be applied on data sets of almost arbitrary size.
- See my new paper #16712:

  I.Triguero et. al. Evolutionary Undersampling for Extremely Imbalanced Big Data Classification under Apache Spark.

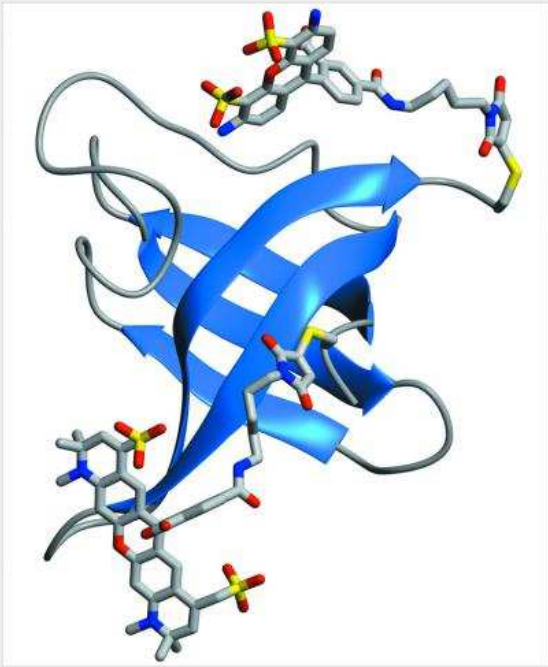
  SS CDCI08, Monday at 17:30, *Room: 202.*

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

ECBDL'14 Big Data Competition Vancouver, 2014

ECBDL'14 Big Data Competition 2014: Self-deployment track



Objective: Contact map prediction

Details:

- 32 million instances
- 631 attributes (539 real & 92 nominal values)
- 2 classes
- 98% of negative examples
- About 56.7GB of disk space

Evaluation:

True positive rate · True negative rate

TPR · TNR

http://cruncher.ncl.ac.uk/bdcomp/index.pl?action=data

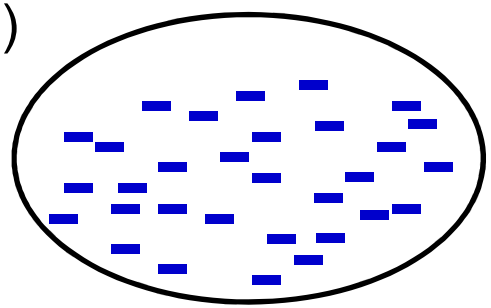
J. Bacardit et al, **Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features**, Bioinformatics 28 (19) (2012) 2441-2448

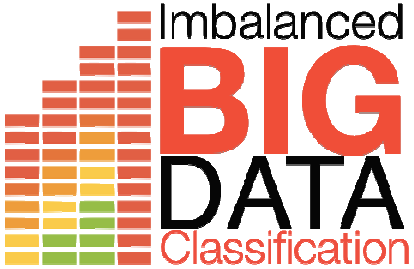
Evolutionary Computation for Big Data and Big Learning Workshop

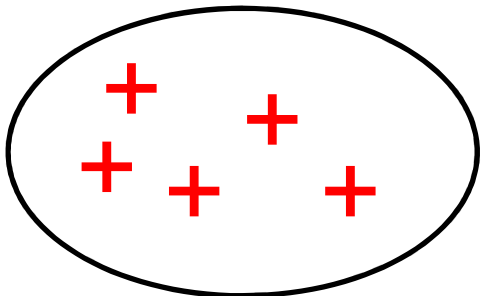
ECBDL'14 Big Data Competition 2014: Self-deployment track

The challenge:

- Very large size of the training set
  - Does not fit all together in memory.
- Even large for the test set (5.1GB, 2.9 million instances)
- Relatively high dimensional data.
- Low ratio (<2%) of true contacts. Imbalance rate: > 49
  - Imbalanced problem!
  - Imbalanced Big Data Classification

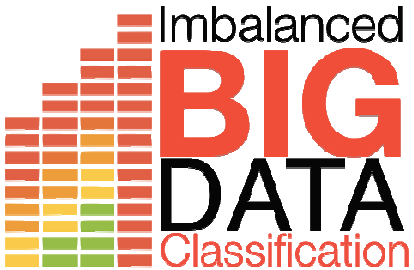






Imbalanced Big Data Classification

A MapReduce Approach



32 million instances, 98% of negative examples

Low ratio of true contacts (<2%). Imbalance rate: > 49. Imbalanced problem!

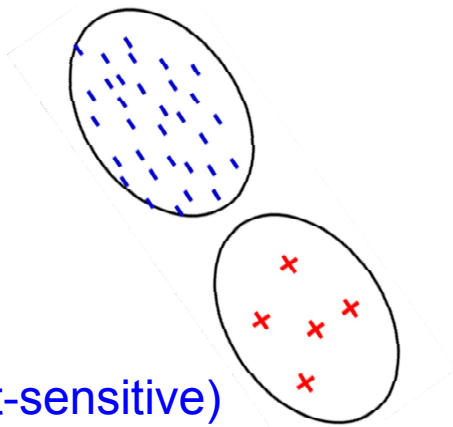
Previous study on extremely imbalanced big data:
S. Río, V. López, J.M. Benítez, F. Herrera, On the use of MapReduce for Imbalanced Big Data using Random Forest. *Information Sciences 285 (2014) 112-137.*

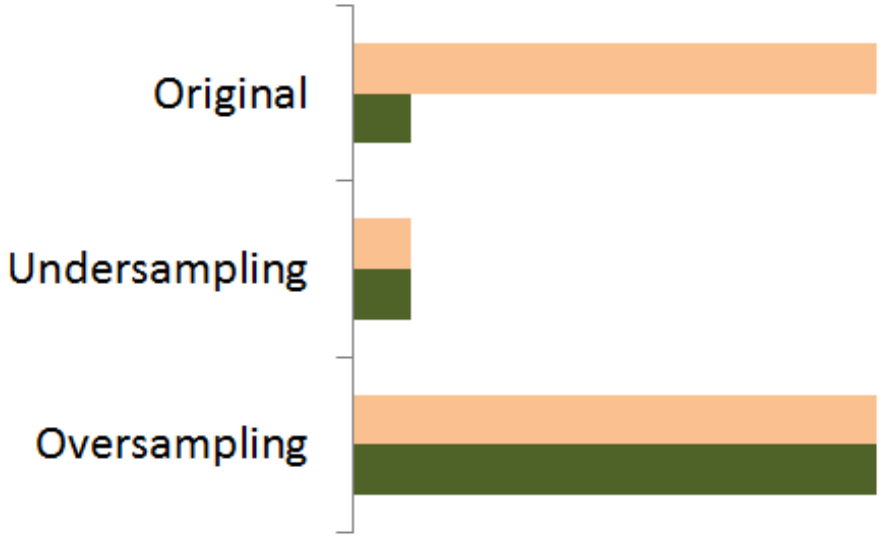
Over-Sampling
- Random
- Focused

Under-Sampling
- Random
- Focused

Cost Modifying (cost-sensitive)

