Data Mining Methods for Big Data Preprocessing

Francisco Herrera

Research Group on Soft Computing and
Information Intelligent Systems (SCI^{2}S)
http://sci2s.ugr.es
Dept. of Computer Science and A.I.
University of Granada, Spain

Email: herrera@decsai.ugr.es

Objectives

- To understand the different problems to solve in the processes of data preprocessing.
- To know the problems related to clean data and to mitigate imperfect data, together with some techniques to solve them.
- To know the data reduction techniques and the necessity of their application.
- To know the problems to apply data preprocessing techniques for big data analytics.
- To know the current big data preprocessing proposals.

Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big Data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big Data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Big Data

EL MUNDO.es
Líder mundial en español | Miércoles 04/09/2013. Actualizado 16:27h.

Alex ' Sandy' Pentland
Media Lab Massachusetts Institute of Technology (MIT)

“It is the decade of data, hence come the revolution”

Alex 'Sandy' Pentland, durante su ponencia. | M. Sáinz

Ben Chams - Fotolia

Big Data

Data is at the centre of the knowledge economy and society

Our world revolves around the data

Ben Chams - Fotolia

What is Big Data?

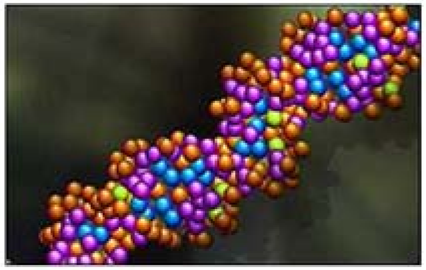
Our world revolves around the data

- 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  - Scanned books, historical documents, social interactions data, …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, …

What is Big Data? **3 Vs of Big Data**

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

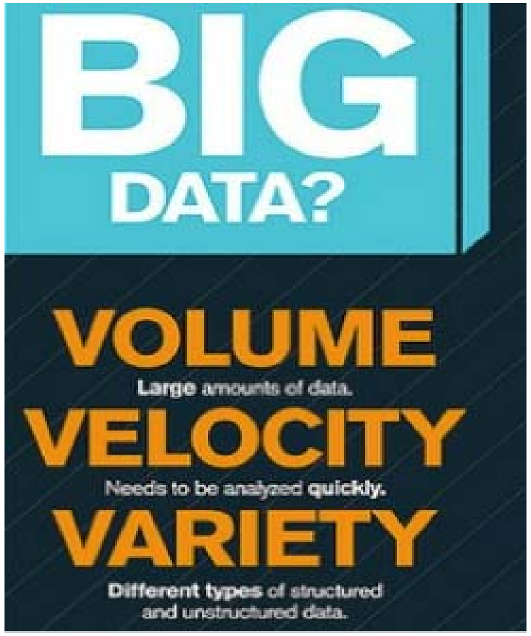
- 47.5 billion transactions in 2005 worldwide
- 115 Terabytes of data transmitted to VisaNet data processing center in 2004

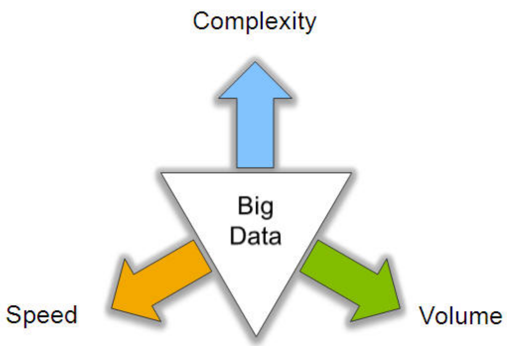
What is Big Data? **3 Vs of Big Data**

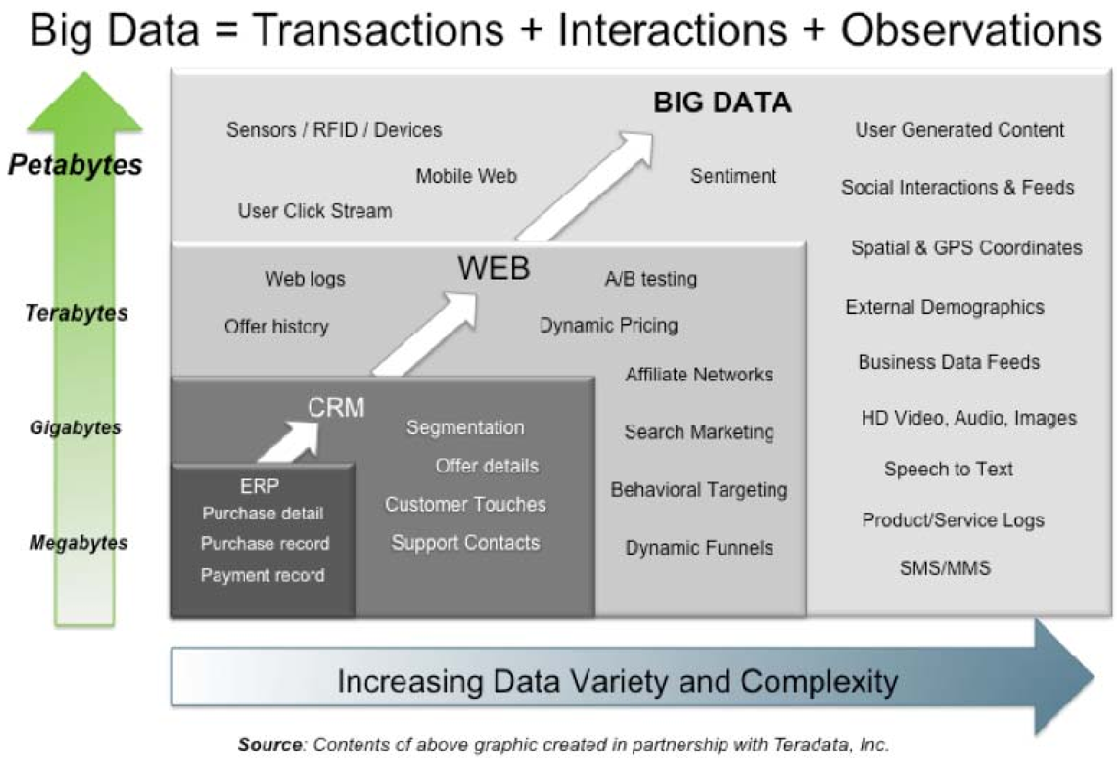
ePROMOTIONS

What is Big Data? **3 Vs of Big Data**

What is Big Data? **3 Vs of Big Data**

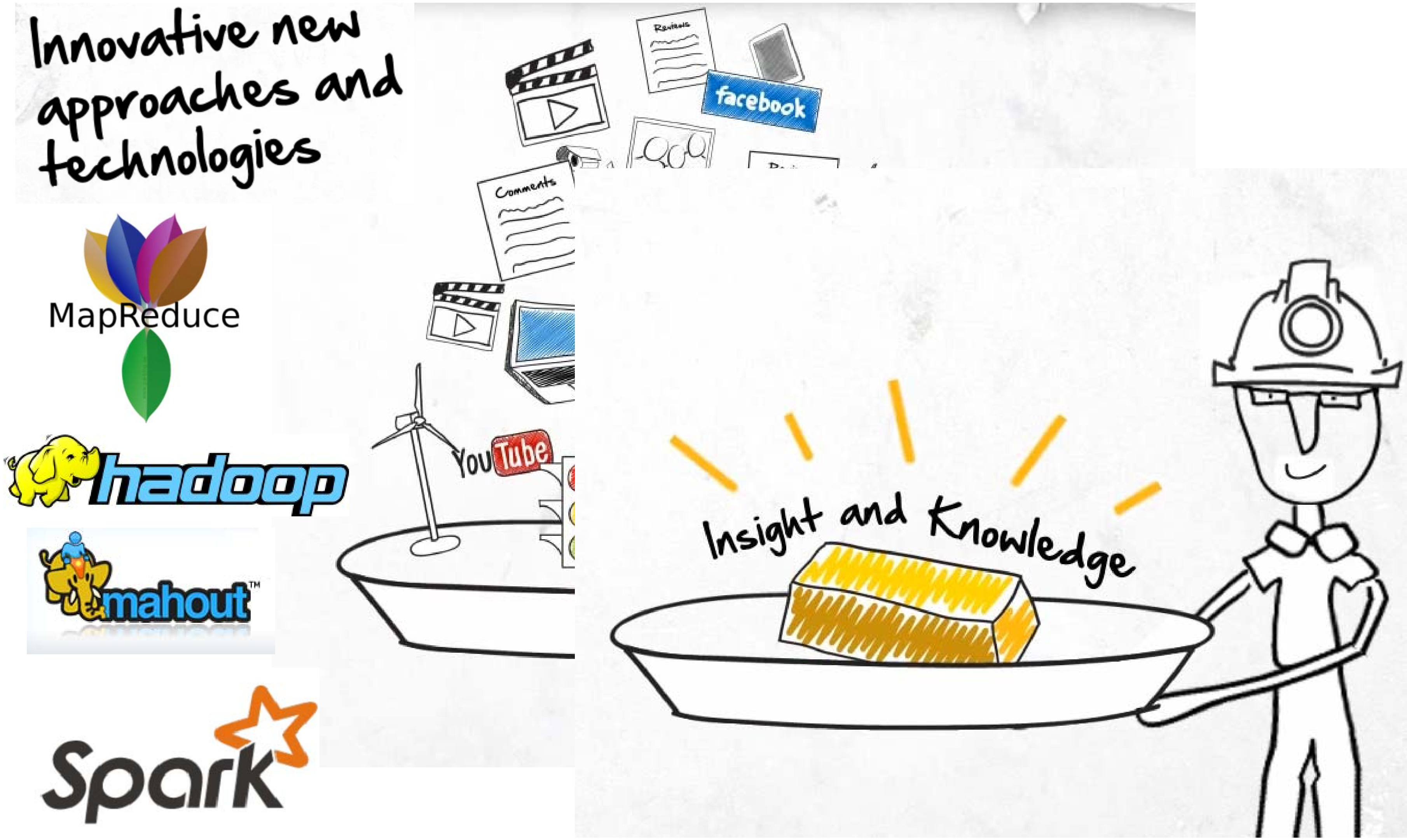






What is Big Data?

4 V's --> Value

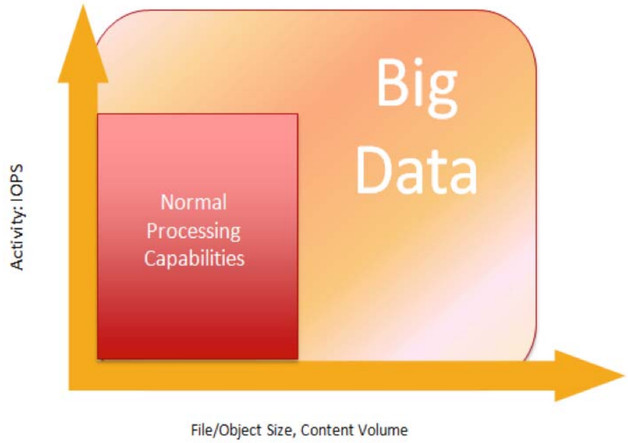


What is Big Data?

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…



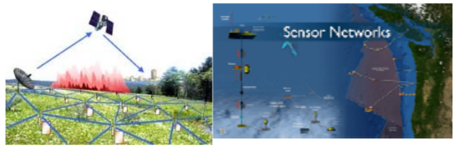
What is Big Data?

Who's Generating Big Data?

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)

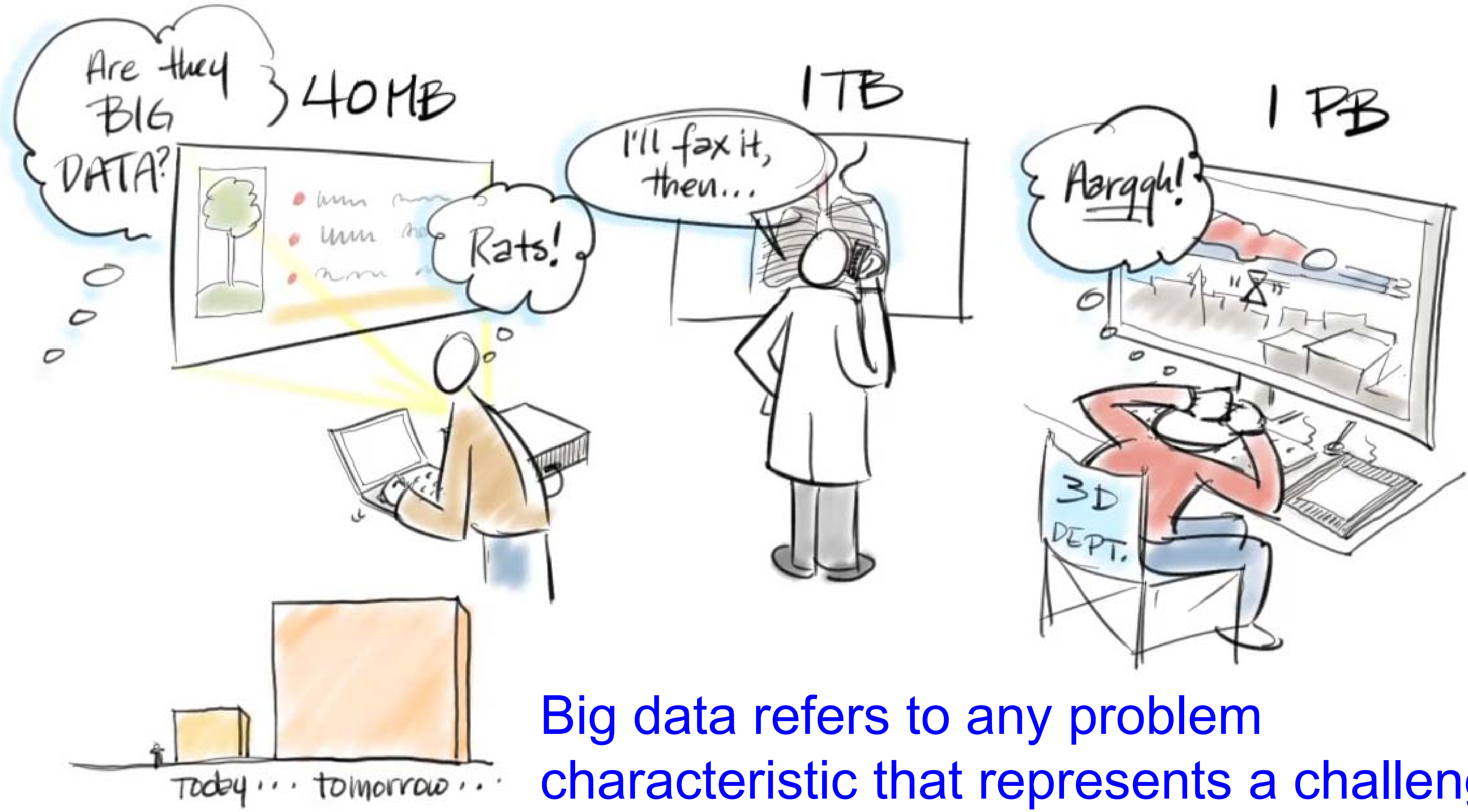


Sensor technology and networks
(measuring all kinds of data)

Transactions

The progress and innovation is no longer hindered by the ability to collect data but, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

What is Big Data? (in short)



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

What is Big Data? Example

ECBDL'14 Big Data Competition 2014
(GEGGO 2014, Vancouver)

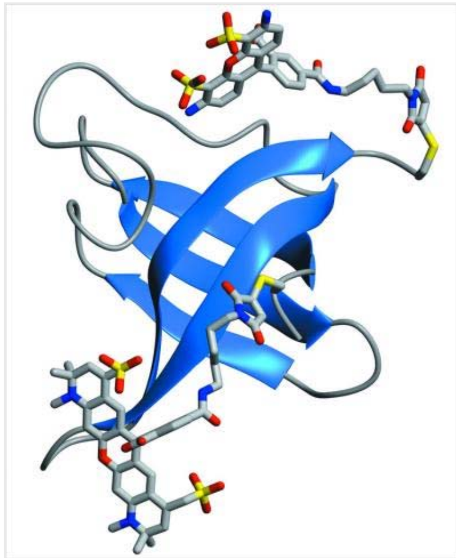
Objective: Contact map prediction

Details:

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

Evaluation:

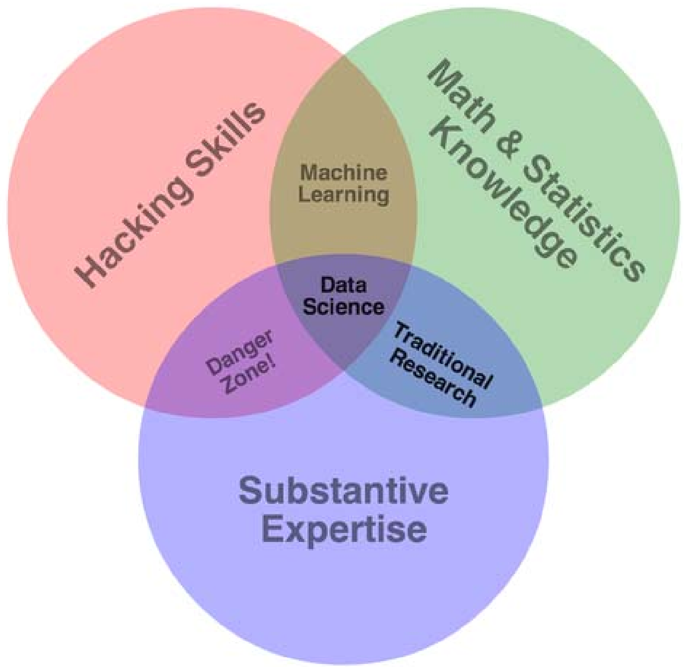
True positive rate · True negative rate
TPR · TNR



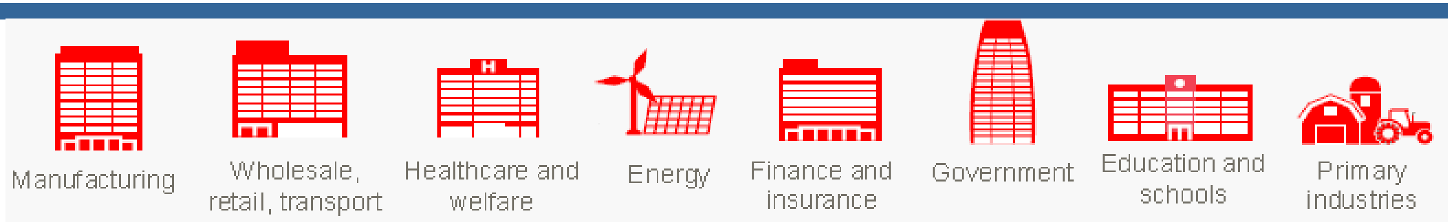
Big Data Science

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

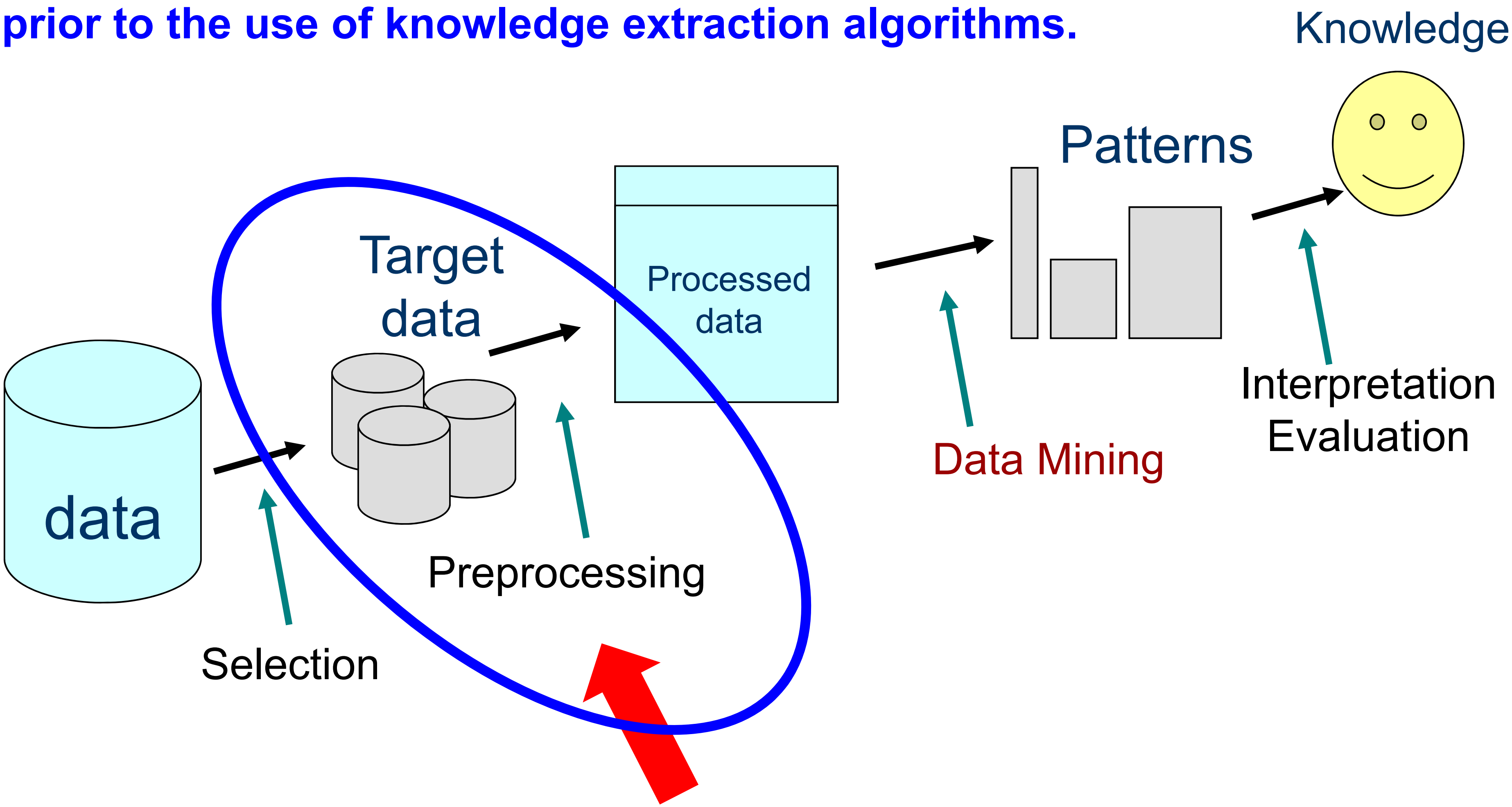
- Explore data
- Represent data
- Link data
- Learn from data
- Deliver insight
- …

Data Analytics

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

Data Preprocessing

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



Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Why Big Data?

- 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)

A single machine can not manage large volumes of data efficiently

Why Big Data? MapReduce

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

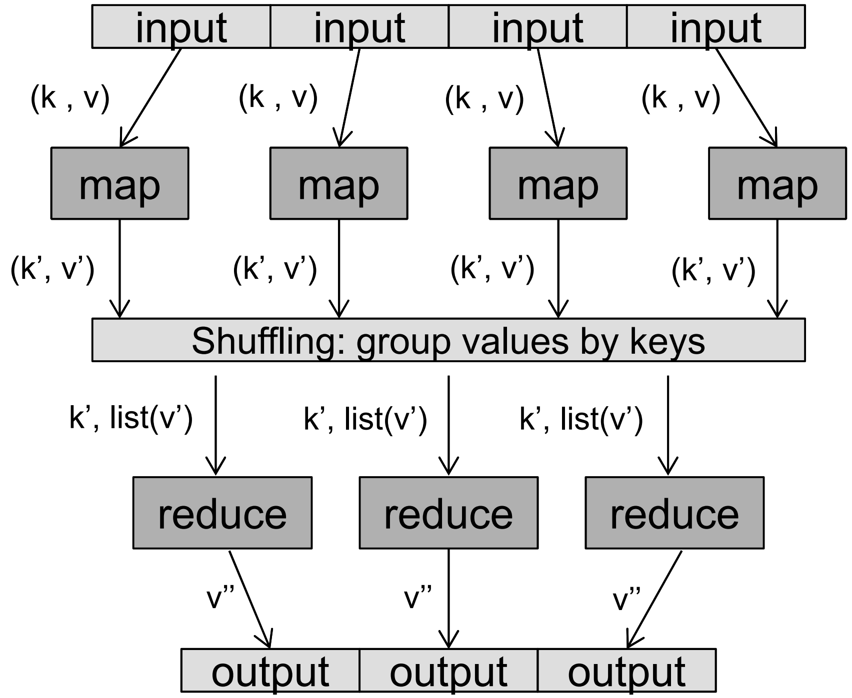
➔ Divide-And-Conquer (i.e., data partitioning)

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

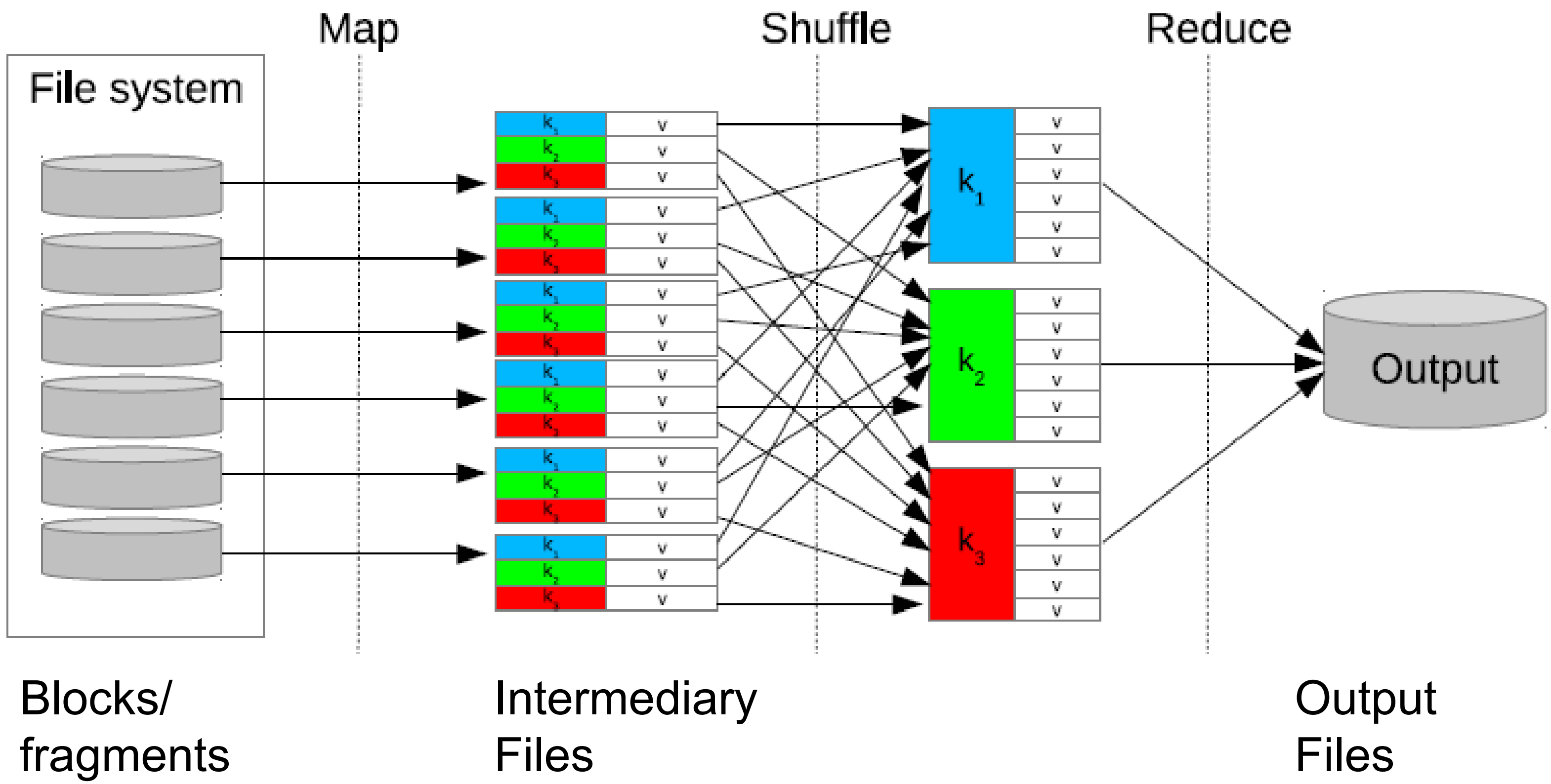


map (k, v) → list (k', v')
reduce (k', list(v')) → v''

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

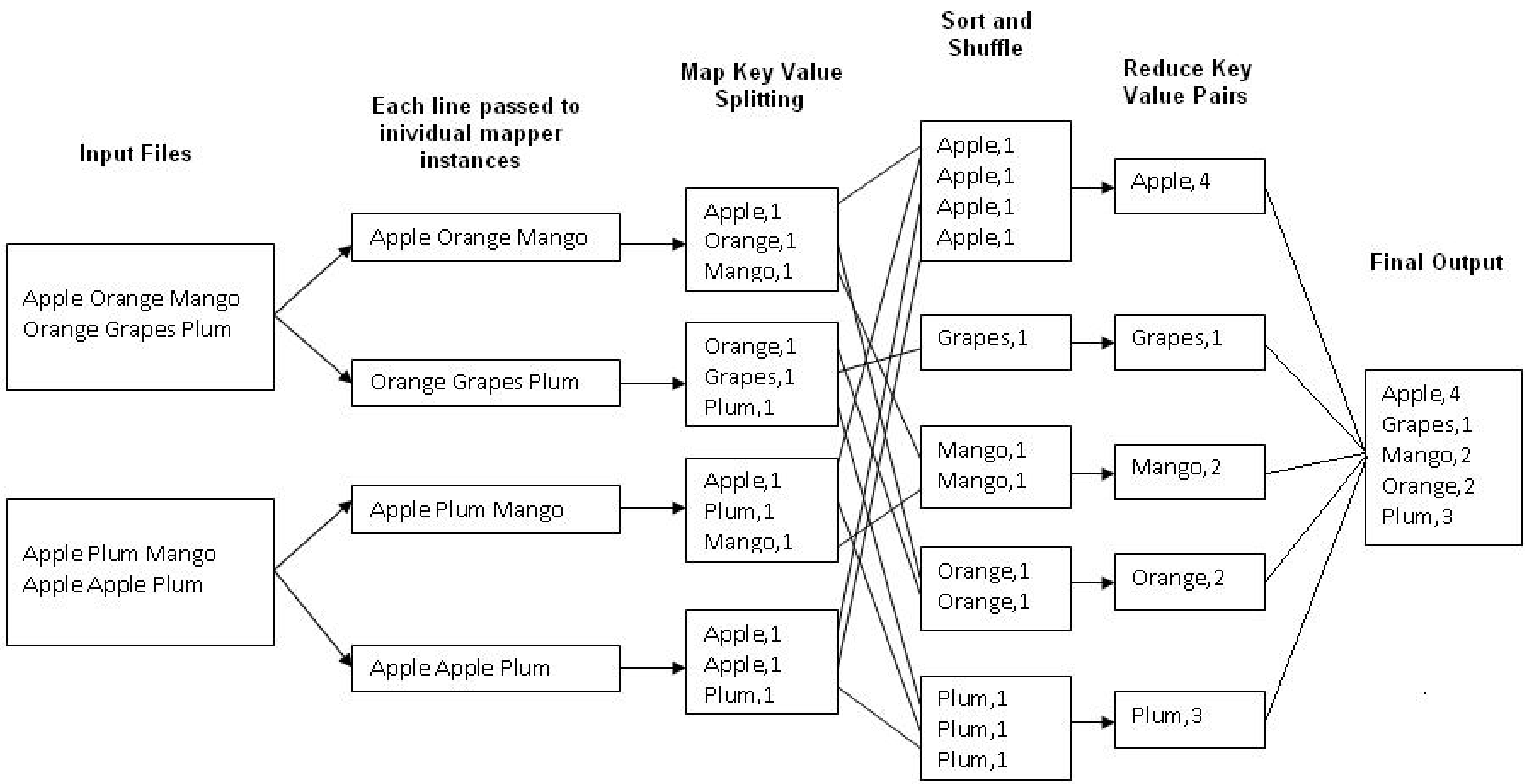
MapReduce

MapReduce workflow



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

MapReduce


Input Files
Each line passed to inividual mapper instances
Map Key Value Splitting
Sort and Shuffle
Reduce Key Value Pairs
Final Output
Apple Orange Mango Orange Grapes Plum
Apple Plum Mango Apple Apple Plum
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple,1 Orange,1 Mango,1
Orange,1 Grapes,1 Plum,1
Apple,1 Plum,1 Mango,1
Apple,1 Apple,1 Plum,1
Apple,1 Apple,1 Apple,1 Apple,1
Grapes,1
Mango,1 Mango,1
Orange,1 Orange,1
Plum,1 Plum,1 Plum,1
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3
Apple,4 Grapes,1 Mango,2 Orange,2 Plum,3

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

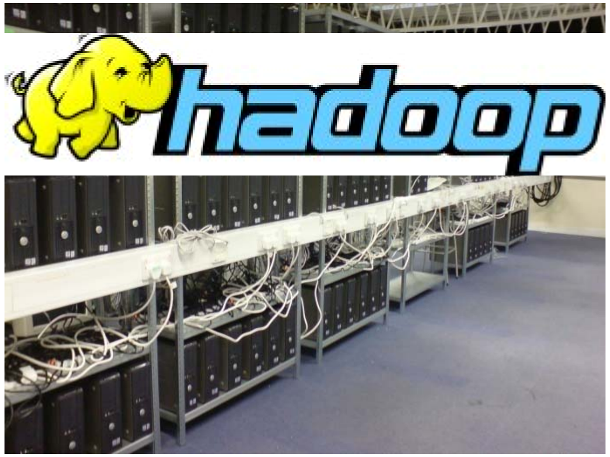
- Advantage: MapReduce’s data-parallel programming model hides complexity of distribution and fault tolerance

- 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

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/

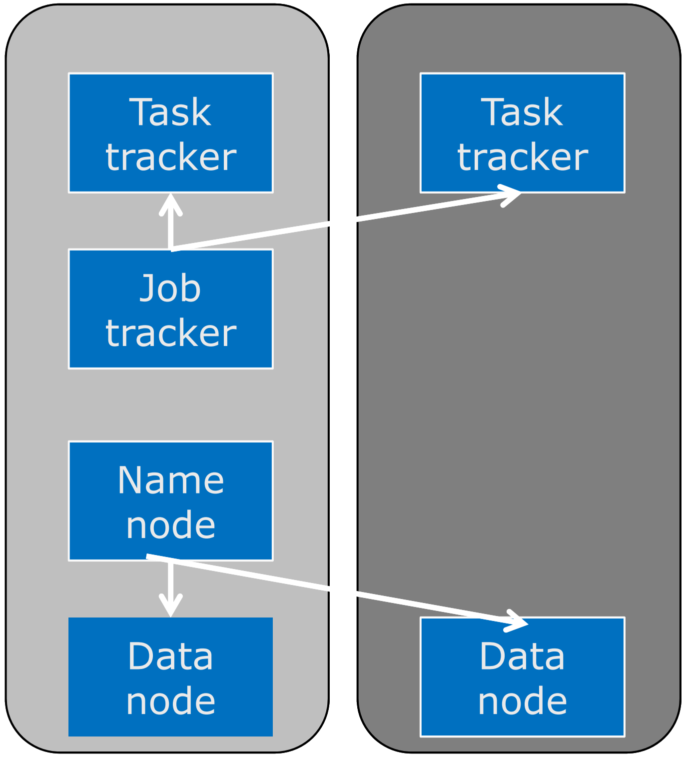
Hadoop

Apache Hadoop is an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license.

Hadoop implements the computational paradigm named MapReduce

Map Reduce Layer

HDFS Layer



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

http://hadoop.apache.org/

Hadoop

How do I access to a Hadoop platform?

Cloud Platform with Hadoop installation

Amazon Elastic Compute Cloud (Amazon EC2)
http://aws.amazon.com/es/ec2/

Windows Azure
http://www.windowsazure.com/

Cluster Instalation
Example ATLAS, SCI^{2}S Research Group

Cluster ATLAS: 4 super servers from Super Micro Computer Inc. (4 nodes per server)
The features of each node are:

- Microprocessors: 2 x Intel Xeon E5-2620 (6 cores/12 threads, 2 GHz, 15 MB Cache)
- RAM 64 GB DDR3 ECC 1600MHz, Registered
- 1 HDD SATA 1TB, 3Gb/s; (system)
- 1 HDD SATA 2TB, 3Gb/s; (distributed file system)

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

	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.

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

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:

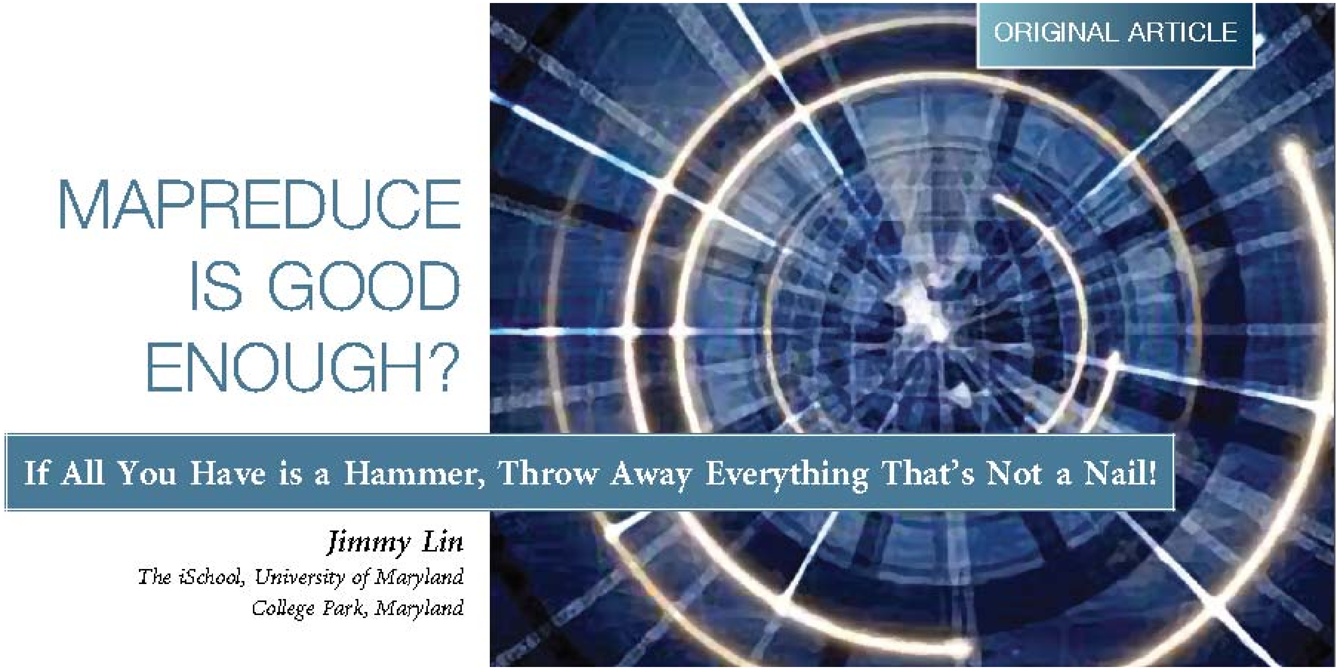
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large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

Recently: Apache Spark

http://hadoop.apache.org/

MapReduce: Limitations

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



The following malfunctions types of algorithms are examples where MapReduce:

Iterative Graph Algorithms: PageRank

Gradient Descent

Expectation Maximization

Pregel (Google)

Pregel: A System for Large-Scale Graph Processing

Hadoop

On the limitations of Hadoop. New platforms

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

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

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

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

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

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

GPU based platforms
Mars
Grex

MapReduce

More than 10000 applications in Google

MapReduce inside Google

Google

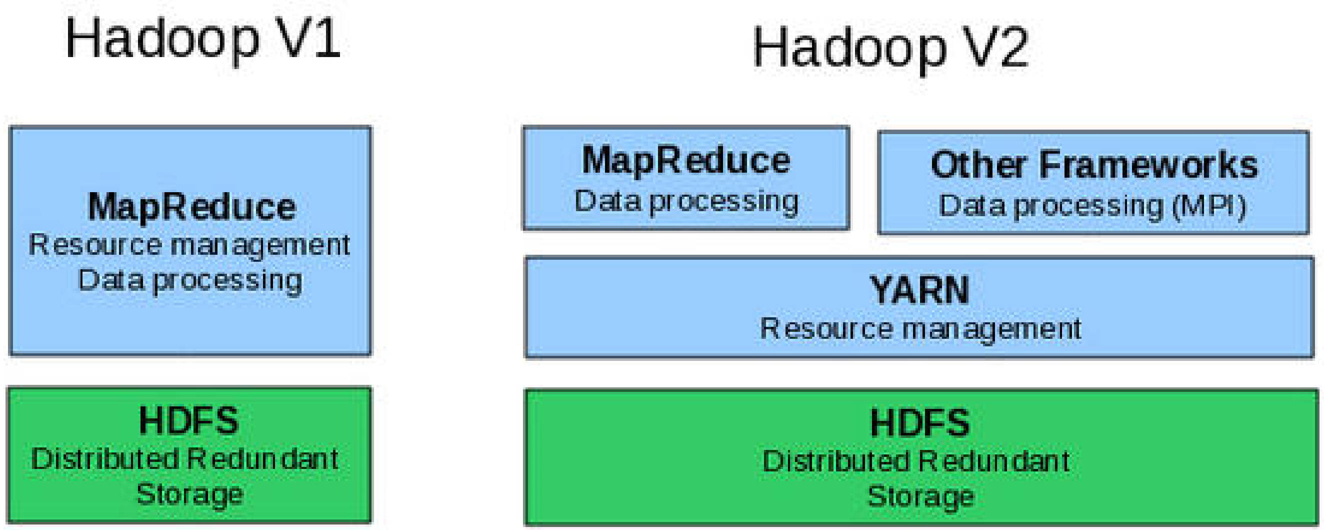
Googlers' hammer for 80% of our data crunching

- Large-scale web search indexing
- Clustering problems for Google News
- Produce reports for popular queries, e.g. Google Trend
- Processing of satellite imagery data
- Language model processing for statistical machine translation
- Large-scale machine learning problems
- Just a plain tool to reliably spawn large number of tasks
  - e.g. parallel data backup and restore

The other 20%? e.g. Pregel

Hadoop Evolution

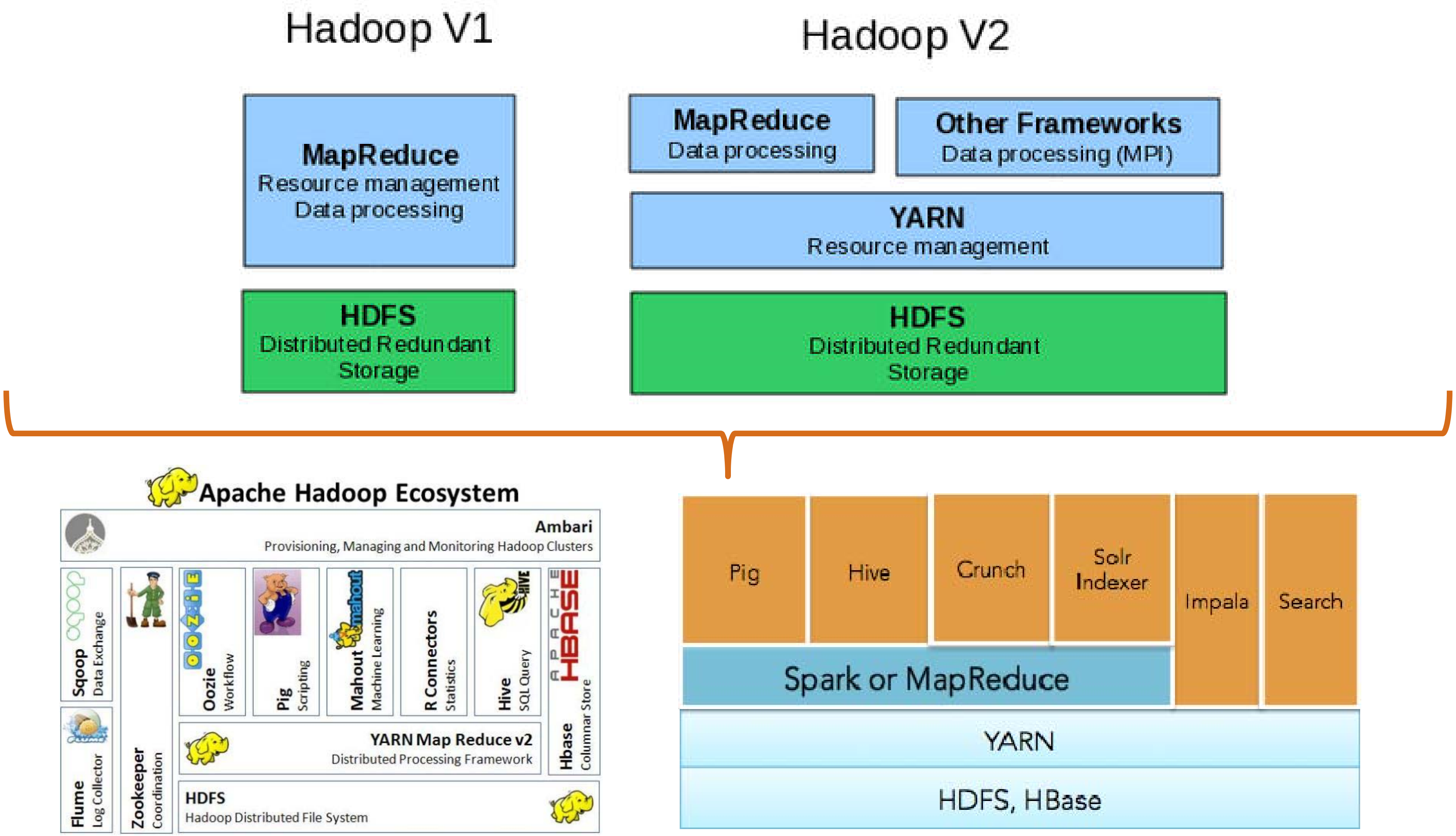
MapReduce Limitations. Graph algorithms (Page Rank, Google), iterative algorithms.