Boosting/Bagging approaches (with preprocessing)





ECBDL'14 Big Data Competition

ECBDL'14 Big Data Competition 2014

Our approach:

1. Balance the original training data
   - ❑ Random Oversampling
   - ❑ (As first idea)

2. Learning a model.
   - ❑ Random Forest

ECBDL'14 Big Data Competition

We initially focused on

- Oversampling rate: 100%

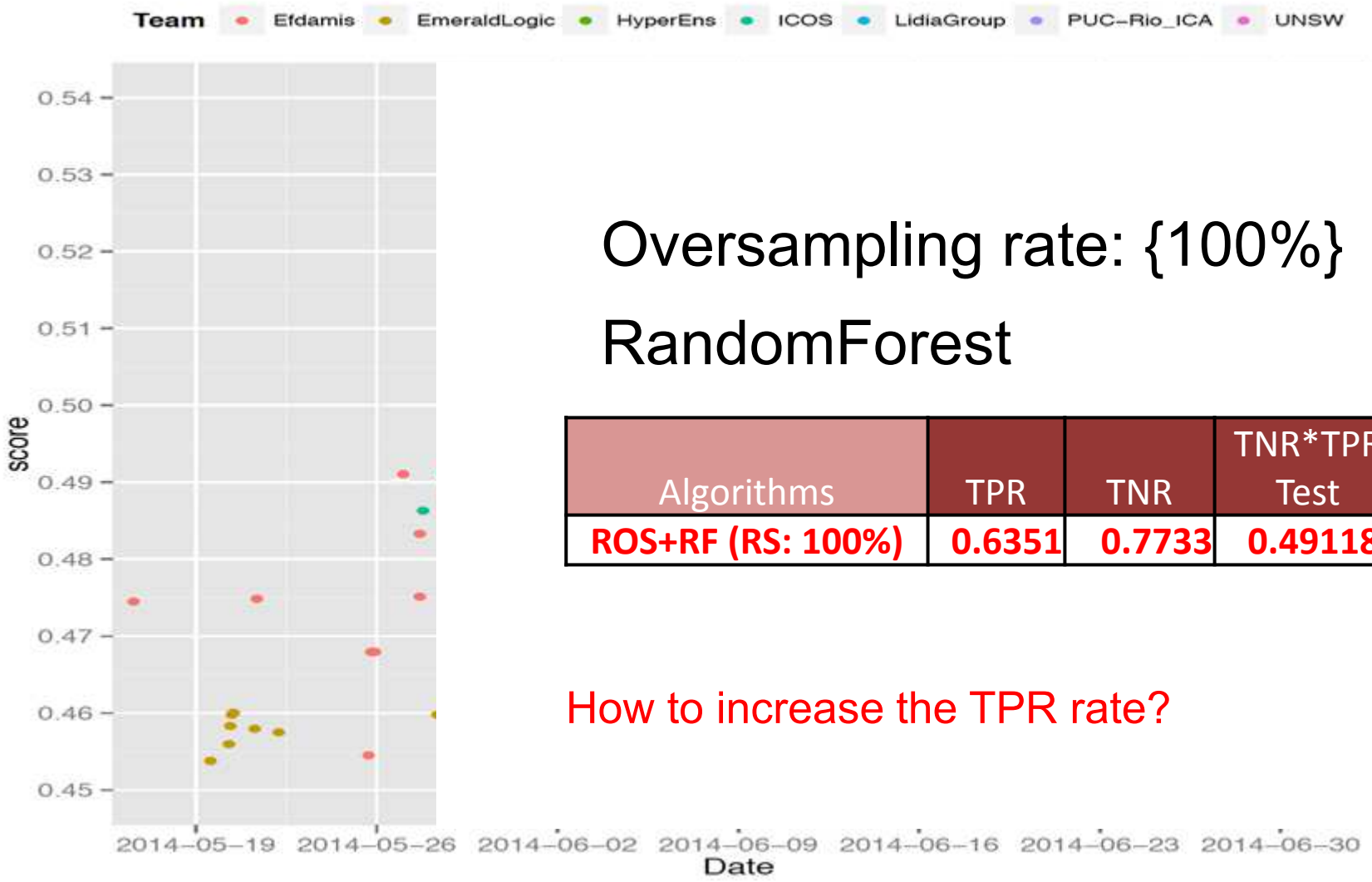
RandomForest:

- Number of used features: 10 (log n +1); Number of trees: 100
- Number of maps: {64, 190, 1024, 2048}

Nº mappers	TPR_tst	TNR_tst	TNR*TPR Test
64	0,601723	0,806269	0,485151

Very low TPR (relevant!)