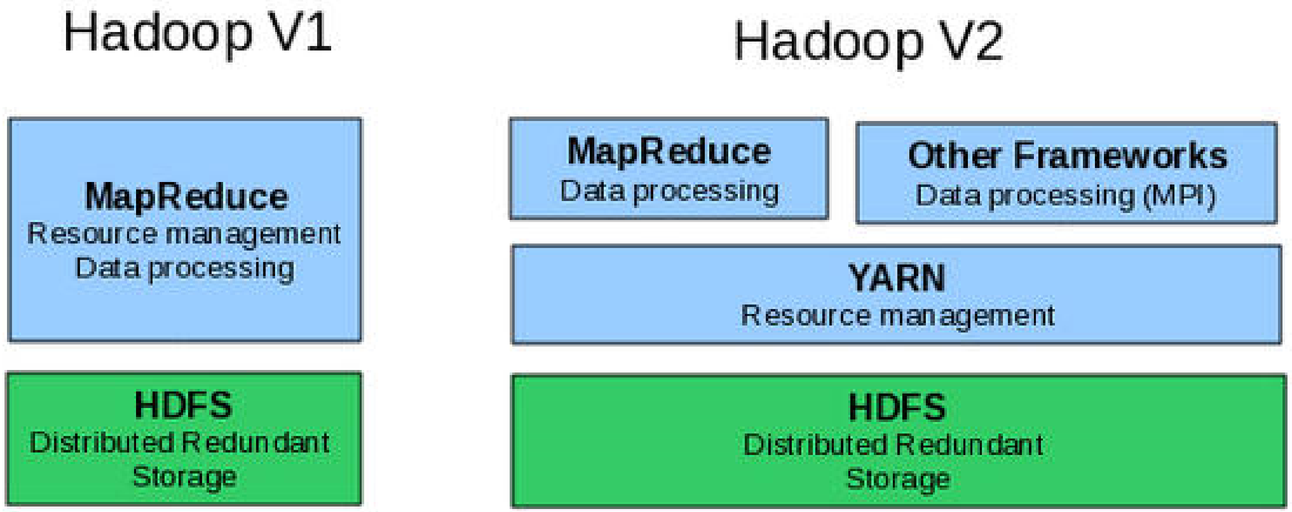
Hadoop Ecosystem

Bibliografía: 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*

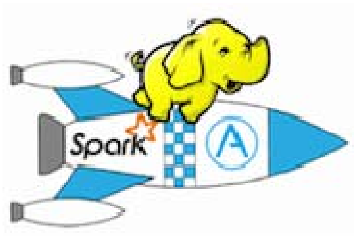
Apache Spark

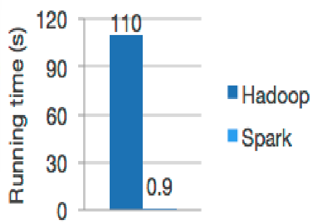


Apache Spark: InMemory

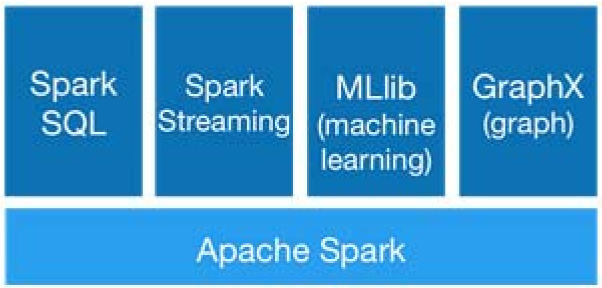


InMemory
HDFS Hadoop + SPARK





Ecosystem
Apache Spark



Future version of Mahout for Spark

Spark birth

Spark
Lightning-Fast Cluster Computing

Gray	Daytona
	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

	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.

	Daytona
Gray	2-way tie: 2014, 4.35 TB/min **TritonSort** 100 TB in 1,378 seconds 186 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.50Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD) Michael Conley, Amin Vahdat, George Porter University of California, San Diego 2014, 4.27 TB/min **Apache Spark** 100 TB in 1,406 seconds 207 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD) Reynold Xin, Parviz Deyhim, Xiangrui Meng, Ali Ghodsi, Matei Zaharia Databricks

http://sortbenchmark.org/

Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Classification

Generation	1st Generation	2nd Generation	3nd 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 Decendent, k-means clustsering, 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

Classification

Mahout

Classification	Single Machine	MapReduce
Logistic Regression - trained via SGD	x	
Naive Bayes / Complementary Naive Bayes		x
Random Forest		x
Hidden Markov Models	x	
Multilayer Perceptron	x	

MLlib

Algorithms

MLlib 1.3 contains the following algorithms:

- linear SVM and logistic regression
- classification and regression tree
- random forest and gradient-boosted trees
- recommendation via alternating least squares
- clustering via k-means, Gaussian mixtures, and power iteration clustering
- topic modeling via latent Dirichlet allocation
- singular value decomposition
- linear regression with L_1- and L_2-regularization
- isotonic regression
- multinomial naive Bayes
- frequent itemset mining via FP-growth
- basic statistics
- feature transformations

Refer to the MLlib guide for usage examples.

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

Classification

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

Classification: Mahout

Scalable machine learning and data mining

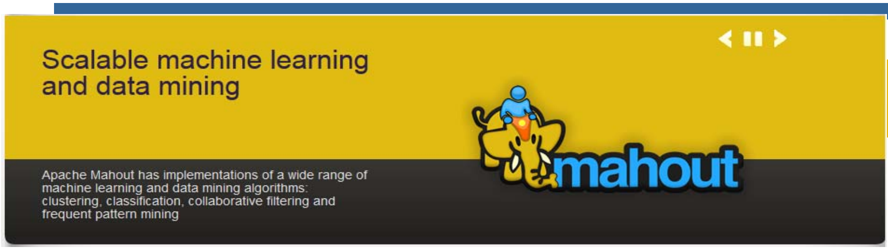
Apache Mahout has implementations of a wide range of machine learning and data mining algorithms: clustering, classification, collaborative filtering and frequent pattern mining

Mahout currently has

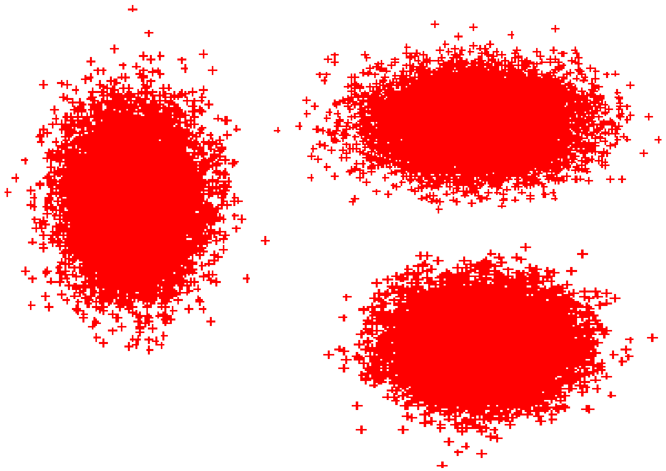
- Collaborative Filtering
- User and Item based recommenders
- K-Means, Fuzzy K-Means clustering
- Mean Shift clustering
- Dirichlet process clustering
- Latent Dirichlet Allocation
- Singular value decomposition
- Parallel Frequent Pattern mining
- Complementary Naive Bayes classifier
- Random forest decision tree based classifier
- High performance java collections (previously colt collections)
- A vibrant community
- and many more cool stuff to come by this summer thanks to Google summer of code

http://mahout.apache.org/

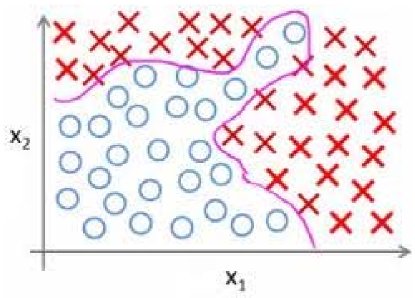
Classification: Mahout



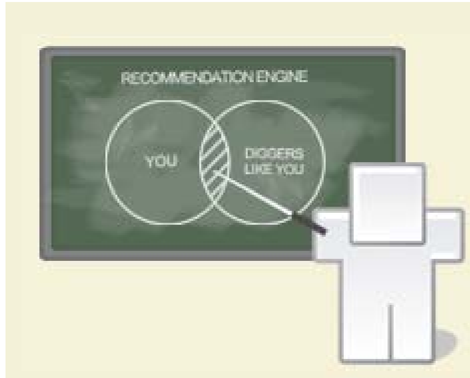
Four great application areas



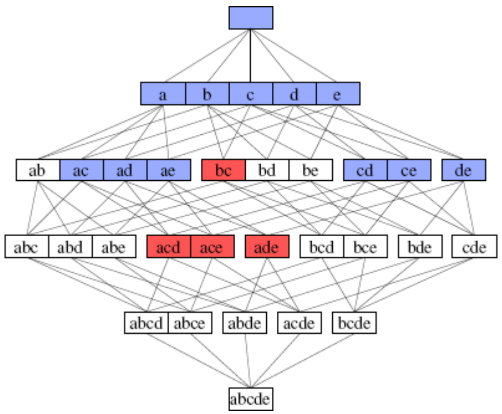
Clustering



Classification

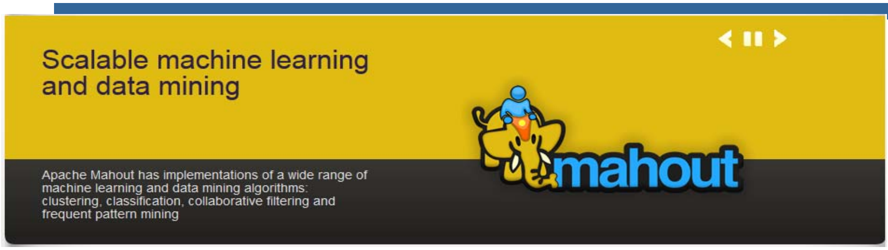


Recommendation Systems

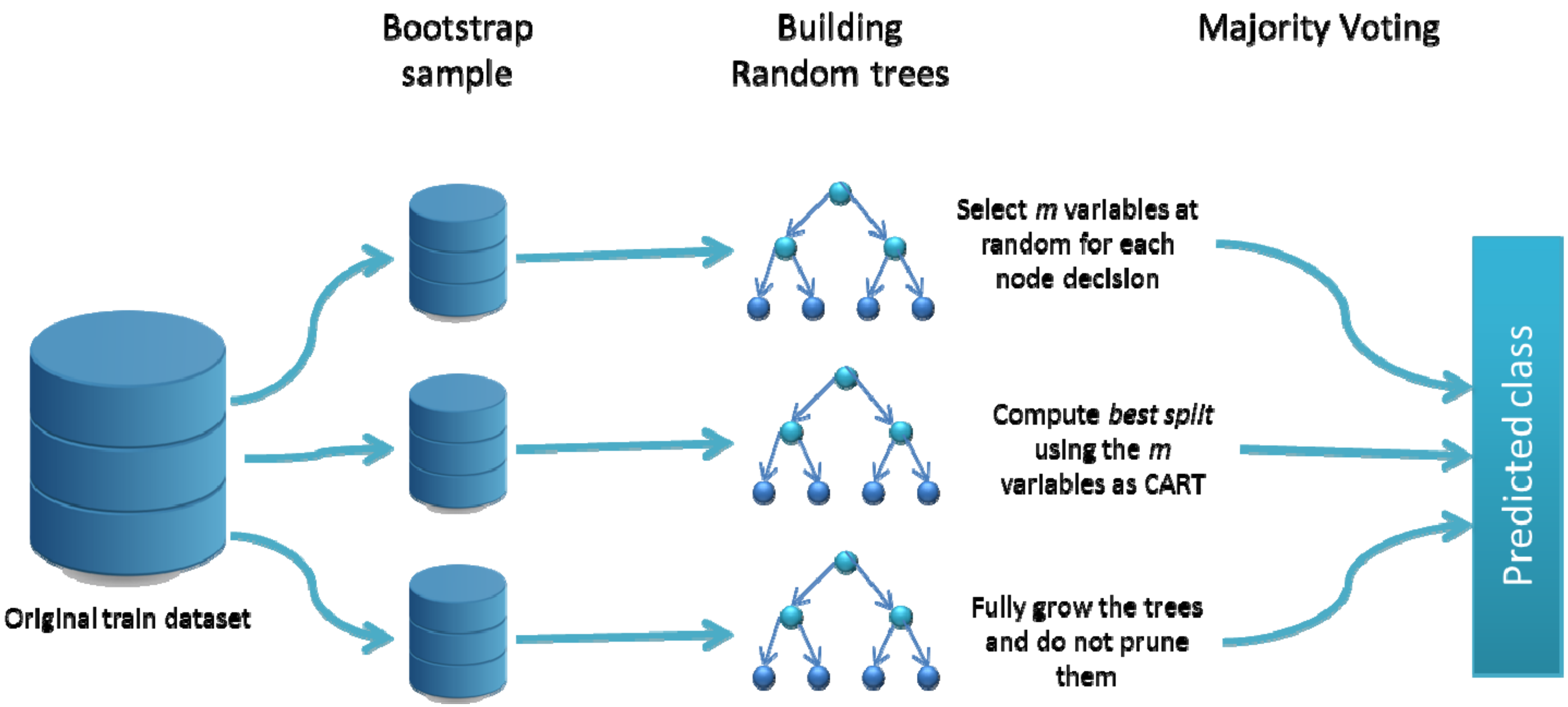


Association

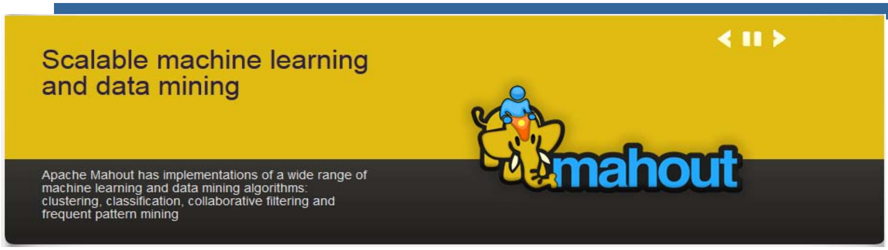
Classification: RF



Case of Study: Random Forest for KddCup'99



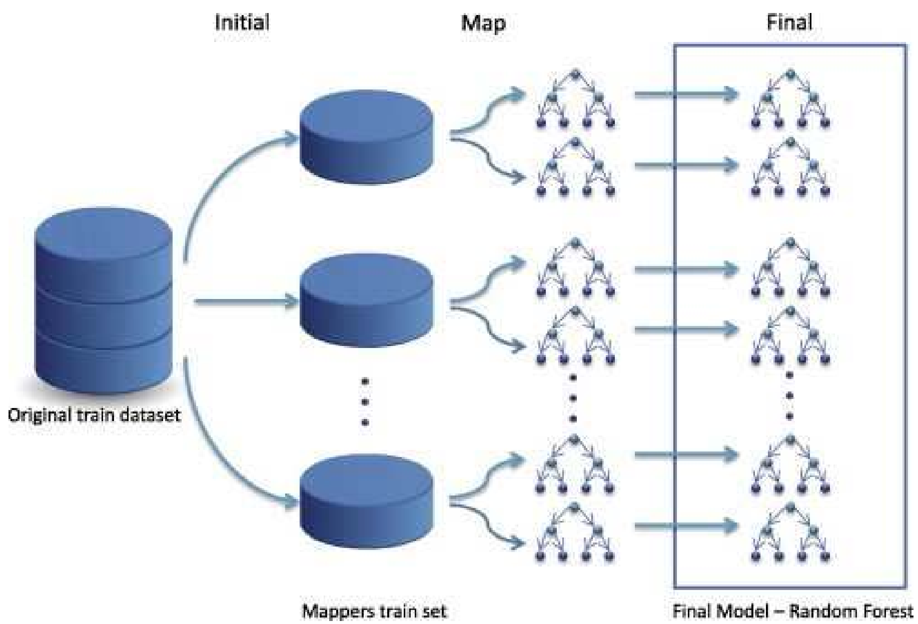
Classification: RF



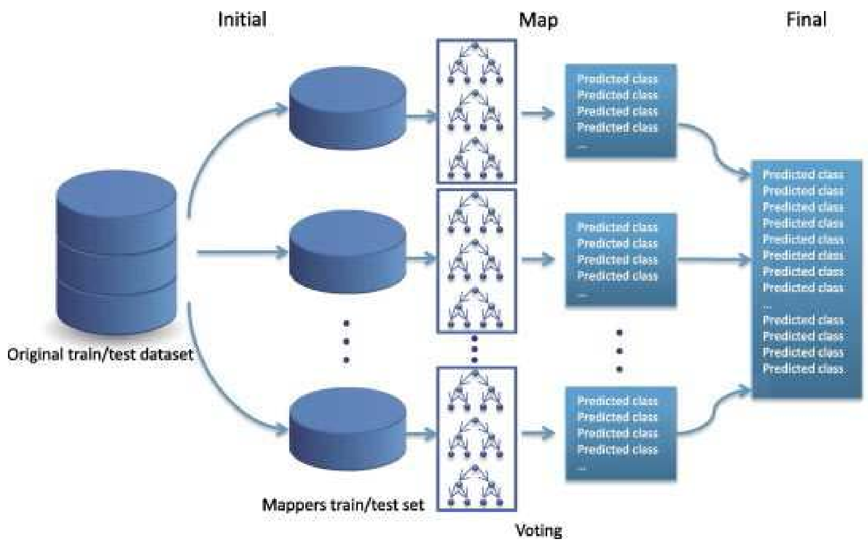
Case of Study: Random Forest for KddCup'99

The RF Mahout Partial implementation: is an algorithm that builds multiple trees for different portions of the data. Two phases:

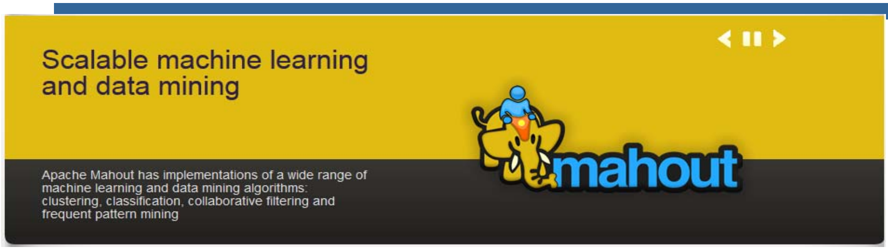
Building phase



Classification phase



Classification: RF



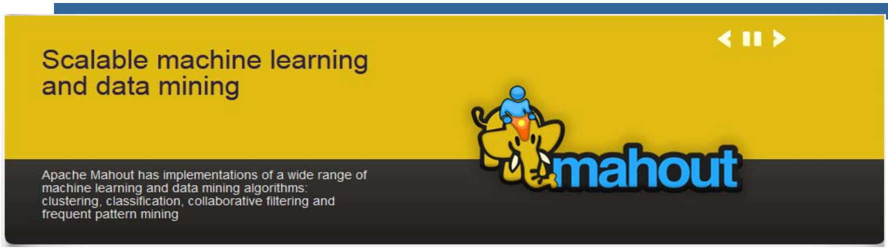
Case of Study: Random Forest for KddCup'99

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

Classification: RF



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:

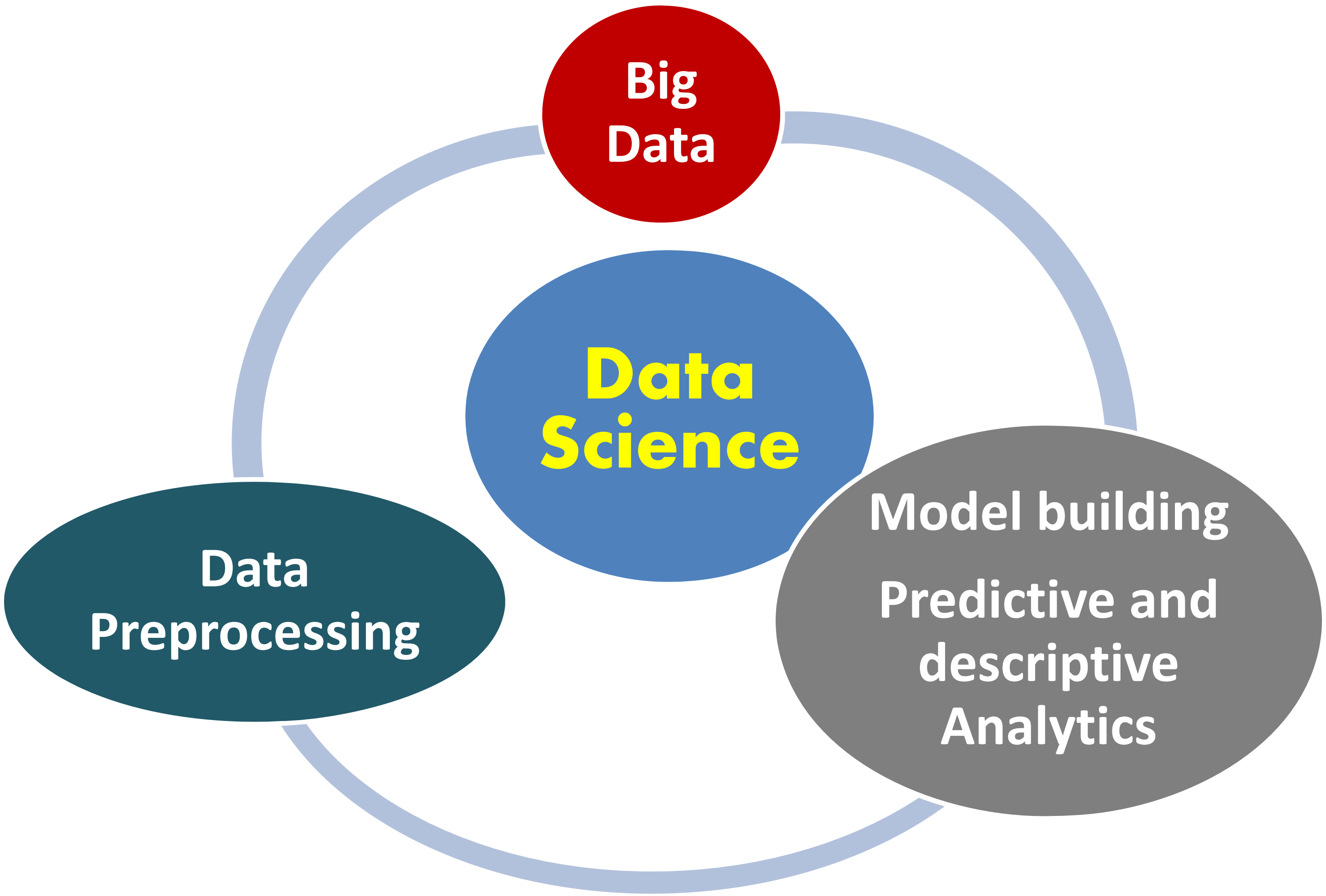
Datasets	RF-BigData		
	10%	50%	full
DOS_versus_normal	98	221	236
DOS_versus_PRB	100	186	190
DOS_versus_R2L	97	157	136
DOS_versus_U2R	93	134	122
normal_versus_PRB	94	58	72
normal_versus_R2L	92	39	69
normal_versus_U2R	93	52	64

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

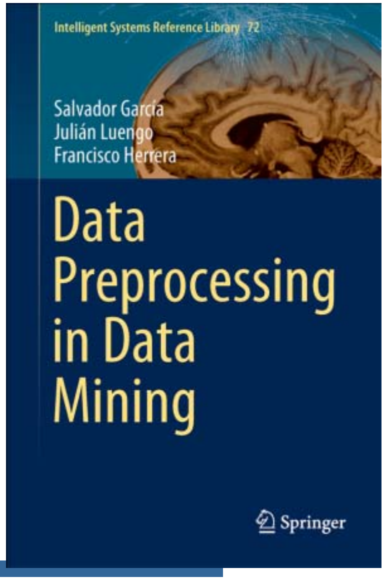
Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big Data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Data Preprocessing



Data Preprocessing



Bibliography:

S. García, J. Luengo, F. Herrera
Data Preprocessing in Data Mining
Springer, January 2015

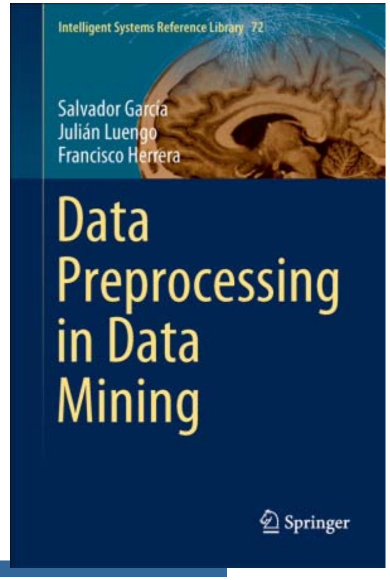
Websites:

http://sci2s.ugr.es/books/data-preprocessing

http://www.springer.com/us/book/9783319102467

1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks

Data Preprocessing

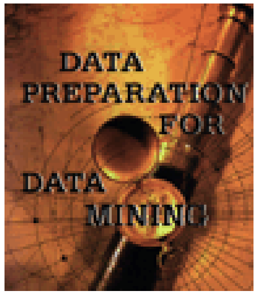


1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks

INTRODUCTION

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

Importance of Data Preprocessing

1. Real data could be dirty and could drive to the extraction of useless patterns/rules.

   This is mainly due to:

   Incomplete data: lacking attribute values, …
   Data with noise: containing errors or outliers
   Inconsistent data (including discrepancies)

Data Preprocessing

Importance of Data Preprocessing

2. Data preprocessing can generate a smaller data set than the original, which allows us to improve the efficiency in the Data Mining process.