ECBDL'14 Big Data Competition



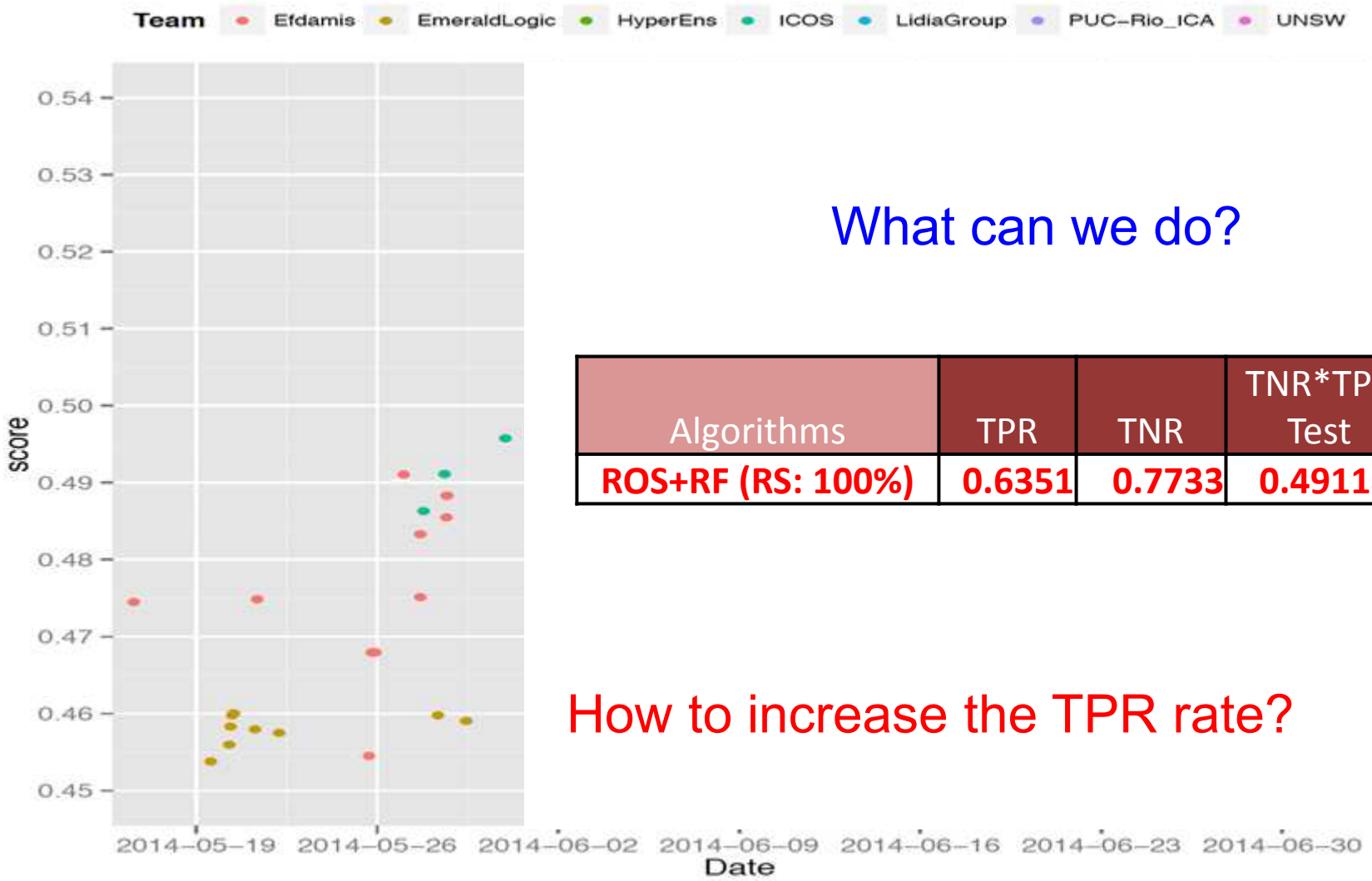
Oversampling rate: {100%}

RandomForest

Algorithms	TPR	TNR	TNR*TPR Test
ROS+RF (RS: 100%)	**0.6351**	**0.7733**	**0.491186**

How to increase the TPR rate?

ECBDL'14 Big Data Competition



What can we do?

Algorithms	TPR	TNR	TNR*TPR Test
ROS+RF (RS: 100%)	**0.6351**	**0.7733**	**0.491186**

How to increase the TPR rate?

ECBDL'14 Big Data Competition

ECBDL'14 Big Data Competition 2014

Our approach:

1. Balance the original training data
   - ❑ Random Oversampling **increasing the ROS percentage**
   - ❑ (As first idea, it was extended)

2. Learning a model.
   - ❑ Random Forest

ECBDL'14 Big Data Competition

How to increase the TPR rate?

Idea: To increase the ROS percentaje

- Oversampling rate: {100, 105, 110, 115, 130}

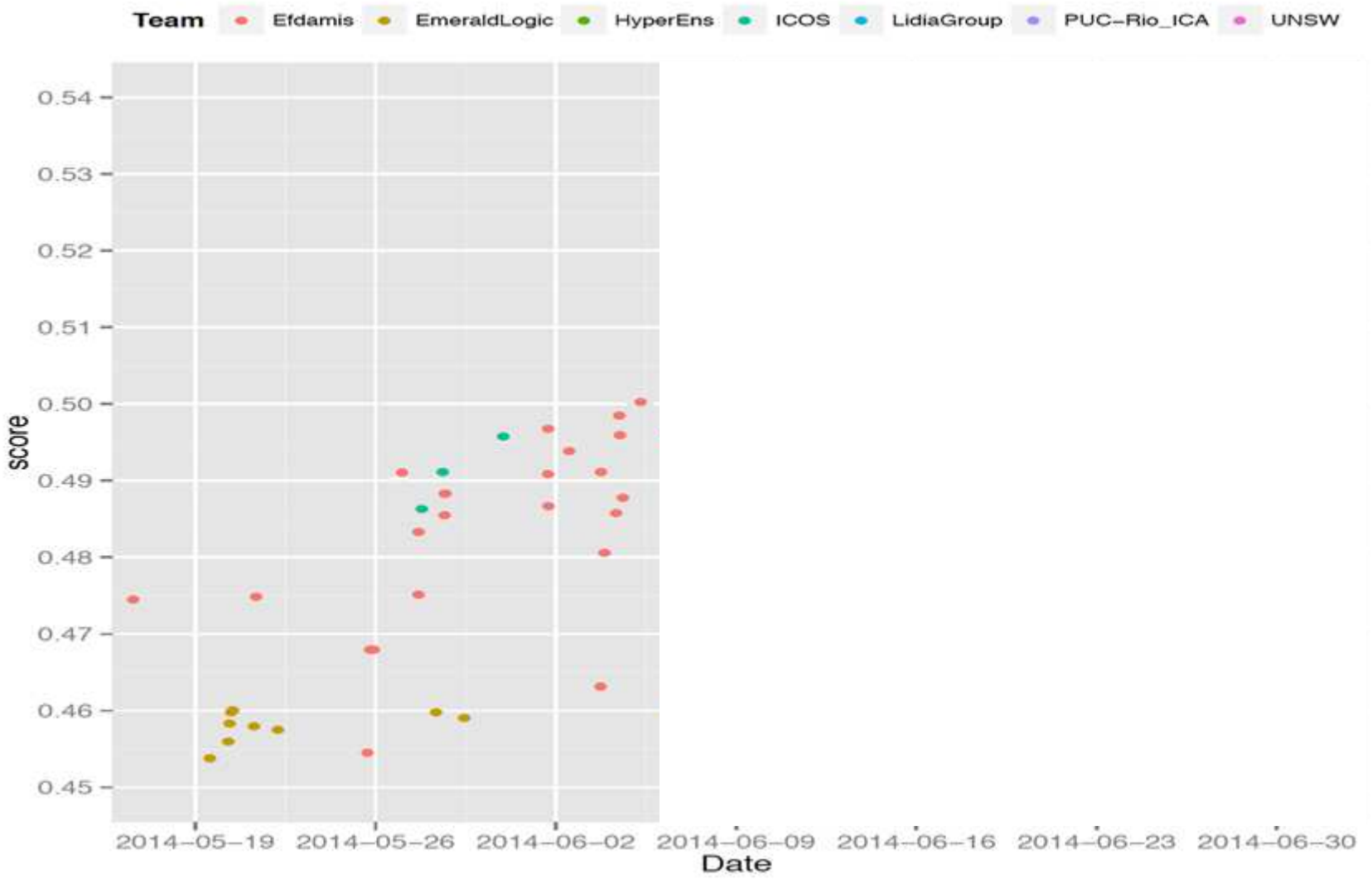
RandomForest:

- Number of used features:10; Number of trees: 100

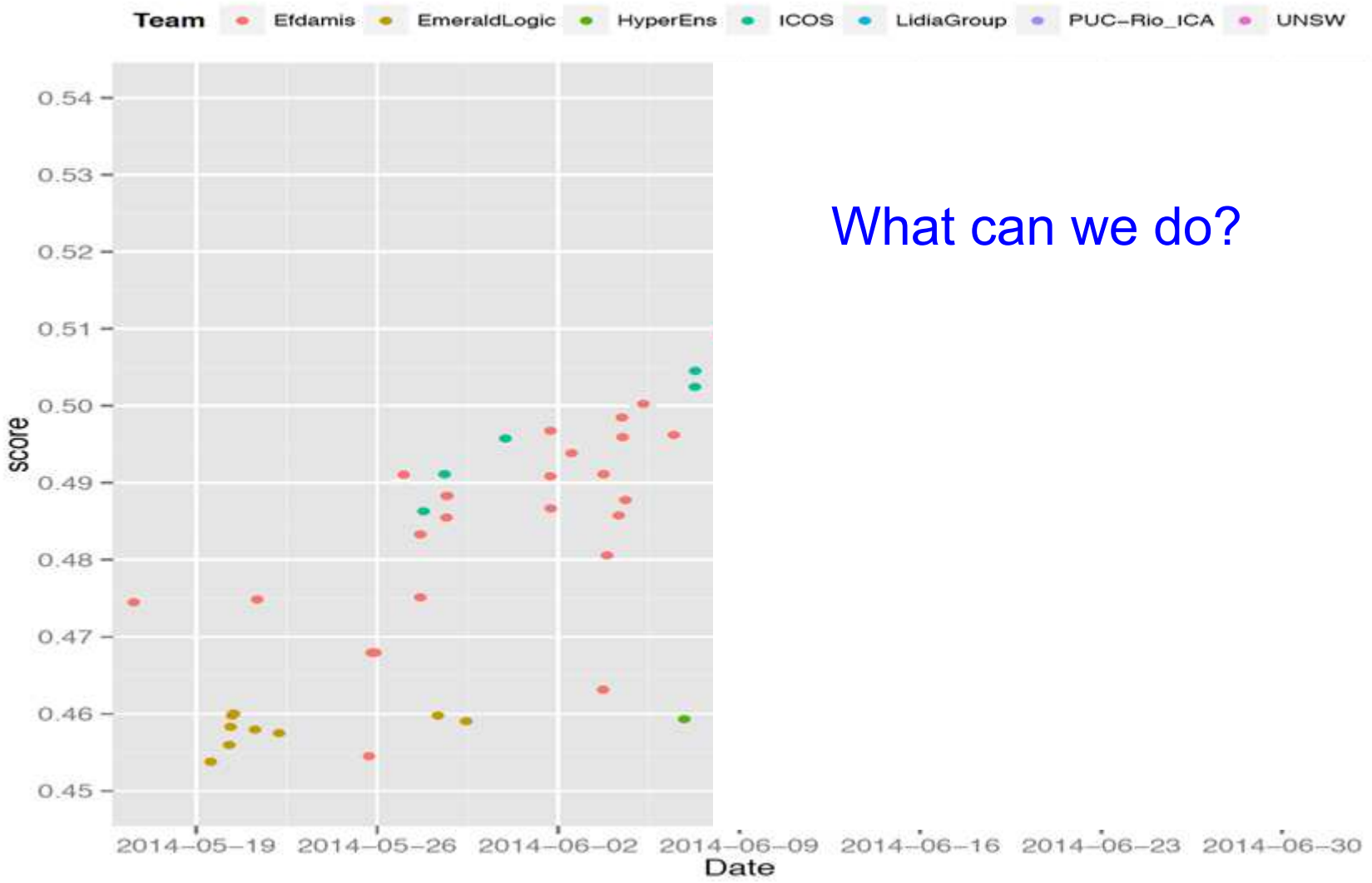
Algorithms	TPR	TNR	TNR*TPR Test
ROS+RF (RS: 100%)	**0.6351**	**0.7733**	**0.491186**
ROS+RF (RS: 105%)	0.6568	0.7555	0.496286
ROS+RF (RS: 110%)	0.6759	0.7337	0.495941
ROS+RF (RS: 115%)	**0.7041**	**0.7103**	**0.500175**
ROS+RF (RS: 130%)	0.7472	0.6609	0.493913

The higher ROS percentage, the higher TPR and the lower TNR

ECBDL'14 Big Data Competition



ECBDL'14 Big Data Competition

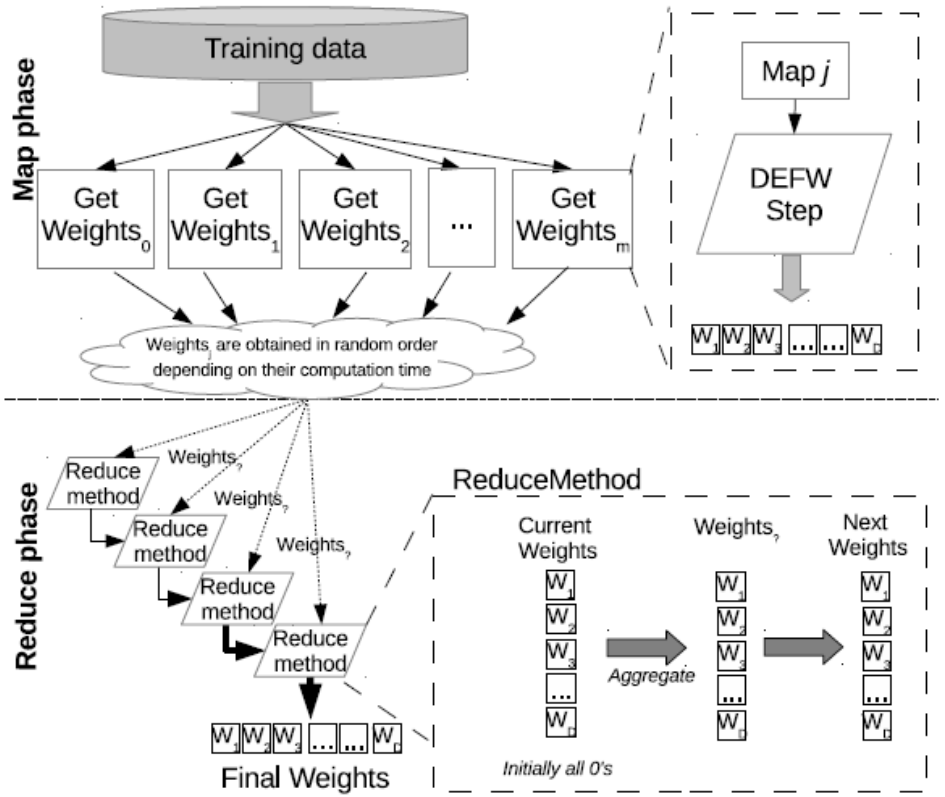


What can we do?

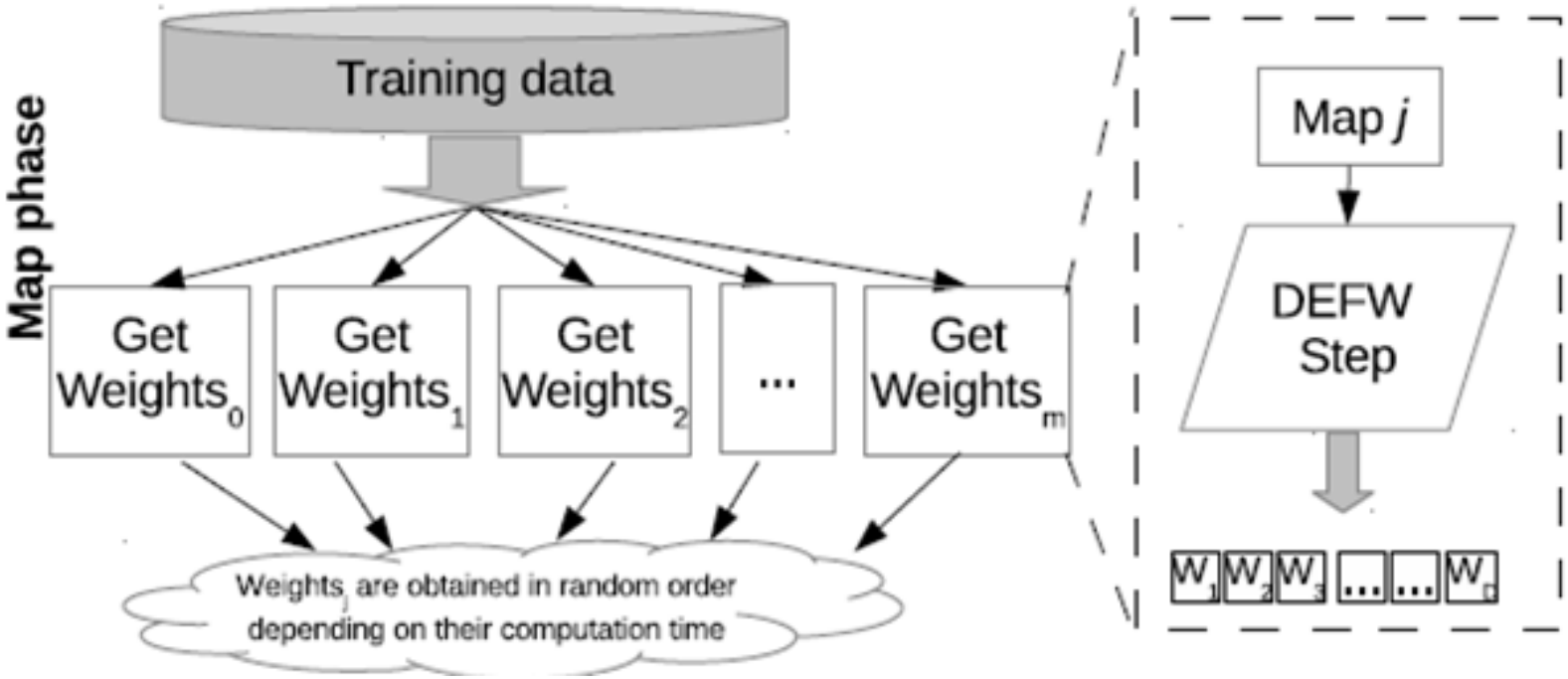
ECBDL'14 Big Data Competition

How to increase the performance?

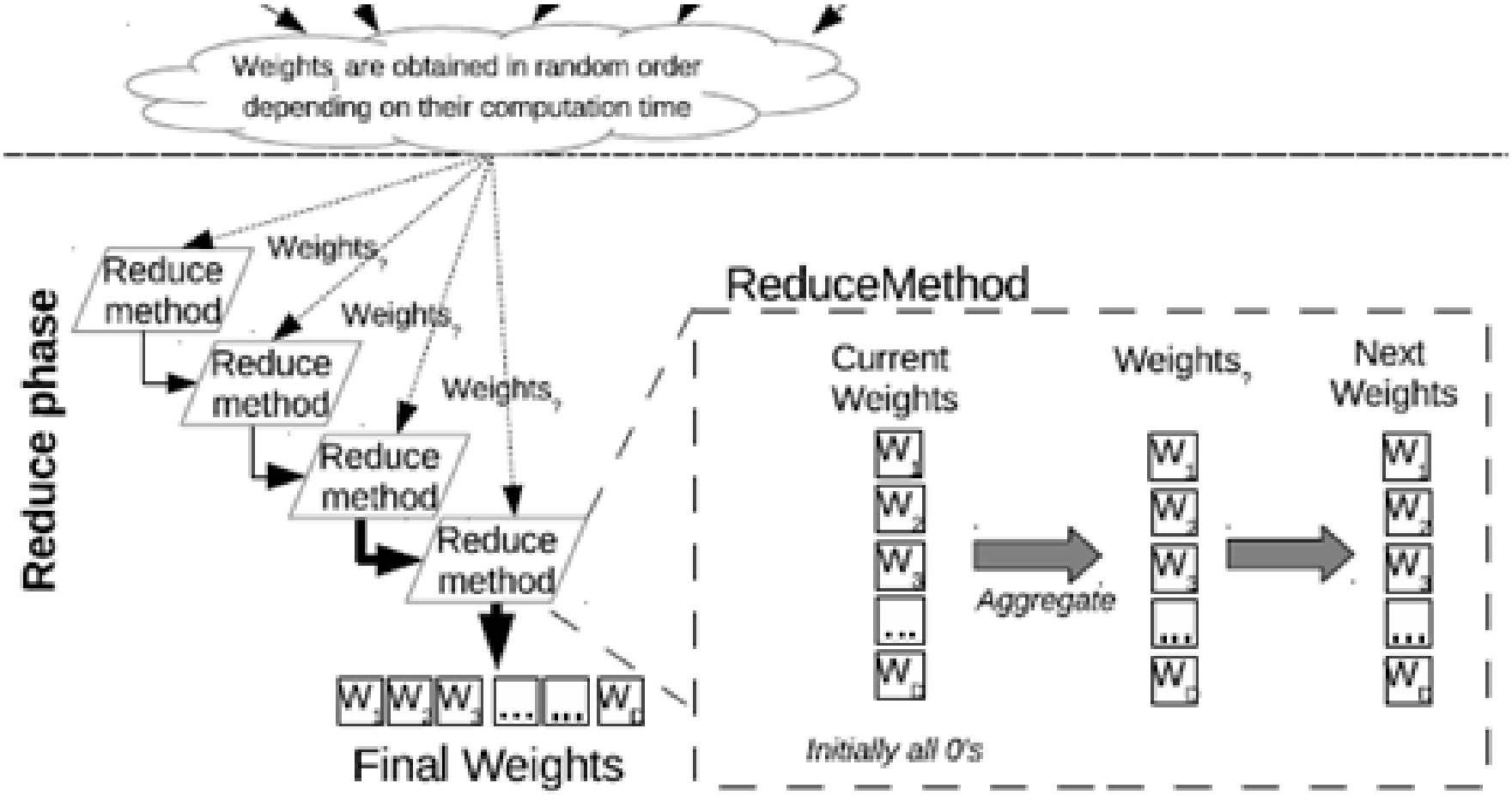
Third component: MapReduce Approach for Feature Weighting for getting a major performance over classes



ECBDL'14 Big Data Competition



ECBDL'14 Big Data Competition



ECBDL'14 Big Data Competition

ECBDL'14 Big Data Competition 2014

Our approach:

1. Balance the original training data
   - ❑ Random Oversampling **increasing the ROS percentage**
   - ❑ (As first idea, it was extended)

2. Learning a model.
   - ❑ Random Forest

3. Detect relevant features.
   1. Evolutionary Feature Weighting

Classifying test set.