   This performing includes Data Reduction techniques:
   Feature selection, sampling or instance selection, discretization.

Data Preprocessing

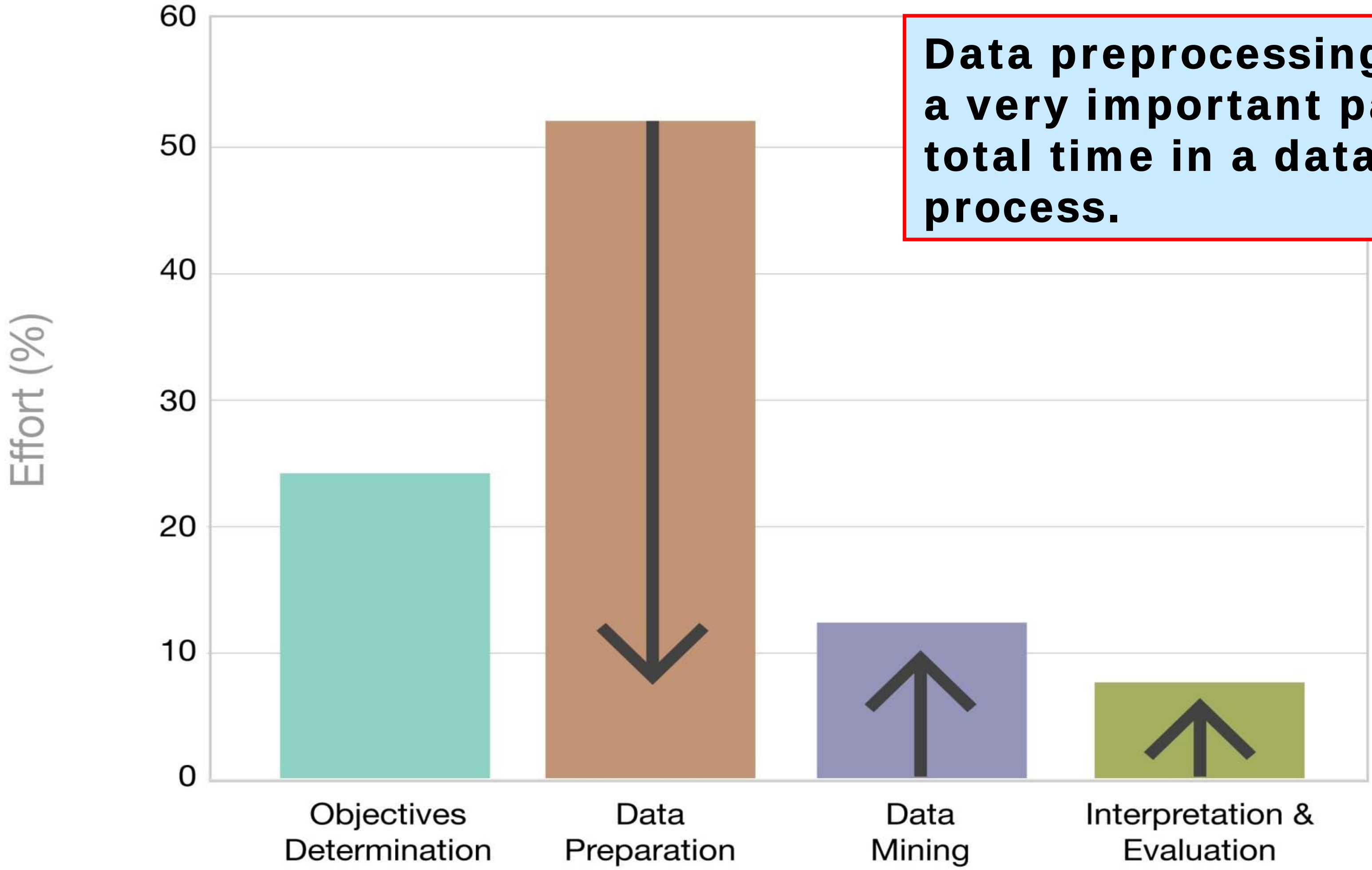
Importance of Data Preprocessing

3. No quality data, no quality mining results!

   Data preprocessing techniques generate “quality data”, driving us to obtain “quality patterns/rules”.

Quality decisions must be based on quality data!

Data Preprocessing



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

Data Preprocessing

What is included in data preprocessing?

Real databases usually contain noisy data, missing data, and inconsistent data, …

Major Tasks in Data Preprocessing

1. Data integration. Fusion of multiple sources in a Data Warehousing.
2. Data cleaning. Removal of noise and inconsistencies.
3. Missing values imputation.
4. Data Transformation.
5. Data reduction.

Data Preprocessing

What is included in data preprocessing?

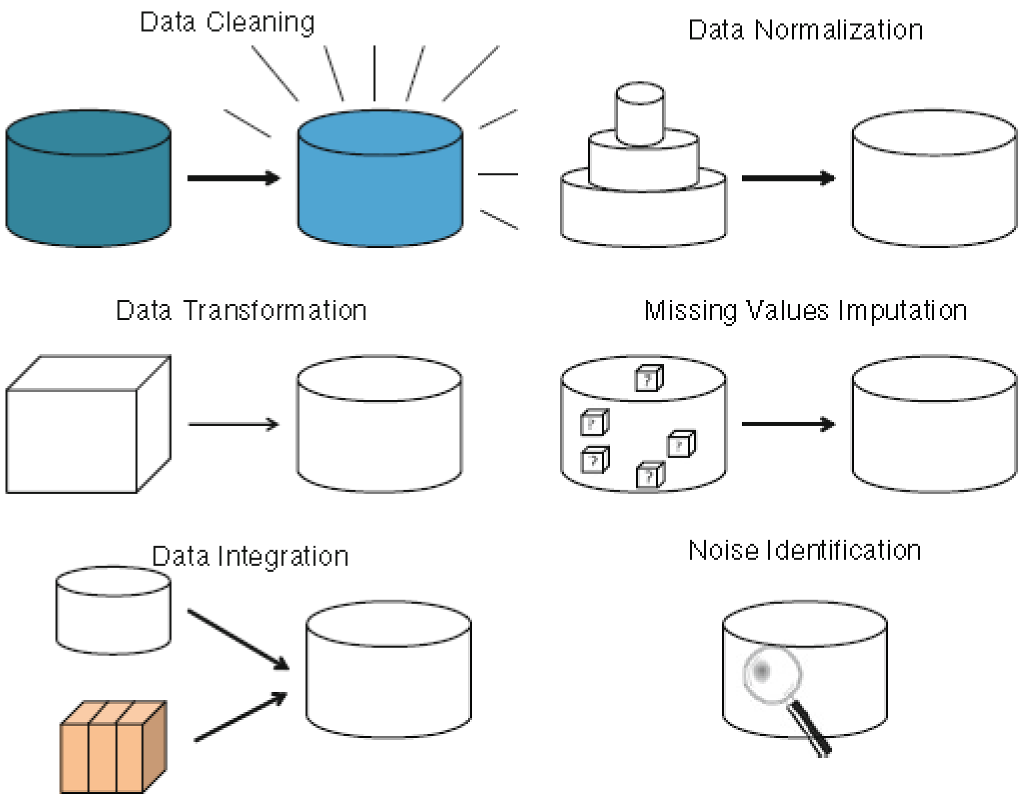


Fig. 1.3 Forms of data preparation

Data Preprocessing

What is included in data preprocessing?

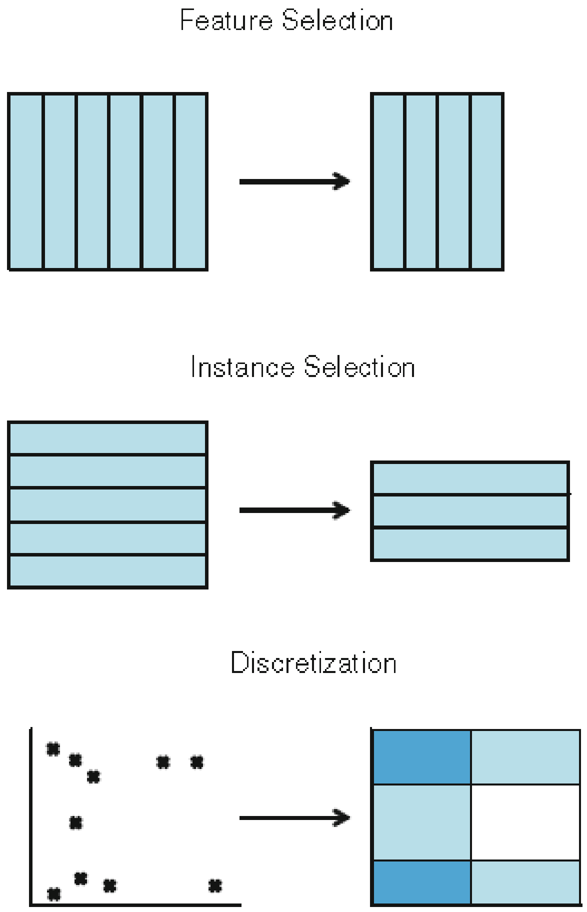
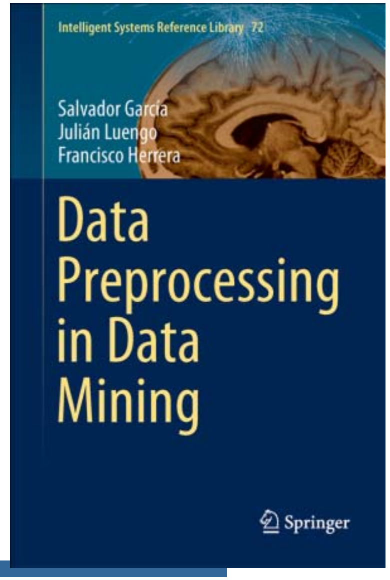


Fig. 1.4 Forms of data reduction

Data Preprocessing



1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks

Integration, Cleaning and Transformation

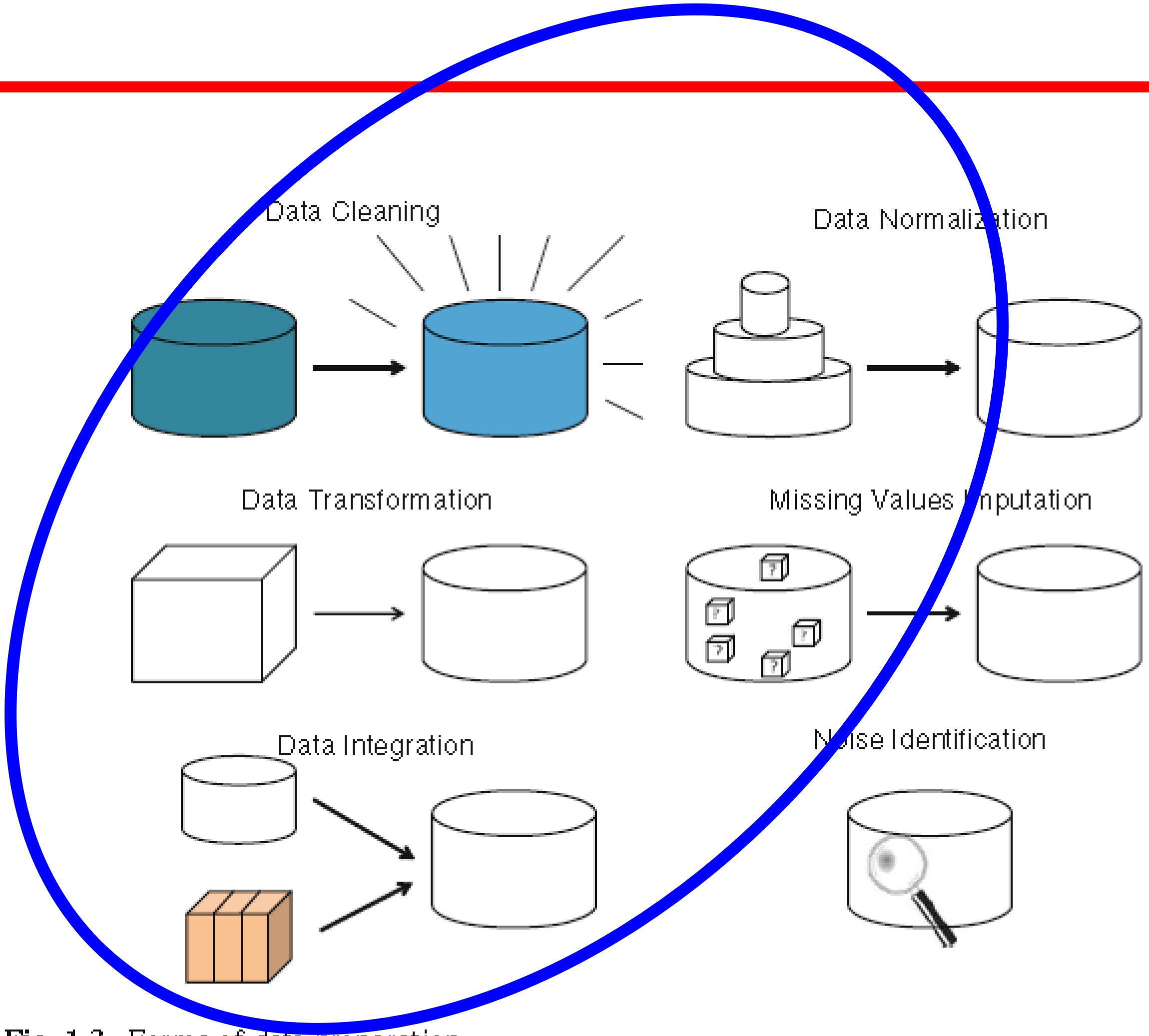
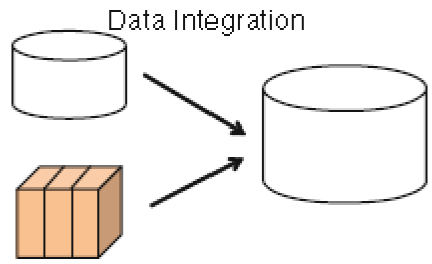
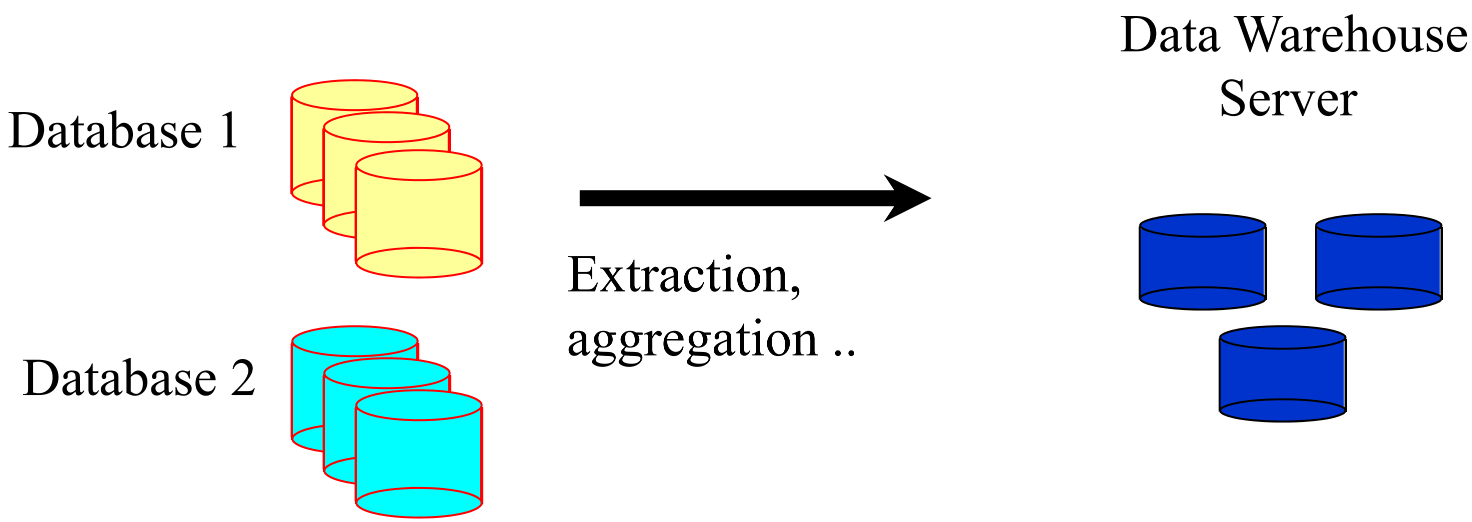


Fig. 1.3 Forms of data preparation

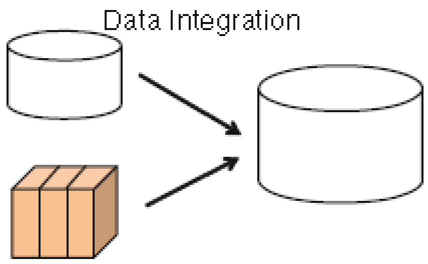
Data Integration



- Obtain data from different information sources.
- Address problems of codification and representation.
- Integrate data from different tables to produce homogeneous information, ...



Data Integration



Examples

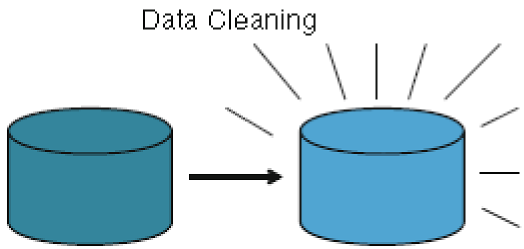
- Different scales: Salary in dollars versus euros (€)

- Derivative attributes: Mensual salary versus annual salary

item	Salary/month
1	5000
2	2400
3	3000

item	Salary
6	50,000
7	100,000
8	40,000

Data Cleaning

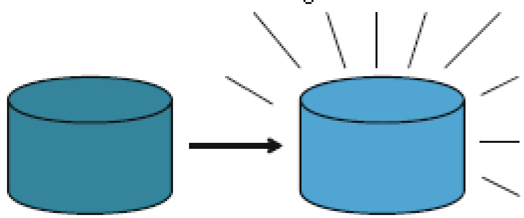


- Objetictives:
  - Fix inconsistencies
  - Fill/impute missing values,
  - Smooth noisy data,
  - Identify or remove *outliers* …

- Some Data Mining algorithms have proper methods to deal with incomplete or noisy data. But in general, these methods are not very robust. It is usual to perform a data cleaning previously to their application.

Bibliography:
W. Kim, B. Choi, E.-D. Hong, S.-K. Kim
A taxonomy of dirty data.
Data Mining and Knowledge Discovery 7, 81-99, 2003.

Data Cleaning



Data cleaning: Example

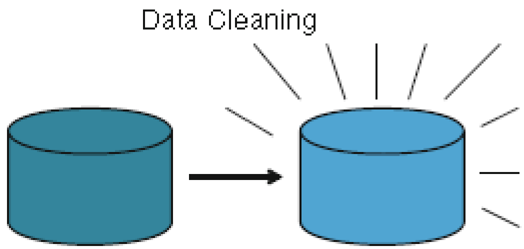
- **Original Data**

**000000000130.06.19971979-10-3080145722 #000310 111000301.01.000100000000004
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000. 000000000000000.000000000000000.0000000......
000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.00
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000
000000000.000000000000000.000000000000000.000000000000000.00 0000000000300.00 0000000000300.00**

- **Clean Data**

**0000000001,199706,1979.833,8014,5722 , ,#000310 ....
,111,03,000101,0,04,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,0300.00**

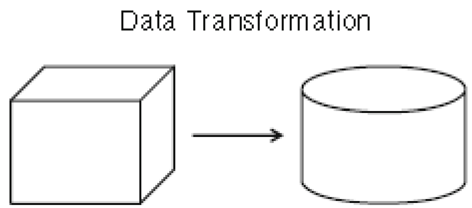
Data Cleaning



Data Cleaning: Inconsistent data

Age="42"
Birth Date="03/07/1997"

Data transformation

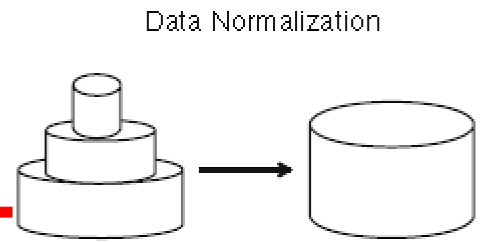


- Objective: To transform data in the best way possible to the application of Data Mining algorithms.