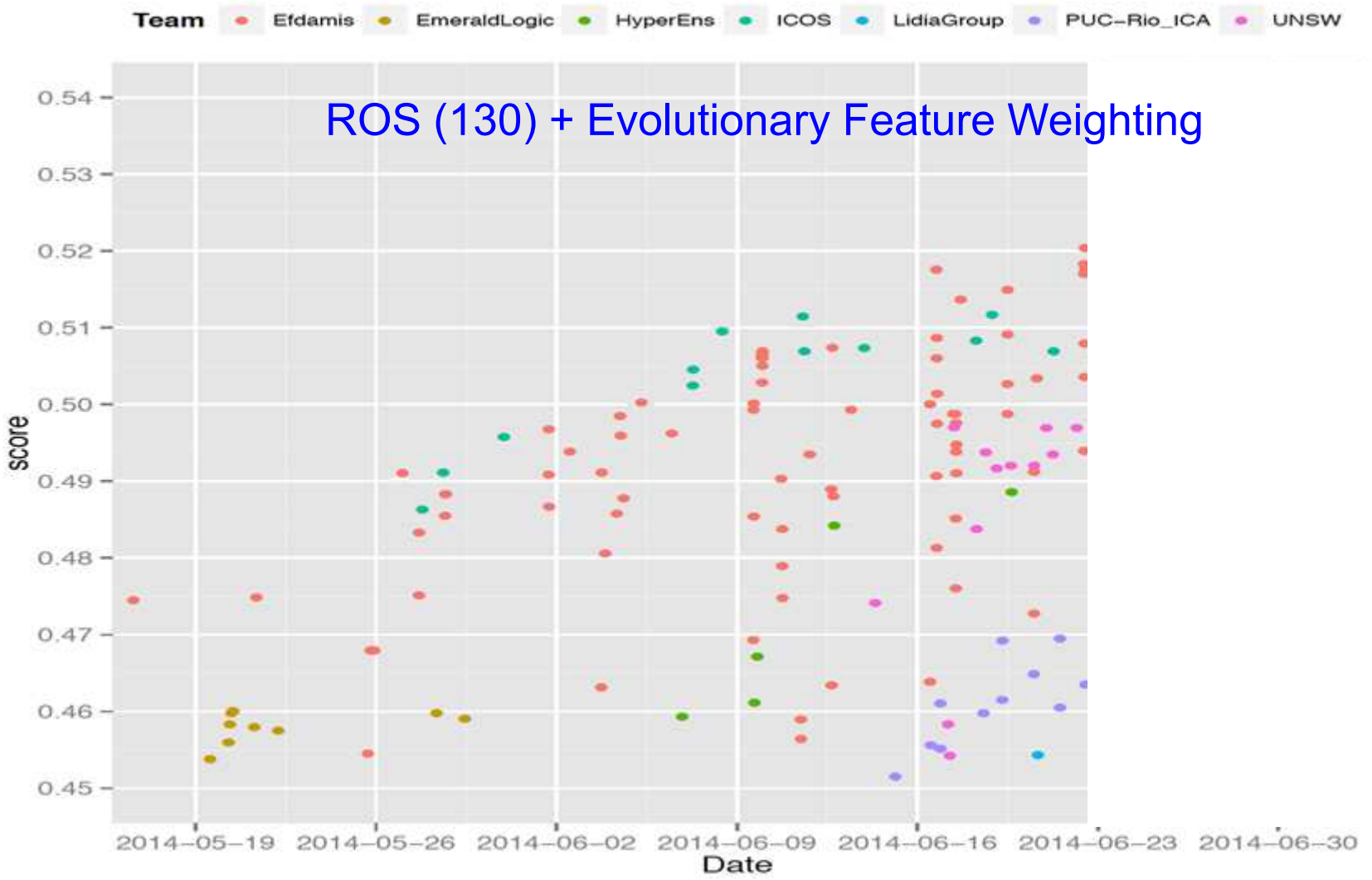
ECBDL'14 Big Data Competition

Evolutionary Feature Weighting.

It allows us to construct several subset of features (changing the threshold).

	64 mappers			
Algorithms	TNR*TPR Training	TPR	TNR	TNR*TPR Test
ROS+RF (130% - Feature Weighting 63)	0.726350	0.66949	0.775652	0.519292
ROS+RF (115% - Feature Weighting 63)	0.736596	0.652692	0.790822	0.516163
ROS+RF (100% - Feature Weighting 63)	0.752824	0.626190	0.811176	0.507950

ECBDL'14 Big Data Competition



ECBDL'14 Big Data Competition

We decided to investigate:

a) On the Random Forest: the influence of the Random Forest's parameters (internal features and number of trees)
b) Higher number of features (90) and ROS with 140%

	190 mappers			
Algorithms	**TNR*TPR Training**	**TPR**	**TNR**	**TNR*TPR Test**
ROS+ RF (130%+ FW 63+6f+100t)	0.604687	0.698152	0.742462	0.518351
ROS+ RF (130%+ FW 63+6f+200t)	0.632078	0.700064	0.745225	0.521705
ROS+ RF (140%+ FW 63+15f+200t)	0.627409	0.719678	0.728912	0.524582
ROS+ RF (140%+ FW 90+15f+200t)	0.635855	0.722639	0.726397	0.524923
ROS+ RF (140%+ FW 90+25f+200t)	**0.629273**	**0.721652**	**0.729740**	**0.526618**

Correct decisions with FW 90 and RF with 25f and 200 trees.
Good trade off between TPR and TNR

ECBDL'14 Big Data Competition

The less number of maps, the less TPR and the high TNR

	190 mappers			
Algorithms	**TNR*TPR Training**	**TPR**	**TNR**	**TNR*TPR Test**
ROS+ RF (140%+ FW 90+25f+200t)	**0.629273**	**0.721652**	**0.729740**	**0.526618**

64 mappers and we got 0.53

	64 mappers			
Algorithms	**TNR*TPR Training**	**TPR**	**TNR**	**TNR*TPR Test**
ROS+ RF (130%+ FW 90+25f+200t)	**0.736987**	**0.671279**	**0.783911**	**0.526223**
ROS+ RF (140%+ FW 90+25f+200t)	**0.717048**	**0.695109**	**0.763951**	**0.531029**

ROS 130 – 65 (140 – 68) replications of the minority instances

4 days to finish the competion:

Can we take decisions for improving the model?

ECBDL'14 Big Data Competition

Last decision: We investigated to increase ROS until 180% with 64 mappers

	64 mappers			
Algorithms	TNR*TPR Training	TPR	TNR	TNR*TPR Test
ROS+ RF (130%+ FW 90+25f+200t)	0.736987	0.671279	0.783911	0.526223
ROS+ RF (140%+ FW 90+25f+200t)	0.717048	0.695109	0.763951	0.531029
ROS+ RF (150%+ FW 90+25f+200t)	0.706934	0.705882	0.753625	0.531971
ROS+ RF (160%+ FW 90+25f+200t)	0,698769	0.718692	0.741976	0.533252
ROS+ RF (170%+ FW 90+25f+200t)	0.682910	0.730432	0.730183	0.533349
ROS+ RF (180%+ FW 90+25f+200t)	0,678986	0.737381	0.722583	0.532819

To increase ROS and reduce the mappers number lead us to get a trade-off with good results

ROS 170 – 85 replications of the minority instances

ECBDL'14 Big Data Competition

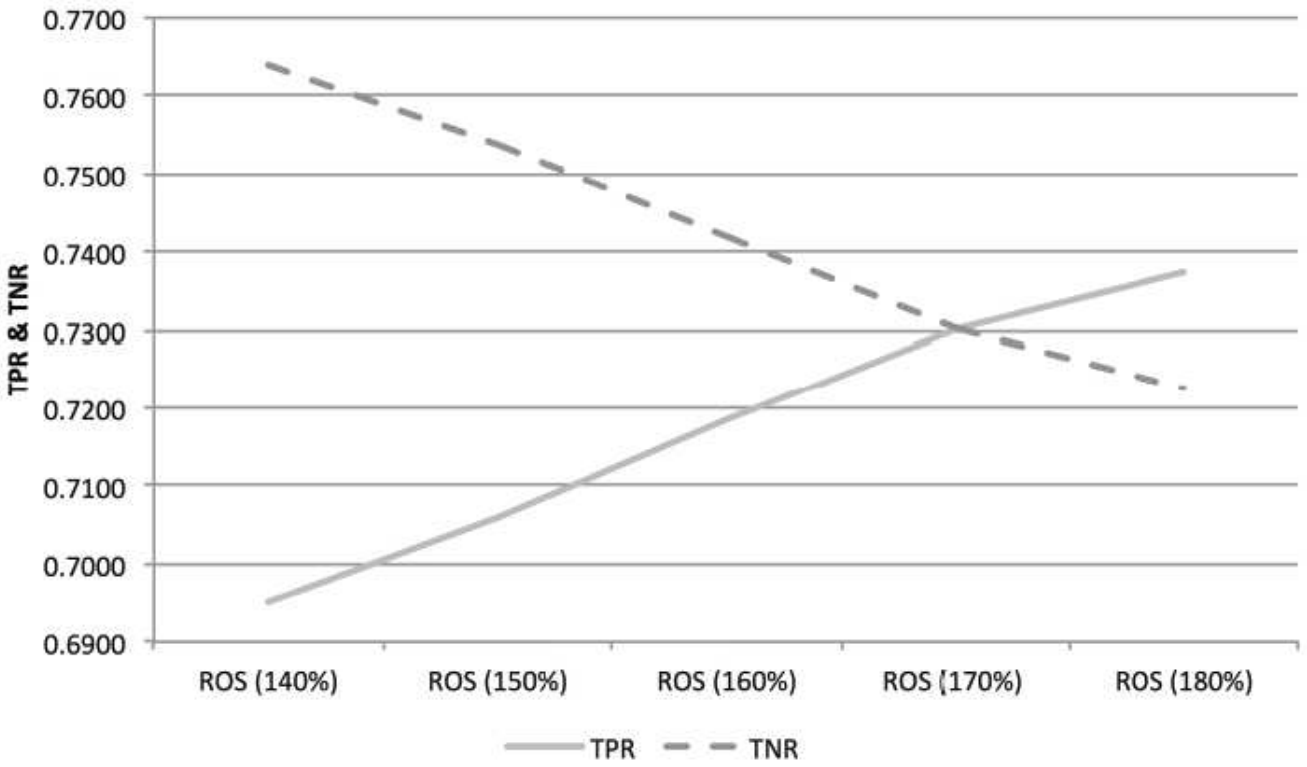


Figure 8: TPR vs. TNR varying the ROS percentage

ROS 170 – 85 replications of the minority instances
Experiments with 64 maps

ECBDL'14 Big Data Competition

Evolutionary Computation for Big Data and Big Learning Workshop

Results of the competition: Contact map prediction

Team Name	TPR	TNR	Acc	TPR · TNR
Efdamis	**0.730432**	**0.730183**	**0.730188**	**0.533349**
ICOS	**0.703210**	**0.730155**	**0.729703**	**0.513452**
UNSW	**0.699159**	**0.727631**	**0.727153**	**0.508730**
HyperEns	0.640027	0.763378	0.761308	0.488583
PUC-Rio_ICA	0.657092	0.714599	0.713634	0.469558

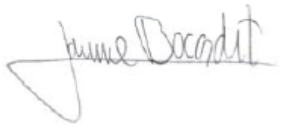
EFDAMIS team ranked first in the ECBDL'14 big data competition

http://cruncher.ncl.ac.uk/bdcomp/index.pl?action=ranking

ECBDL'14 Big Data Competition

ECBDL'14: Evolutionary Computation for Big Data and Big Learning Workshop
July 13th, 2014
GECCO-2014, Vancouver, Canada

This is to certify that team EFDAMIS, formed by Isaac Triguero, Sara del Río, Victoria López, José Manuel Benítez and Francisco Herrera, ranked **first** in the ECBDL'14 big data competition



Jaume Bacardit, organizer
ECBDL'14 big data competition

ECBDL'14 Big Data Competition

Final comments

Team Name	Learning strategy	Computational Infrastructure
Efdamis	Oversampling+EFW+Random Forest	MapReduce
ICOS	Oversampling+Ensemble of Rule sets	Batch HPC
UNSW	Ensemble of Deep Learning classifiers	Parallel HPC
HyperEns	SVM	Parallel HPC
PUC-Rio_ICA	Linear GP	GPUs
EmeraldLogic	~Linear GP	GPUs
LidiaGroup	1-layer NN	Spark

ECBDL'14 Big Data Competition

At the beginning ROS+RF (RS: 100%)

Nº mappers	TPR_tst	TNR_tst	TNR*TPR Test
64	0,601723	0,806269	0,485151

At the end

	64 mappers			
Algorithms	TNR*TPR Training	TPR	TNR	TNR*TPR Test
ROS+ RF (160%+ FW 90+25f+200t)	0,698769	0.718692	0.741976	0.533252
ROS+ RF (170%+ FW 90+25f+200t)	0.682910	0.730432	0.730183	0.533349
ROS+ RF (180%+ FW 90+25f+200t)	0,678986	0.737381	0.722583	0.532819