- Some typical operations:
  - Aggregation. i.e. Sum of the totality of month sales in an unique attribute called anual sales,...
  - Data generalization. It is to obtain higher degrees of data from the currently available, by using concept hierarchies.
    - streets → cities
    - Numerical age → {young, adult, half-age, old}
  - Normalization: Change the range [-1,1] or [0,1].
  - Lineal transformations, quadratic, polinominal, …

Bibliography:
T. Y. Lin. Attribute Transformation for Data Mining I: Theoretical Explorations. International Journal of Intelligent Systems 17, 213-222, 2002.

Normalization



- Objective: convert the values of an attribute to a better range.

- Useful for some techniques such as Neural Networks o distance-based methods (k-Nearest Neighbors,…).

- Some normalization techniques:

  Z-score normalization

$$v' = \frac{v - \overline{A}}{\sigma_A}$$

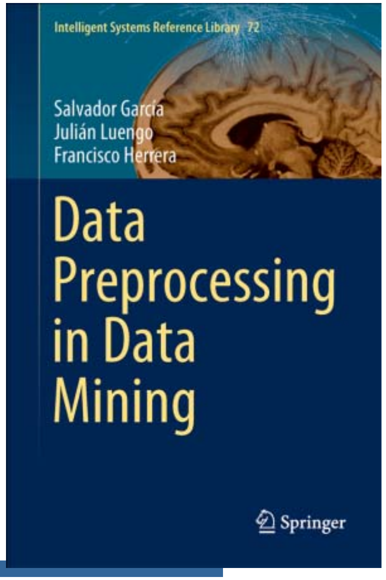
  min-max normalization: Perform a lineal transformation of the original data.

$$[\min_A, \max_A] \to [new_{\min_A}, new_{\max_A}]$$

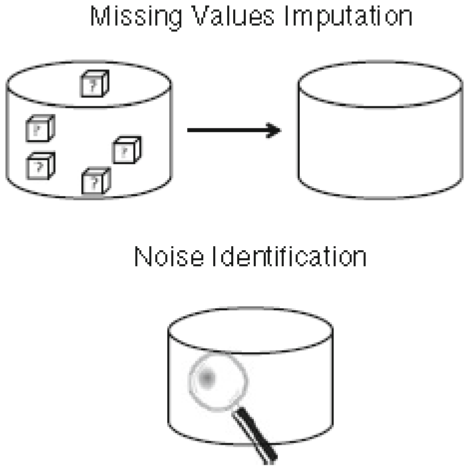
$$v' = \frac{v - \min_A}{\max_A - \min_A}(new_{\max_A} - new_{\min_A}) + new_{\min_A}$$

  The relationships among original data are maintained.

Data Preprocessing



1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks



Imperfect data

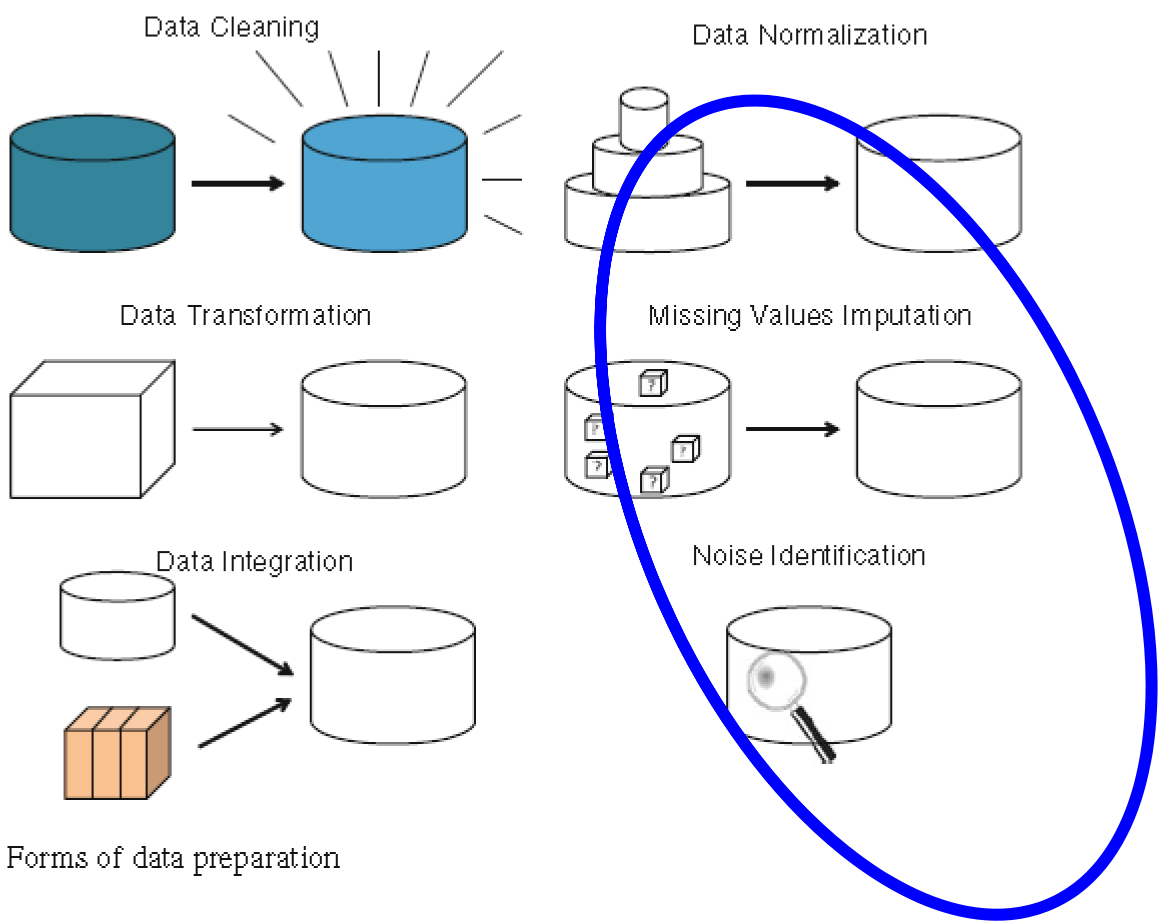
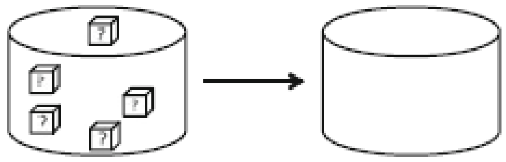


Fig. 1.3 Forms of data preparation

Missing values

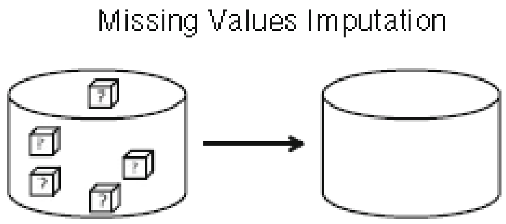
Missing Values Imputation



Attributes

Instances	1	2	3	4	5	…	m
1					?		
2			?				
3		?		?			
4							
5							
6						?	
7			?		?		
8							
9							
10			?			?	
11		?					
⋮				?			
n							?

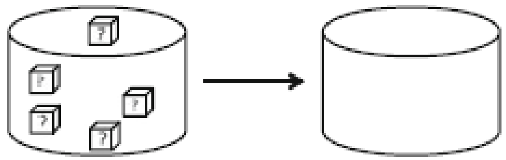
Missing values



It could be used the next choices, although some of them may skew the data:

- Ignore the tuple. It is usually used when the variable to classify has no value.
- Use a global constant for the replacement. I.e. “unknown”,”?”,…
- Fill tuples by means of mean/deviation of the rest of the tuples.
- Fill tuples by means of mean/deviation of the rest of the tuples belonging to the same class.
- Impute with the most probable value. For this, some technique of inference could be used, i.e., bayesian or decision trees.

Missing Values Imputation



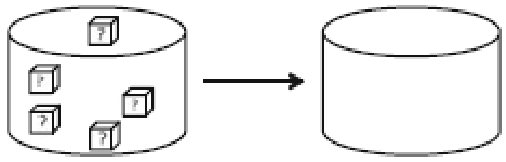
Missing values

MISSING VALUES		
Full Name	Short Name	Reference
Delete Instances with Missing Values	Ignore-MV	P.A. Gourraud, E. Ginin, A. Cambon-Thomsen. Handling Missing Values In Population Data: Consequences For Maximum Likelihood Estimation Of Haplotype Frequencies. European Journal of Human Genetics 12:10 (2004) 805-812.
Event Covering Synthesizing	EventCovering-MV	D.K.Y. Chiu, A.K.C. Wong. Synthesizing Knowledge: A Cluster Analysis Approach Using Event-Covering. IEEE Transactions on Systems, Man and Cybernetics, Part B 16:2 (1986) 251-259.
K-Nearest Neighbor Imputation	KNN-MV	G.E.A.P.A. Batista, M.C. Monard. An Analysis Of Four Missing Data Treatment Methods For Supervised learning. Applied Artificial Intelligence 17:5 (2003) 519-533.
Most Common Attribute Value	MostCommon-MV	J.W. Grzymala-Busse, L.K. Goodwin, W.J. Grzymala-Busse, X. Zheng. Handling Missing Attribute Values in Preterm Birth Data Sets. 10th International Conference of Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC'05). LNCS 3642, Springer 2005, Regina (Canada, 2005) 342-351.
Assign All Posible Values of the Attribute	AllPossible-MV	J.W. Grzymala-Busse. On the Unknown Attribute Values In Learning From Examples. 6th International Symposium on Methodologies For Intelligent Systems (ISMIS91). Charlotte (USA, 1991) 368-377.
K-means Imputation	KMeans-MV	J. Deogun, W. Spaulding, B. Shuart, D. Li. Towards Missing Data Imputation: A Study of Fuzzy K-means Clustering Method. 4th International Conference of Rough Sets and Current Trends in Computing (RSCTC'04). LNCS 3066, Springer 2004, Uppsala (Sweden, 2004) 573-579.
Concept Most Common Attribute Value	ConceptMostCommon-MV	J.W. Grzymala-Busse, L.K. Goodwin, W.J. Grzymala-Busse, X. Zheng. Handling Missing Attribute Values in Preterm Birth Data Sets. 10th International Conference of Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC'05). LNCS 3642, Springer 2005, Regina (Canada, 2005) 342-351.

15 methods

http://www.keel.es/

Missing Values Imputation



Missing values

Algorithm 3 kNNI algorithm.

function kNNI(T - dataset with MVs, k - number of neighbors per instance to be chosen, $D(x, y)$ - a distance or dissimilarity function of x and y, S - the imputed version of T)

    initialize: $S = \{\}$

    for each instance y_i in T **do**

        $\widehat{y_i} \leftarrow y_i$

        if y_i contains any missing value **then**

            Find set I_{Ki} with the k nearest instances to y_i from T using D

            for each missing value in attribute h of y_i **do**

                if h is numerical **then**

                    $\widehat{y_{ih}} = \left(\sum_{j \in I_{Kih}} y_{jh}\right) / (|I_{Kih}|)$

                else

                    $\widehat{y_{ih}} = mode(I_{Kih})$

                end if

            end for

        end if

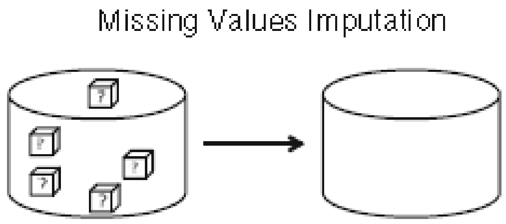
        $S \leftarrow \widehat{y_{ih}}$

    end for

    return S

end function

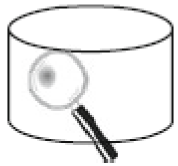
Missing values



Bibliography:
WEBSITE: http://sci2s.ugr.es/MVDM/

J. Luengo, S. García, F. Herrera, **A Study on the Use of Imputation Methods for Experimentation with Radial Basis Function Network Classifiers Handling Missing Attribute Values: The good synergy between RBFs and EventCovering method**. *Neural Networks, doi:10.1016/j.neunet.2009.11.014, 23(3) (2010) 406-418*.

S. García, F. Herrera, **On the choice of the best imputation methods for missing values considering three groups of classification methods**. *Knowledge and Information Systems 32:1 (2012) 77-108, doi:10.1007/s10115-011-0424-2*



Noise cleaning

Types of examples

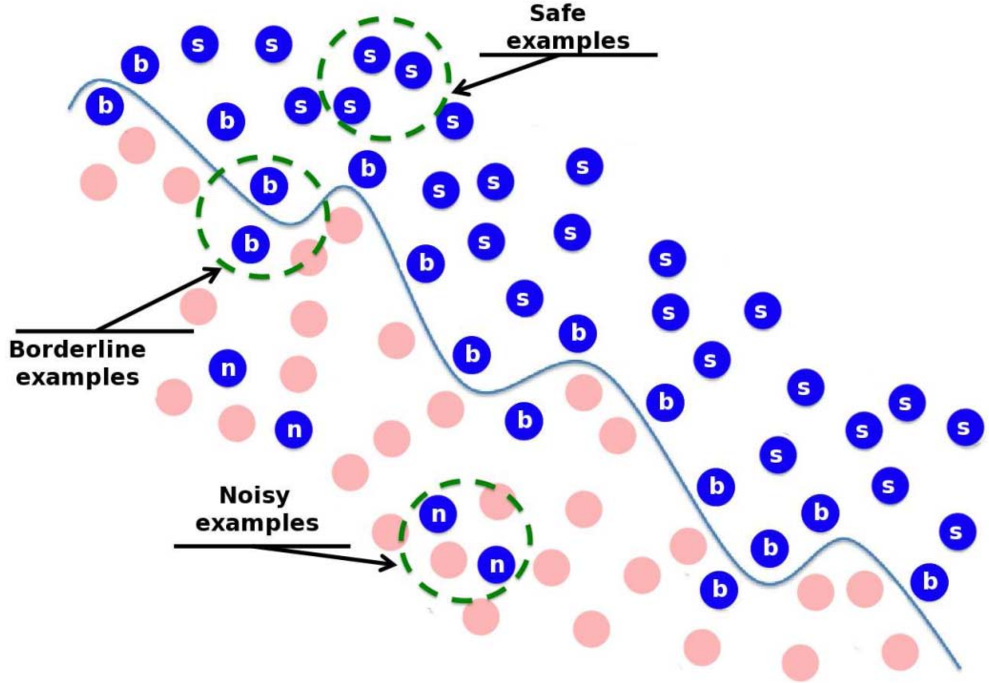
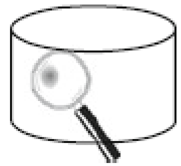


Fig. 5.2 The three types of examples considered in this book: safe examples (labeled as *s), borderline* examples (labeled as *b) and noisy examples (labeled as n). The continuous line shows the* decision boundary between the two classes



Noise cleaning

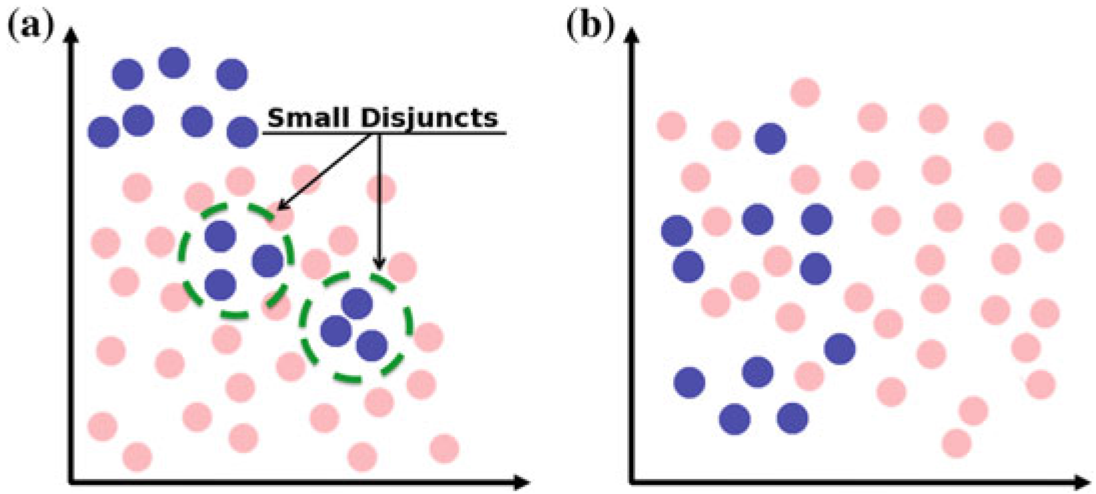
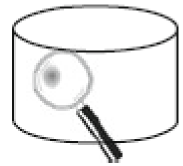


Fig. 5.1 Examples of the interaction between classes: a) small disjuncts and b) overlapping between classes



Noise cleaning

Use of noise filtering techniques in classification

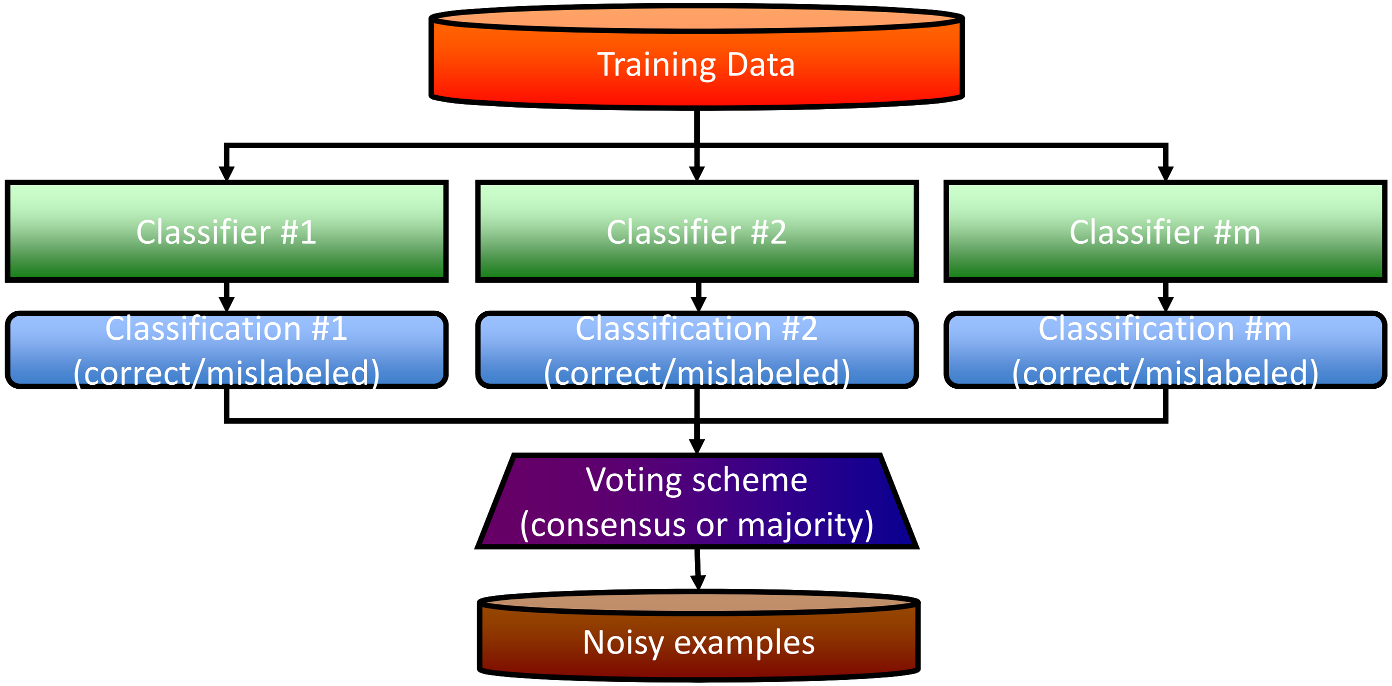
The three noise filters mentioned next, which are the most-known, use a voting scheme to determine what cases have to be removed from the training set:

- ***Ensemble Filter (EF)***
- ***Cross-Validated Committees Filter***
- ***Iterative-Partitioning Filter***

Ensemble Filter (EF)

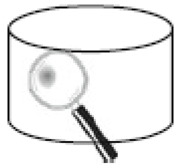
- C.E. Brodley, M.A. Friedl. **Identifying Mislabeled Training Data**. *Journal of Artificial Intelligence Research 11 (1999) 131-167*.
- **Different learning algorithm** (C4.5, 1-NN and LDA) are used to create classifiers in several subsets of the training data that serve as noise filters for the training sets.
- Two main steps:

1. For each learning algorithm, a ***k-fold cross-validation*** is used to tag each training example as **correct** (prediction = training data label) or **mislabeled** (prediction ≠ training data label).
2. A ***voting scheme*** is used to identify the final set of noisy examples.
    - **Consensus voting**: it removes an example if it is misclassified by all the classifiers.
    - **Majority voting**: it removes an instance if it is misclassified by more than half of the classifiers.



Ensemble Filter (EF)

Noise Identification



Algorithm 4 EF algorithm.

function EF(T - dataset with MVs, Γ - number of subsets, μ - number of filters to be used, F - set of classifiers)

  Split the training data set T into $T_i, i = 1 \dots \Gamma$ equal sized subsets

  for each filter $F_x, x = 1 to \mu$ **do**

    for each subset T_i **do**

      Use $\{T_j, j \neq i\}$ to train F_x resulting in F_x^i

      for each instance t in T_i **do**

        Classify t with every F_x^i

      end for

    end for

  end for

  for each instance t in T **do**

    Use a voting scheme to include t in T_N according to the classifications made by each filter F_x

  end for

  return $T - T_N$

end function

Cross-Validated Committees Filter (CVCF)

- S. Verbaeten, A.V. Assche. **Ensemble methods for noise elimination in classification problems**. *4th International Workshop on Multiple Classifier Systems (MCS 2003). LNCS 2709, Springer 2003, Guilford (UK, 2003) 317-325*.

- CVCF is similar to EF ➔ two main differences:

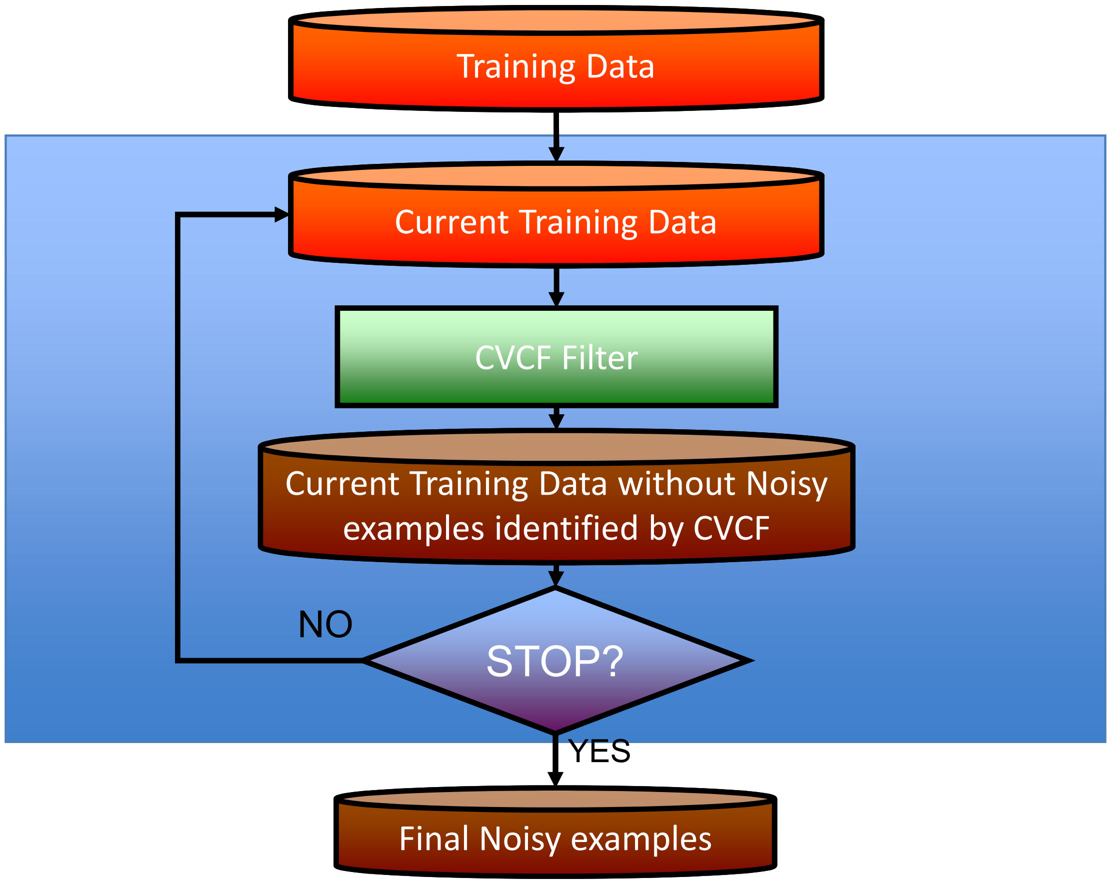
  1. **The same learning algorithm (C4.5)** is used to create classifiers in several subsets of the training data.

     The authors of CVCF place special emphasis on using **ensembles of decision trees** such as C4.5 because they work well as a filter for noisy data.

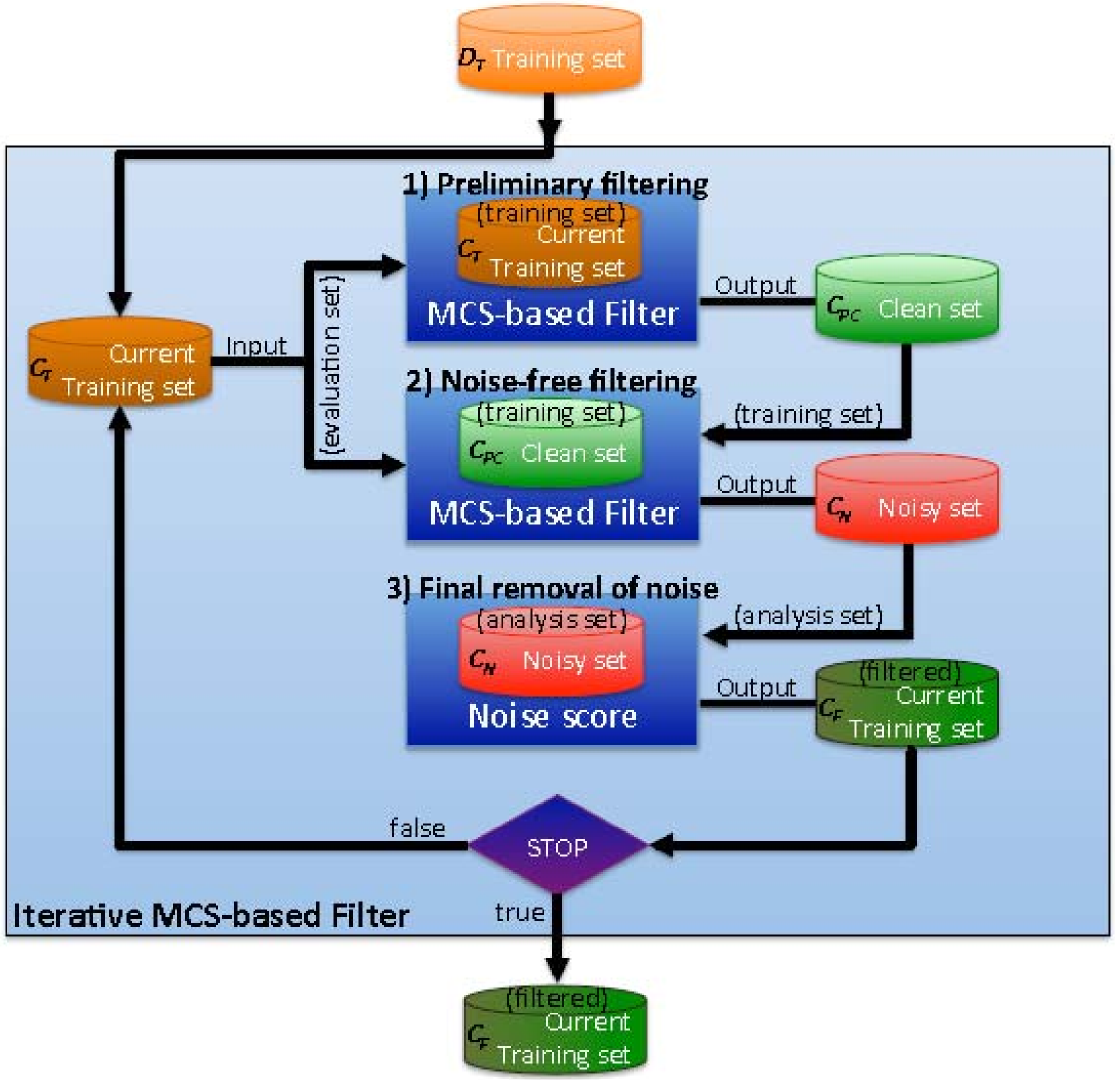
  2. Each classifier built with the ***k-fold cross-validation*** is used to tag **ALL the training examples** (not only the test set) as **correct** (prediction = training data label) or **mislabeled** (prediction ≠ training data label).

Iterative Partitioning Filter (IPF)

- T.M. Khoshgoftaar, P. Rebours. **Improving software quality prediction by noise filtering techniques**. *Journal of Computer Science and Technology 22 (2007) 387-396.*
- IPF removes noisy data in **multiple iterations** using **CVCF** until a stopping criterion is reached.
- The iterative process stops if, for a number of consecutive iterations, the number of noisy examples in each iteration is less than a percentage of the size of the training dataset.



INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control



INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control

Method	Test accuracy						
	0%	5%	10%	15%	20%	25%	30%
C4.5							
AllKNN	79.20	78.87	78.48	78.00	77.39	77.36	76.35
CF	80.43	80.21	79.83	79.37	78.87	78.63	78.01
ENN	80.09	79.87	79.75	79.06	78.76	78.33	77.65
EF	80.41	80.22	79.83	79.56	79.33	79.01	78.46
IPF	81.18	80.79	80.56	79.92	79.32	79.27	79.03
ME	77.88	77.60	76.79	76.60	75.52	75.32	74.45
NCNE	80.58	80.23	79.98	79.18	78.65	78.43	77.41
INFFC	**81.77**	**81.57**	**81.21**	**80.97**	**80.44**	**80.07**	**79.99**
None	81.32	80.89	80.35	79.46	78.24	77.21	76.16

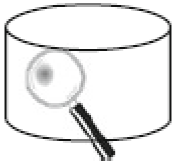
Best (out of 25)						
0%	5%	10%	15%	20%	25%	30%
1	0	0	1	1	1	1
0	0	0	0	1	0	0
0	2	2	0	3	5	1
2	5	6	5	5	3	3
6	3	2	3	1	5	1
3	1	0	0	0	2	0
1	1	2	3	1	1	0
8	**9**	**9**	**12**	**12**	**8**	**17**
7	5	5	2	2	0	2

ELSEVIER
Information Fusion
Volume 27, January 2016, Pages 19–32

INFFC: An iterative class noise filter based on the fusion of classifiers with noise sensitivity control

José A. Sáez[a], Mikel Galar[c], Julián Luengo[d], Francisco Herrera[b]

http://www.sciencedirect.com/science/article/pii/S156625351500038X



Noise cleaning

Bibliography:

WEBSITE:

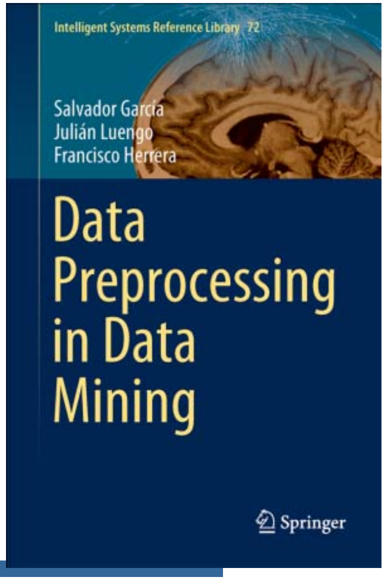
http://sci2s.ugr.es/noisydata/

http://www.keel.es/

NOISY DATA FILTERING

Full Name	Short Name	Reference
Saturation Filter	SaturationFilter-F	D. Gamberger, N. Lavrac, S. Dzroski. Noise detection and elimination in data preprocessing: Experiments in medical domains. Applied Artificial Intelligence 14:2 (2000) 205-223.
Pairwise Attribute Noise Detection Algorithm Filter	PANDA-F	J.D. Hulse, T.M. Khoshgoftaar, H. Huang. The pairwise attribute noise detection algorithm. Knowledge and Information Systems 11:2 (2007) 171-190.
Classification Filter	ClassificationFilter-F	D. Gamberger, N. Lavrac, C. Groselj. Experiments with noise filtering in a medical domain. 16th International Conference on Machine Learning (ICML99). San Francisco (USA, 1999) 143-151.
Automatic Noise Remover	ANR-F	X. Zeng, T. Martinez. A Noise Filtering Method Using Neural Networks. IEEE International Workshop on Soft Computing Techniques in Instrumentation, Measurement and Related Applications (SCIMA2003). Utah (USA, 2003) 26-31.
Ensemble Filter	EnsembleFilter-F	C.E. Brodley, M.A. Friedl. Identifying Mislabeled Training Data. Journal of Artificial Intelligence Research 11 (1999) 131-167.
Cross-Validated Committees Filter	CVCommitteesFilter-F	S. Verbaeten, A.V. Assche. Ensemble methods for noise elimination in classification problems. 4th International Workshop on Multiple Classifier Systems (MCS 2003). LNCS 2709, Springer 2003, Guilford (UK, 2003) 317-325.
Iterative-Partitioning Filter	IterativePartitioningFilter-F	T.M. Khoshgoftaar, P. Rebours. Improving software quality prediction by noise filtering techniques. Journal of Computer Science and Technology 22 (2007) 387-396.

Data Preprocessing



1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks

Data Reduction

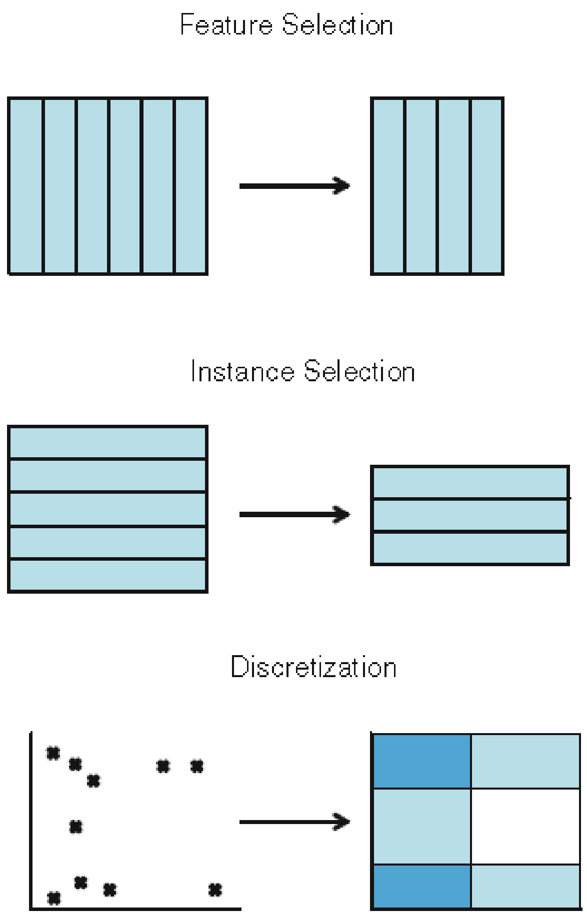
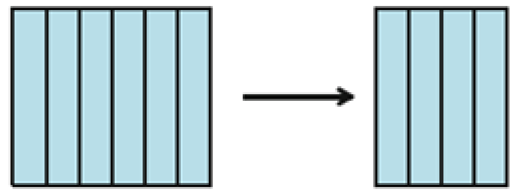


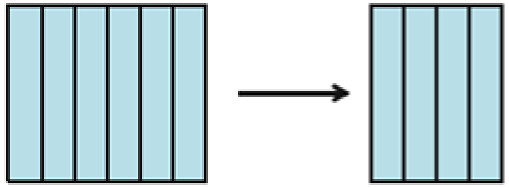
Fig. 1.4 Forms of data reduction

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



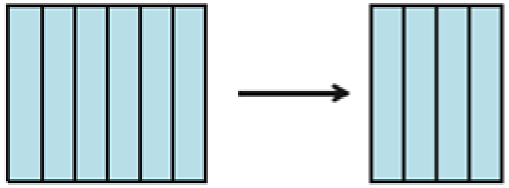
Var. 1.

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

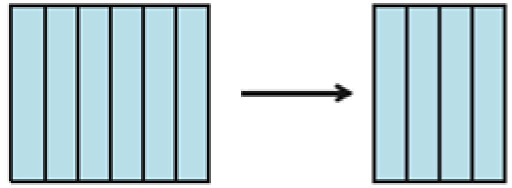


The problem of *Feature Subset Selection (FSS*) consists of finding a subset of the attributes/features/variables of the data set that optimizes the probability of success in the subsequent data mining taks.

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 couls learn quickly
  - Higher accuracy → the algorithm better generalizes
  - Simpler results → easier to understand them

- **FS has as extension the extraction and construction of attributes.**

Feature Selection

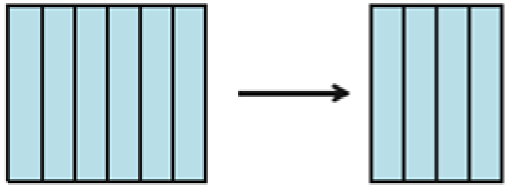
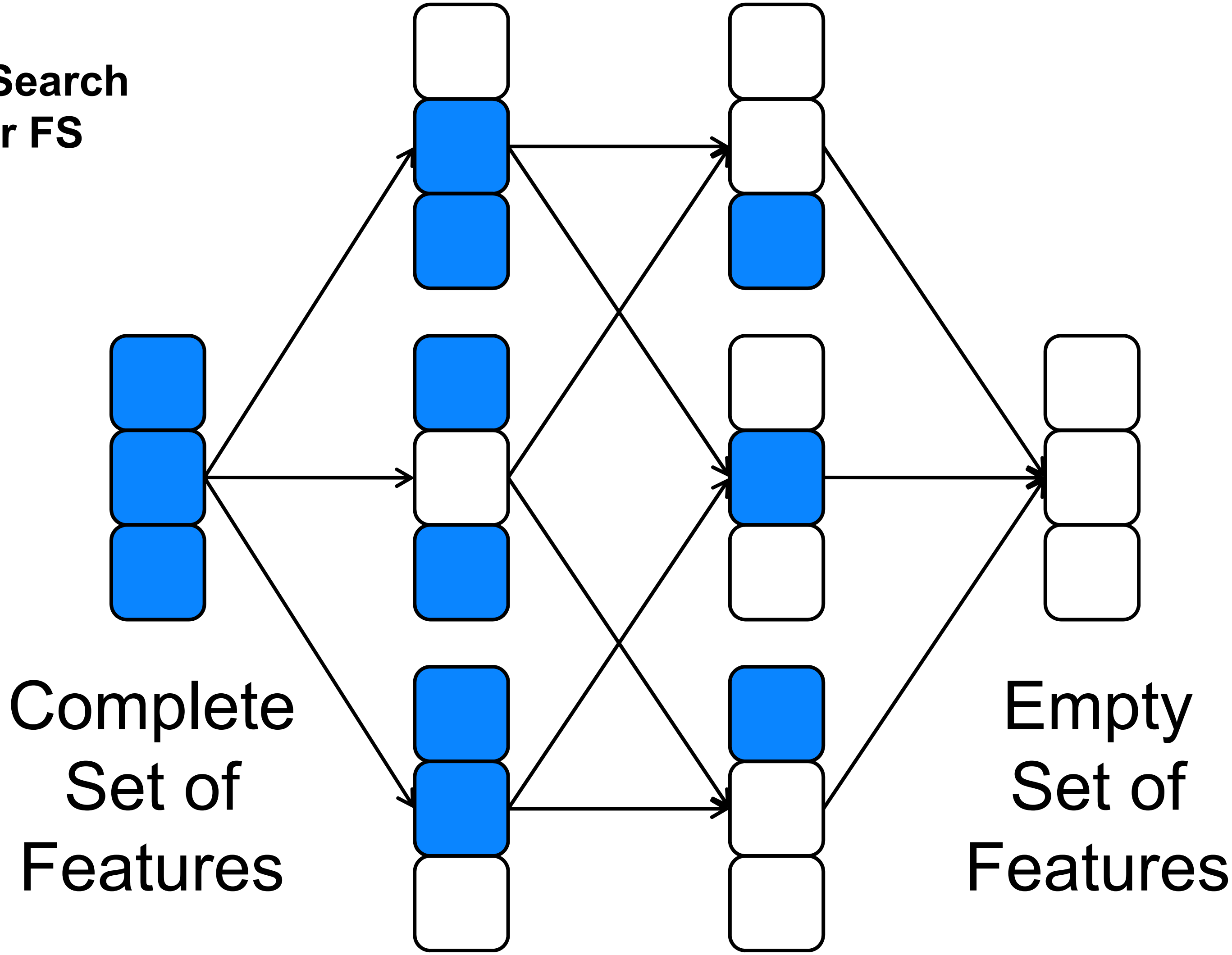
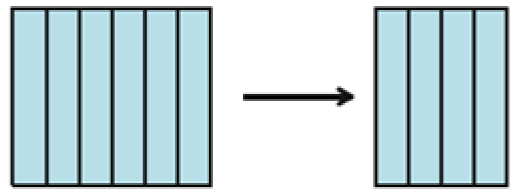


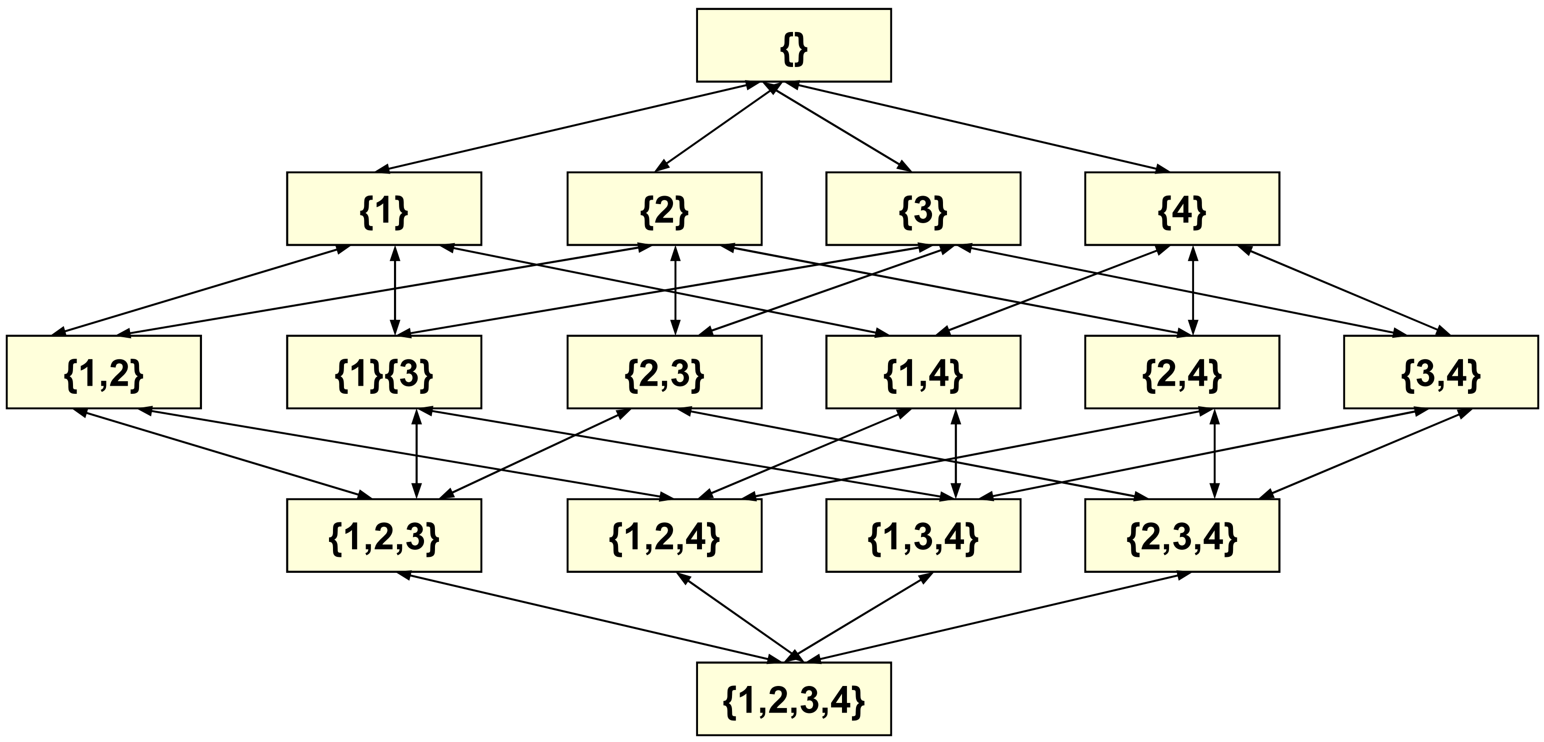
Fig. 7.1 Search space for FS

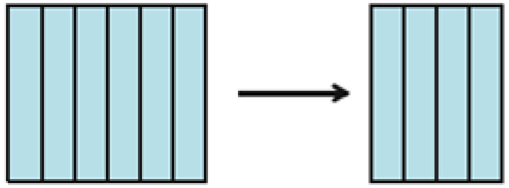


Feature Selection



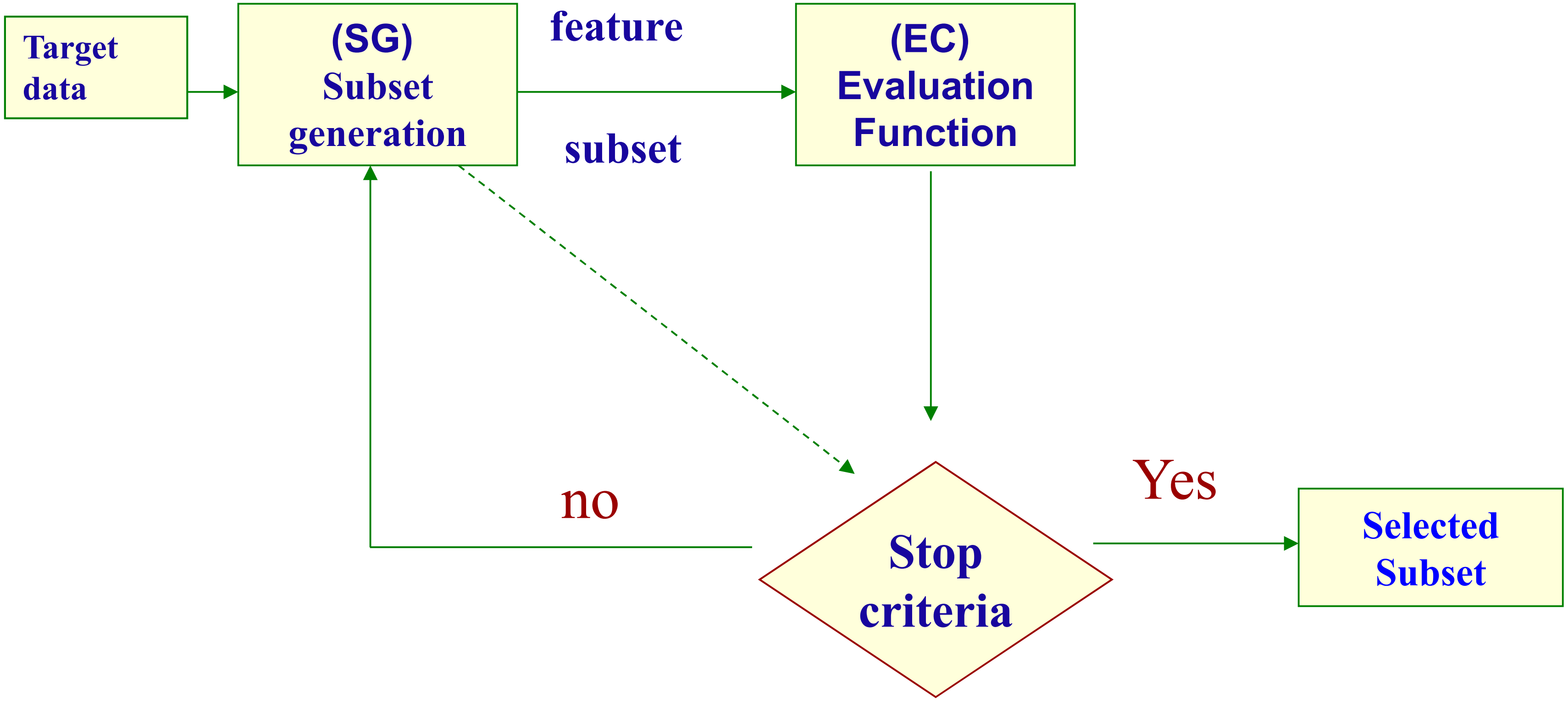
It can be considered as a search problem



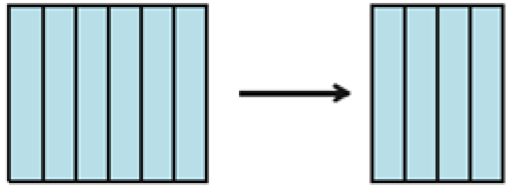


Feature Selection

Process



Feature Selection

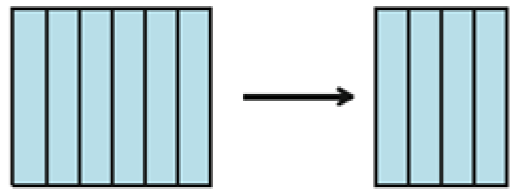


Goal functions: There are two different approaches

- **Filter.** The goal function evaluates the subsets basing on the information they contain. Measures of class separability, statistical dependences, information theory,… are used as the goal function.

- **Wrapper.** The goal function consists of applying the same learning technique that will be used later over the data resulted from the selection of the features. The returned value usually is the accuracy rate of the constructed classifier.

Feature Selection



Process

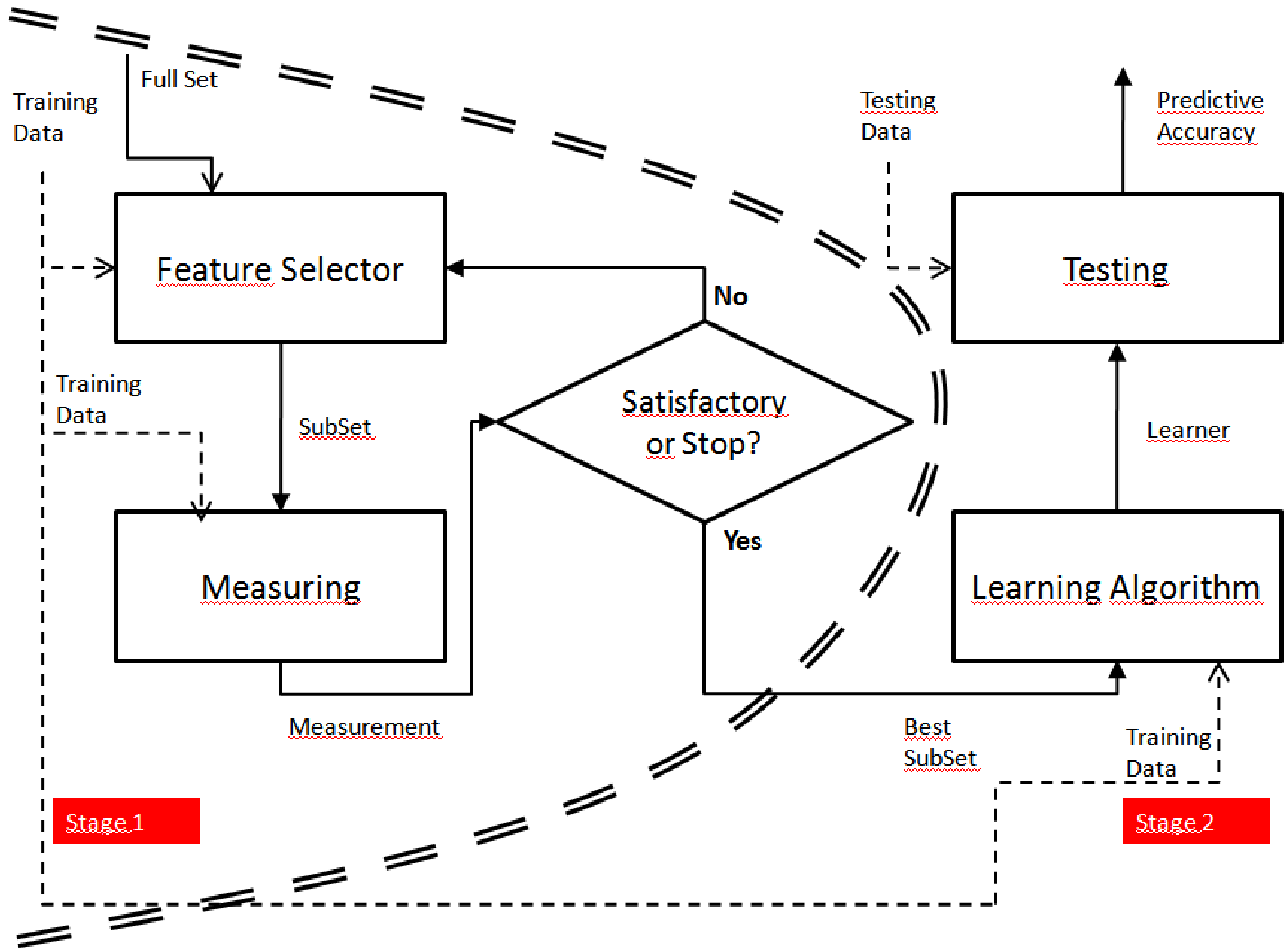
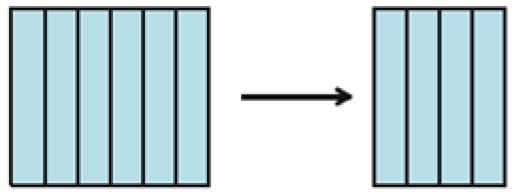


Fig. 7.2 A filter model for FS

Feature Selection



Filtering measures

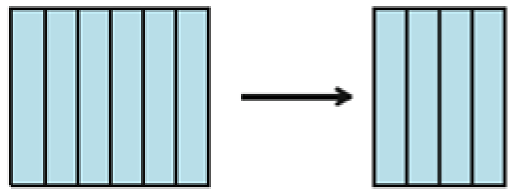
- **Separability measures.** They estimate the separability among classes: euclidean, Mahalanobis,…
  - I.e. In a two-class problem, a FS process based on this kind of measures determined that X is bettern than Y if X induces a greater difference than Y between the two prior conditional probabilities between the classes.

- **Correlation.** Good subset will be those correlated with the class variable

$$f(X_1,\ldots,X_M)=\frac{\sum_{i=1}^{M}\rho_{ic}}{\sum_{i=1}^{M}\sum_{j=i+1}^{M}\rho_{ij}}$$

where ρ_{ic} is the coefficient of correlation between the variable X_i and the label c of the class (C) and ρ_{ij} is the correlation coefficient between X_i and X_j

Feature Selection



- **Information theory based measures**
  - Correlation only can estimate lineal dependences. A more powerful method is the mutual information $I(X_{1,\ldots,M}; C)$

$$f(X_{1,\ldots,M}) = I(X_{1,\ldots,M}; C) = H(C) - H(C | X_{1,\ldots,M}) =$$
$$\sum_{c=1}^{|C|} \int_{X_{1,\ldots,M}} P(X_{1\ldots M}, \omega_c) \log \frac{P(X_{1\ldots M}, \omega_c)}{P(X_{1\ldots M}) P(\omega_c)} dx$$

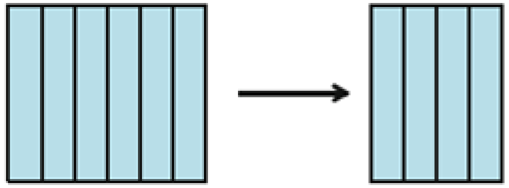
where H represents the entropy and ω_c the c-th label of the class C

  - Mutual information measures the quantity of uncertainty that decreases in the class C when the values of the vector $X_{1\ldots M}$ are known.
  - Due to the complexity of the computation of I, it is usual to use heurisctics rules

$$f(X_{1\ldots M}) = \sum_{i=1}^{M} I(X_i; C) - \beta \sum_{i=1}^{M} \sum_{j=i+1}^{M} I(X_i; X_j)$$

with β=0.5, as example.

Feature Selection

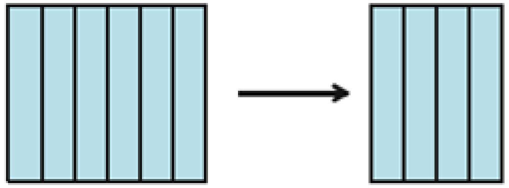


- **Consistency measures**

  - The three previous groups of measures try to find those features than could, maximally, predict the class better than the remain.
    - This approach cannot distinguish between two attributes that are equally appropriate, it does not detect redundant features.

  - Consistency measures try to find a minimum number of features that are able to separate the classes in the same way that the original data set does.

Feature Selection



Process

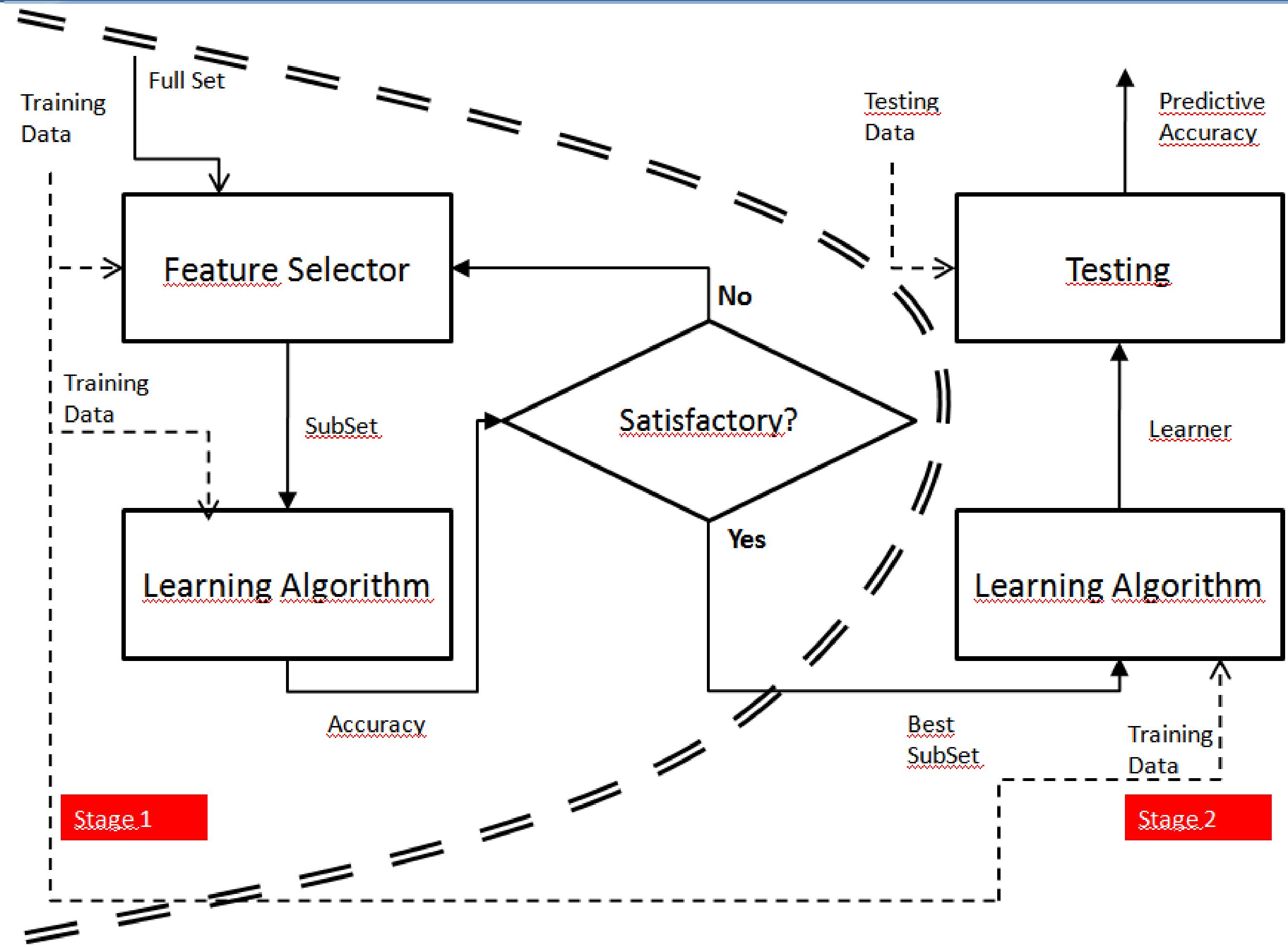
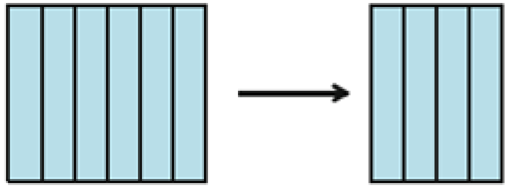


Fig. 7.2 A wrapper model for FS

Feature Selection



Process

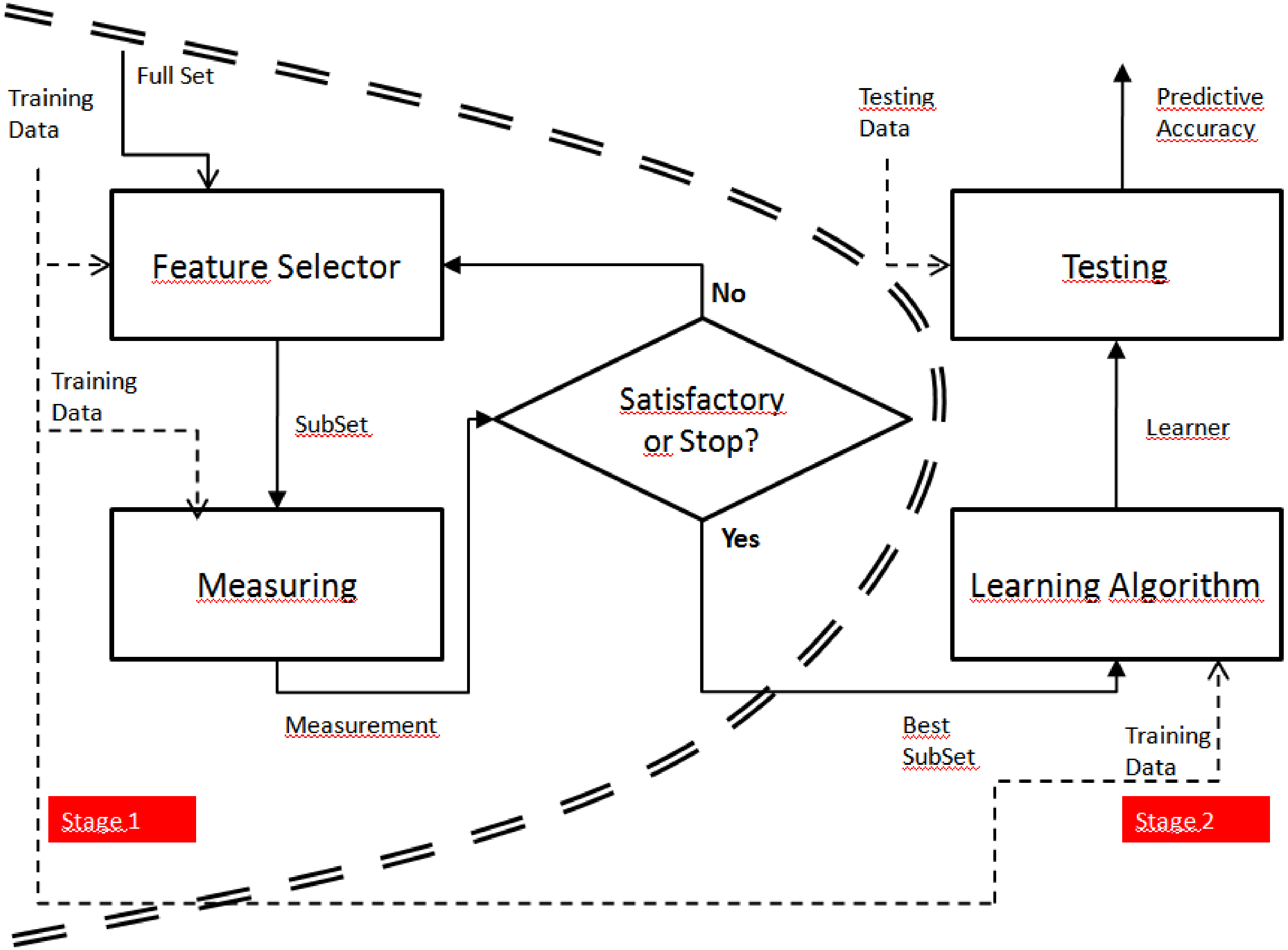
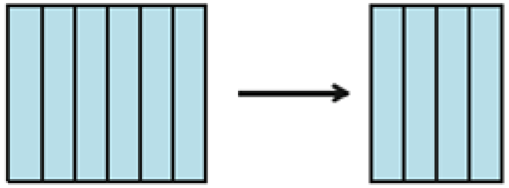


Fig. 7.2 A filter model for FS

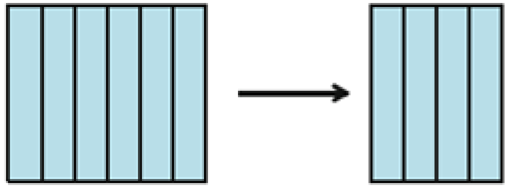
Feature Selection



Advantages

- **Wrappers:**
  - **Accuracy:** generally, they are more accurate than filters, due to the interaction between the classifier used in the goal function and the training data set.
  - **Generalization capability:** they pose capacity to avoid overfitting due to validation techniques employed.
- **Filters:**
  - **Fast:** They usually compute frequencies, much quicker than training a classifier.
  - **Generality:** Due to they evaluate instrinsic properties of the data and not their interaction with a classifier, they can be used in any problem.

Feature Selection

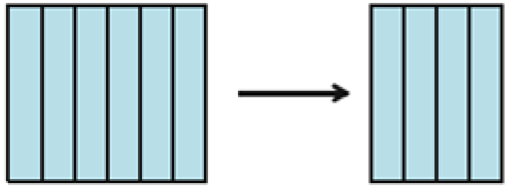


Drawbacks

- **Wrappers:**
  - **Very costly:** for each evaluation, it is required to learn and validate a model. It is prohibitive to complex classifiers.
  - **Ad-hoc solutions:** The solutions are skewed towards the used classifier.

- **Filters:**
  - **Trend to include many variables:** Normally, it is due to the fact that there are monotone features in the goal function used.
    - The use should set the threshold to stop.

Feature Selection



Categories

1. According to evaluation:

   filter

   wrapper

2. Class availability:

   Supervised

   Unsupervised

3. According to the search:

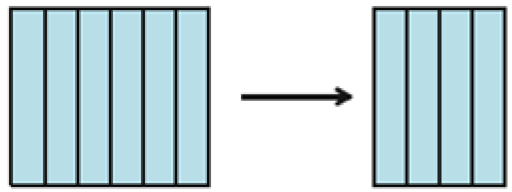
   Complete $O(2^N)$
   Heurístic $O(N^2)$
   Random ??

4. According to outcome:

   Ranking

   Subset of features

Feature Selection



Algorithms for getting subset of features

They return a subset of attributes optimized according to an evaluation criterion.

Input: x attributes – U evaluation criterion

Subset = {}

Repeat

    S_k = generateSubset(x)

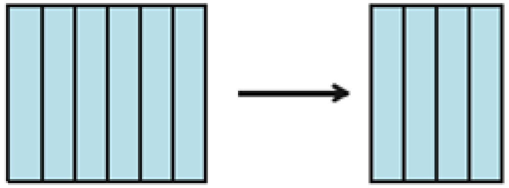
    if improvement(S, S_k,U)

        Subset = S_k

Until StopCriterion()

Output: List, of the most relevant atts.

Feature Selection



Ranking algorithms

They return a list of attributes sorted by an evaluation criterion.

Input: x attributed – U evaluation criterion

List = {}

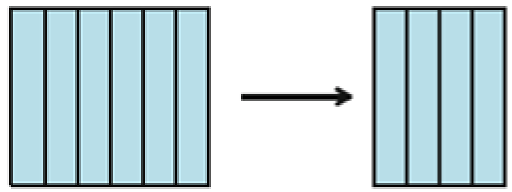
For each Attribute x_i, i $\in$ {1,...,N}

    v_i = compute(x_i,U)

    set x_i within the List according to v_i

Output: List, more relevant atts first

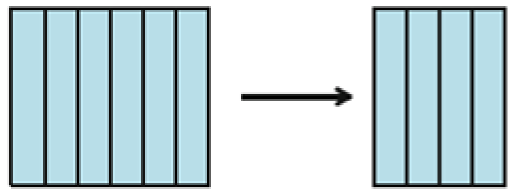
Feature Selection



Ranking algorithms

Attributes	A1	A2	A3	A4	A5	A6	A7	A8	A9
Ranking	A5	A7	A4	A3	A1	A8	A6	A2	A9
	A5	A7	A4	A3	A1	A8	(6 attributes)		

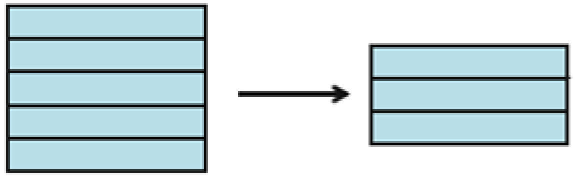
Feature Selection



Some relevant algorithms:

- Focus algorithm. Consistency measure for forward search
- Mutual Information based Features Selection (MIFS).
- mRMR: Minimum Redundancy Maximum Relevance
- Las Vegas Filter (LVF)
- Las Vegas Wrapper (LVW)
- Relief Algorithm

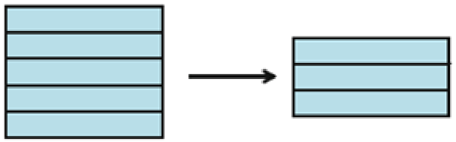
Instance Selection



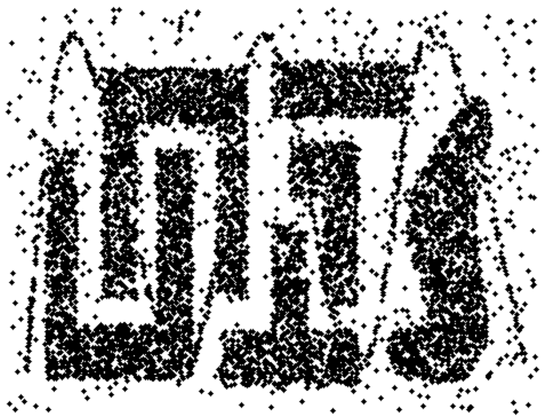
- Instance selection try to choose the examples which are relevant to an application, achieving the maximum performance. The outcome of IS would be:
  - Less data → algorithms learn quicker
  - Higher accuracy → the algorithm better generalizes
  - Simpler results → easier to understand them

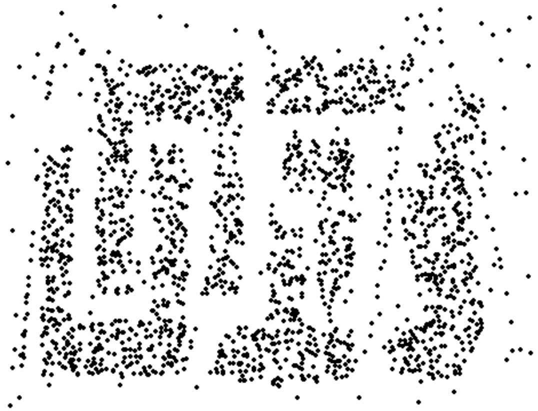
- **IS has as extension the generation of instances (prototype generation)**

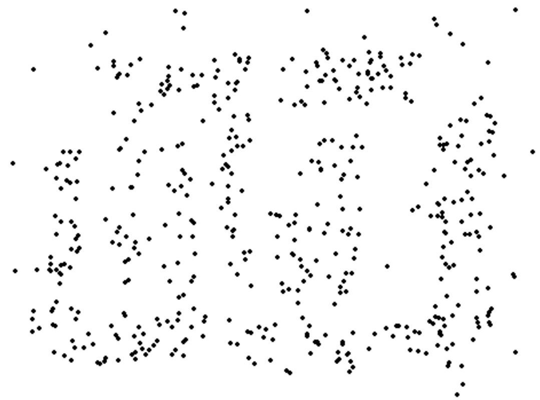
Instance Selection



Different size examples

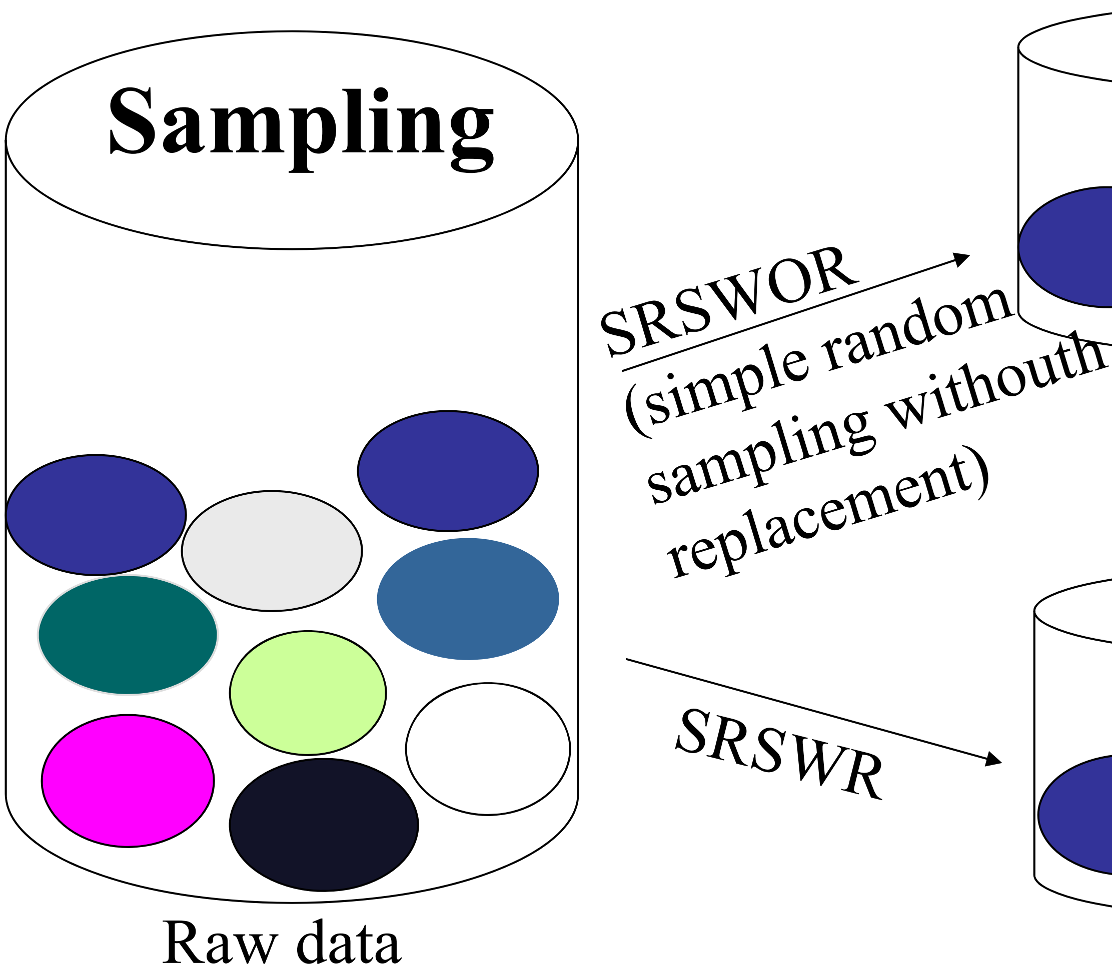




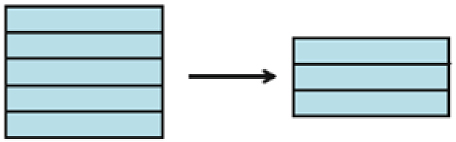


8000 points **2000 points** **500 points**

Instance Selection

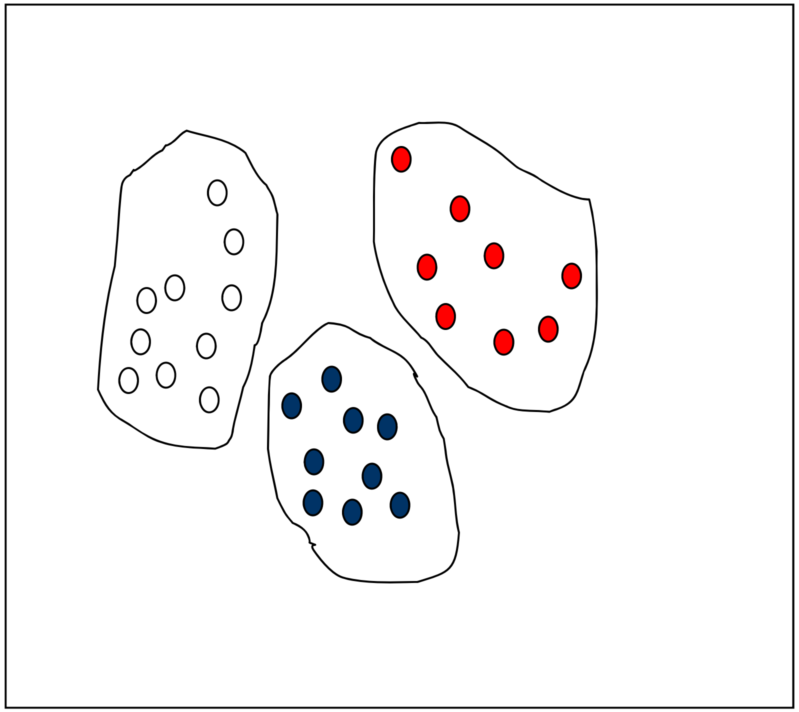


Instance Selection

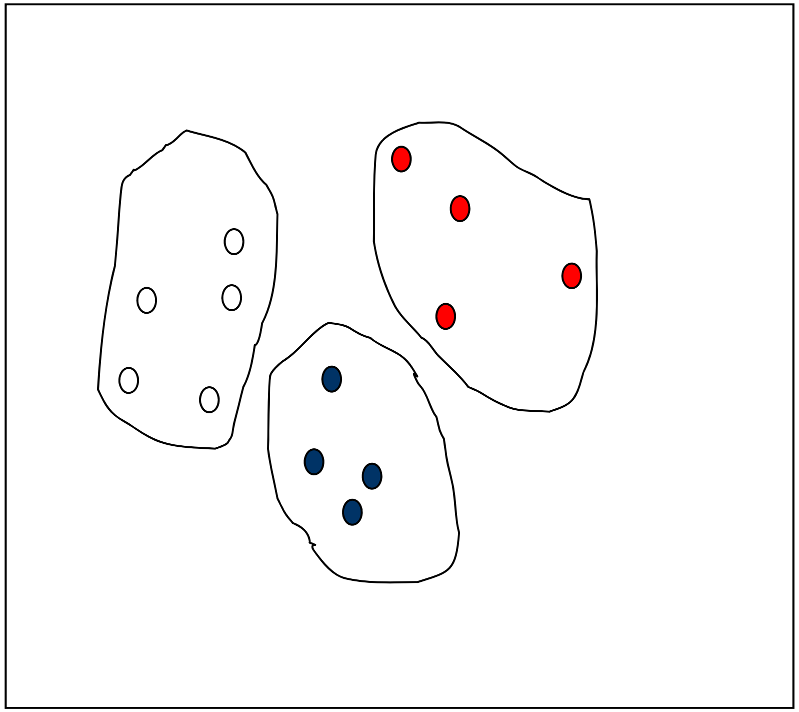


Sampling

Raw Data



Simple reduction



Instance Selection

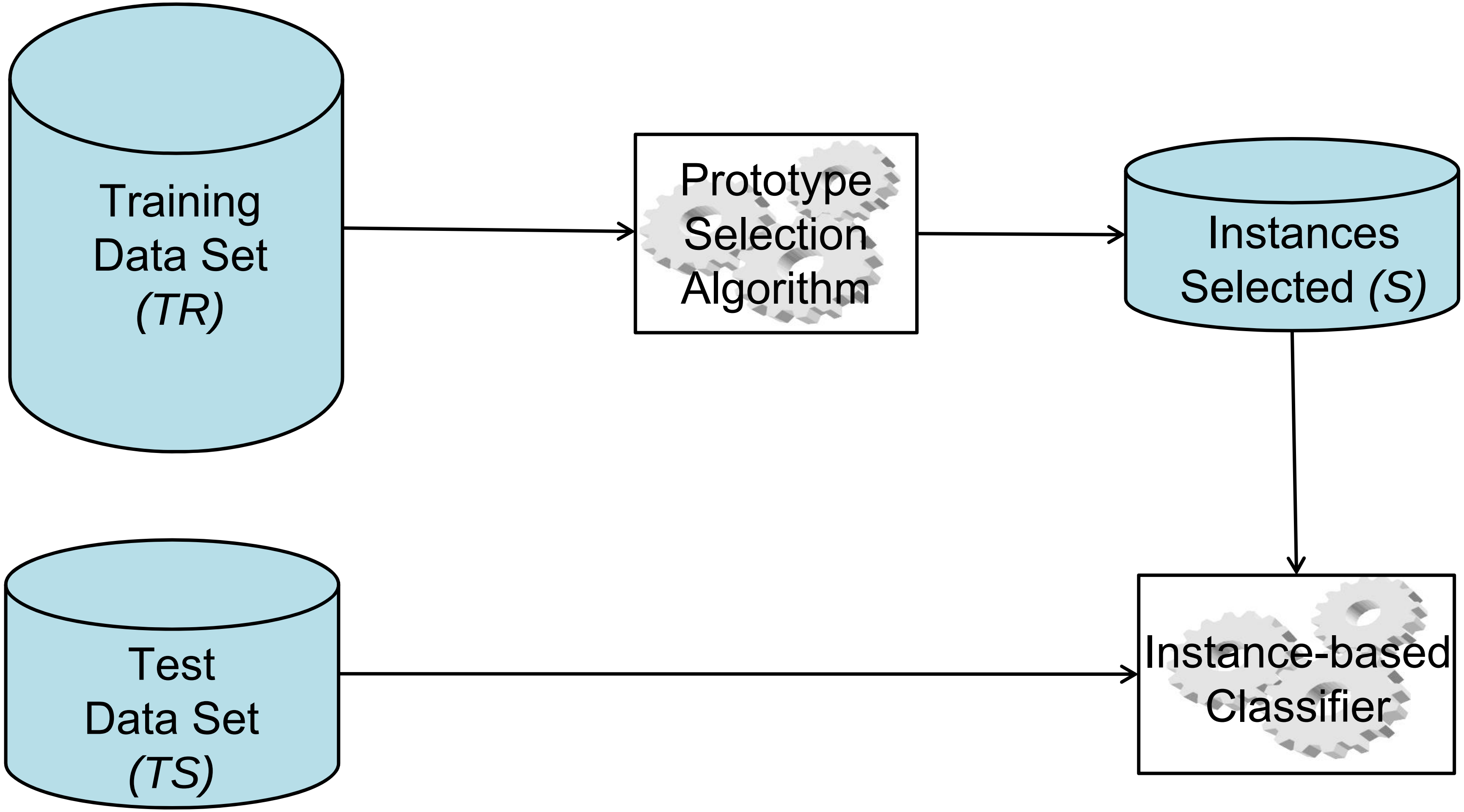
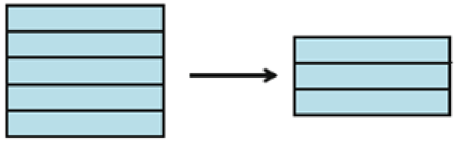


Fig. 8.1 PS process

Instance Selection



Prototype Selection (instance-based learning)

Properties:

- **Direction of the search:** Incremental, decremental, batch, hybrid or fixed.
- **Selection type:** Condensation, Edition, Hybrid.
- **Evaluation type:** Filter or wrapper.

Instance Selection

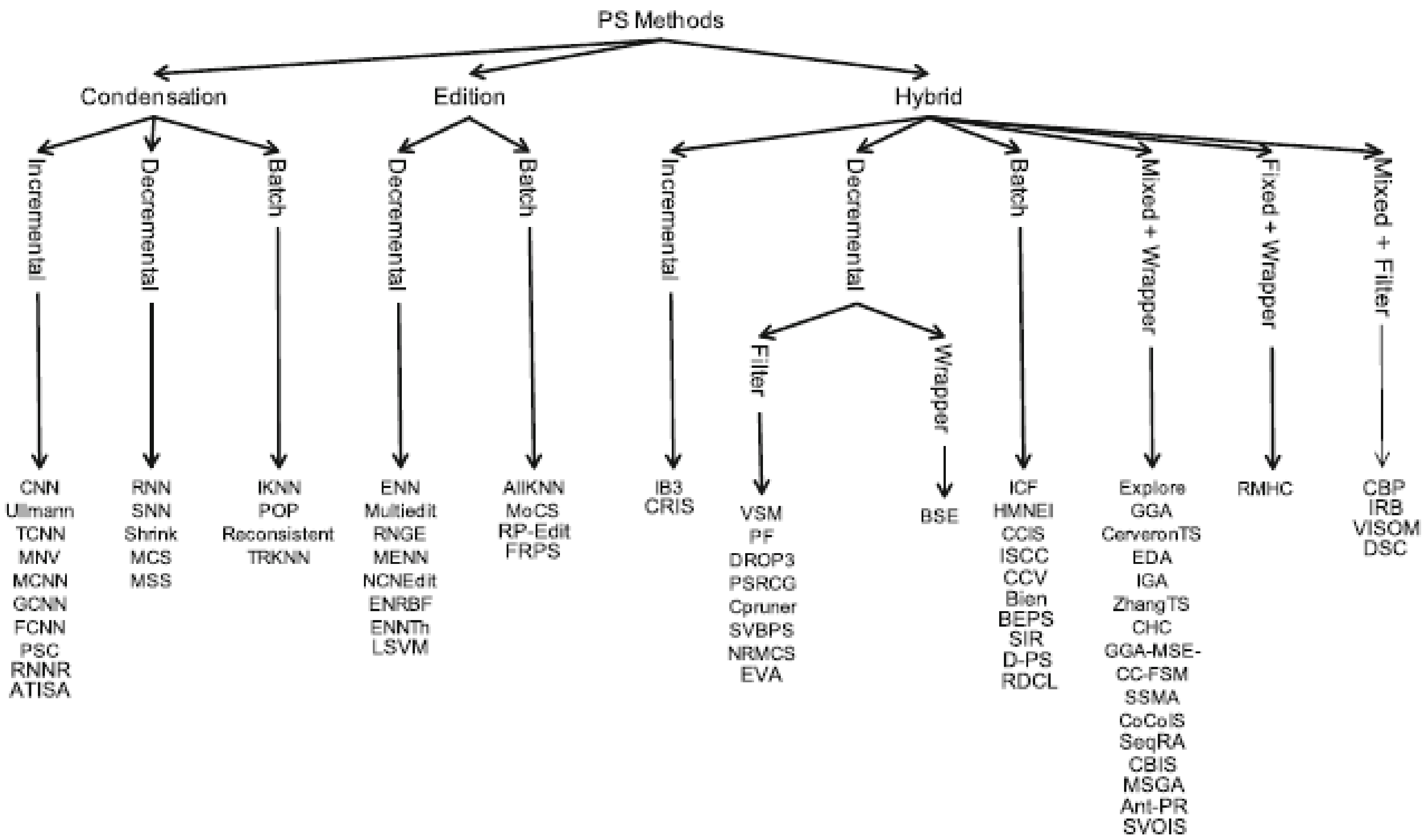
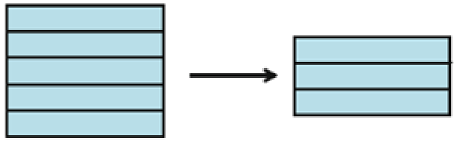


Fig. 8.3 PS taxonomy

Instance Selection



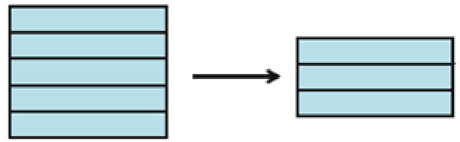
A pair of classical algorithms:

- Classical algorithm of condensation: Condensed Nearest Neighbor (CNN)
  - Incremental
  - It only inserts the misclassified instances in the new subsets.
  - Dependant on the order of presentation.
  - It only retains borderline examples.

Algorithm 10 CNN algorithm.

function CNN(T - training data)
    initialize: $S = \emptyset$
    repeat
        for all $x \in T$ (in random order) **do**
            Find $x' \in S$ s.t. $||x - x'|| = \min_{x^j \in S} ||x - x^j||$
            if $class(x) \neq class(x')$ **then**
                $S = S \cup \{x\}$
            end if
        end for
    until S does not change
    return S
end function

Instance Selection



A pair of classical algorithms:

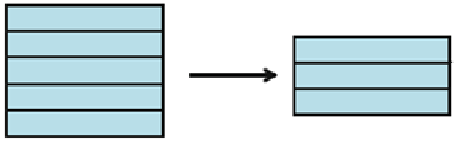
- Classical algorithm for Edition: Edited Nearest Neighbor (ENN)
  - Batch
  - It removes those instances which are wrongly classified by using a k-nearest neighbor scheme (k = 3, 5 or 9).
  - It “smooths” the borders among classes, but also retains the rest of points.

Algorithm 11 ENN algorithm.

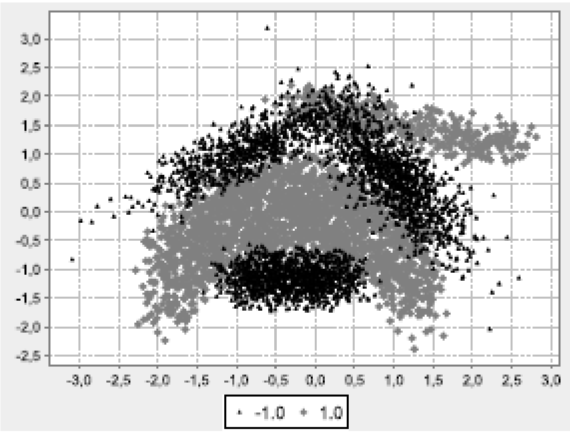
---

function ENN(T - training data, k - number of nearest neighbor)
    initialize: $S = T$
    for all $x \in S$ **do**
        $X' = \emptyset$
        for $i = 1$ to k **do**
            Find $x'_i \in T$ s.t. $x \neq x'_i$ and $||x - x'_i|| = \min_{x^j \in (T \setminus X')} ||x - x^j||$
            $X' = X' \cup \{x'_i\}$
        end for
        if $class(x) \neq majorityClass(X')$ **then**
            $S = S \setminus \{x\}$
        end if
    end for
    return S
end function