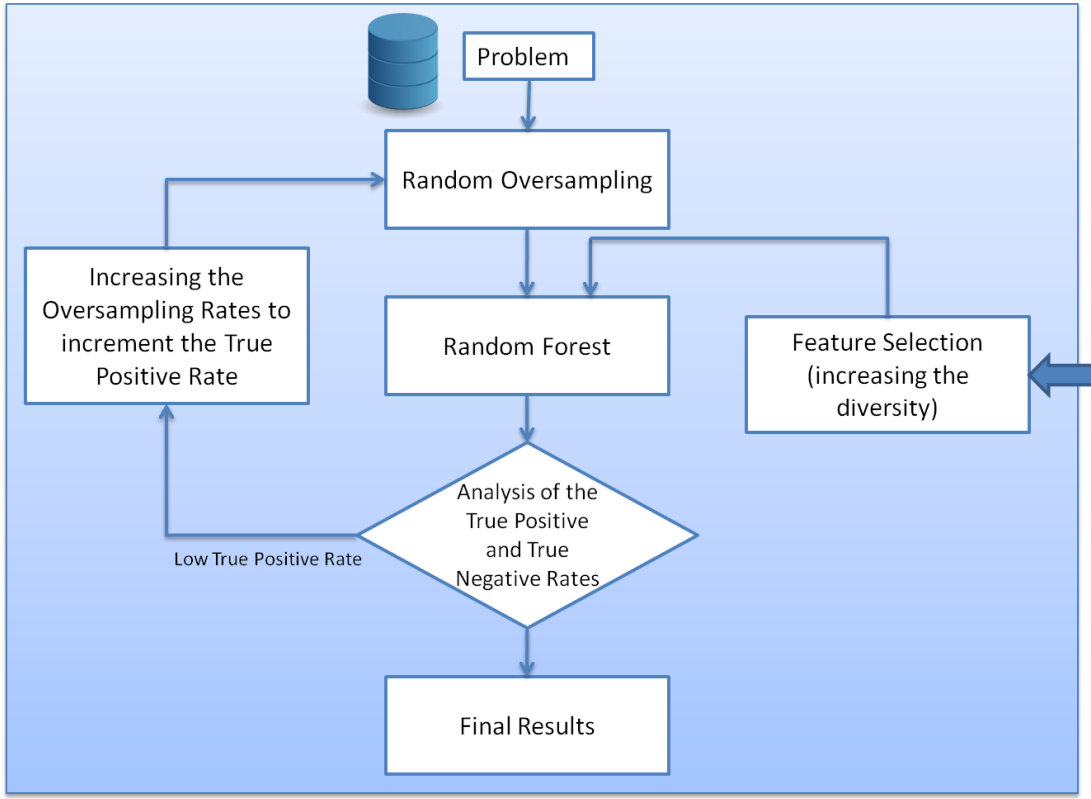
BIG DATA Preprocessing

Quality models based on quality data!

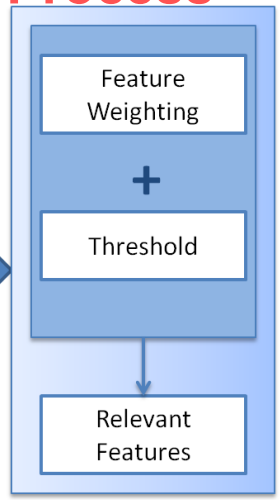
ECBDL'14 Big Data Competition

Our algorithm: ROSEFW-RF

Iterative MapReduce Process



MapReduce Process



I. Triguero, S. del Río, V. López, J. Bacardit, J.M. Benítez, F. Herrera. ROSEFW-RF: The winner algorithm for the ECBDL'14 Big Data Competition: An extremely imbalanced big data bioinformatics problem. Knowledge-Based Systems, Volume 87, October 2015, Pages 69–79

https://github.com/triguero/ROSEFW-RF

Outline

- A gentle introduction to Big Data
- Big data tecnologies
- Big Data analytics
- Fuzzy-based models for Big Data Learning
- Evolutionary models for Big Data Preprocessing
- A case of study: the ECBDL'14 competition
- Final Comments

Final Comments

Data Mining, Machine learning and data preprocessing: Huge collection of algorithms

Big Data Analytics

Big Data: A small subset of algorithms

Big Data Preprocessing:
A few methods for preprocessing in Big Data analytics.

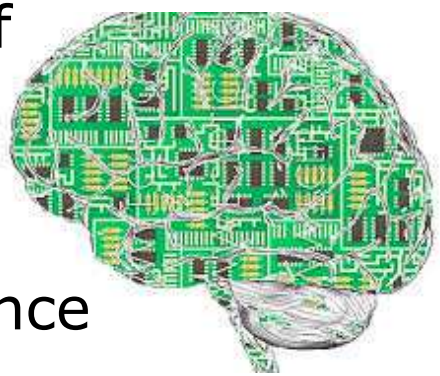
Final Comments

Where we are going: 3 Big Data stages

http://www.kdnuggets.com/2013/12/3-stages-big-data.html
By Gregory Piatetsky, Dec 8, 2013.

Big Data 3.0: Intelligent
Google Now, Watson (IBM) ...

Big Data 3.0 would be a combination of data, with huge knowledge bases and a very large collection of algorithms, perhaps reaching the level of true Artificial Intelligence (Singularity?).



Big Data 1.0:
Transactional

Big Data 2.0:
Networked

Big Data 3.0:
Intelligent

Final Comments

Computational Intelligence Approaches for Big Data

Francisco Herrera

Soft Computing and
Information Intelligent Systems (SCI²S)
University of Granada, Spain
Email: herrera@decsai.ugr.es
http://sci2s.ugr.es

Isaac Triguero

School of Computer Science
University of Nottingham
United Kingdom
Email: Isaac.Triguero@nottingham.ac.uk

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Big Data at SCI²S - UGR

http://sci2s.ugr.es/BigData

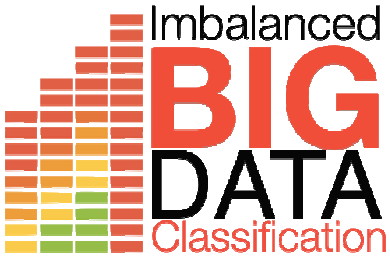
SCI²S website

Home » Thematic Sites » Big Data: Algorithms for Data Preprocessing, Computational Intelligence, and Imbalanced Classes

Big Data: Algorithms for Data Preprocessing, Computational Intelligence, and Imbalanced Classes

The web is organized according to the following summary:

1. Introduction to Big Data
2. Big Data Technologies: Hadoop ecosystem and Spark
3. Big Data preprocessing
4. Imbalanced Big Data classification
5. Big Data classification with fuzzy models
6. Big Data Applications
7. Dataset Repository
8. Literature review: surveys and overviews
9. Keynote slides
10. Links of interest



This **Website** contains SCI²S research material on algorithms for data preprocessing, computational intelligence and classification with imbalanced datasets in the scenario of Big Data. All information shown here is related to the following SCI²S review and papers: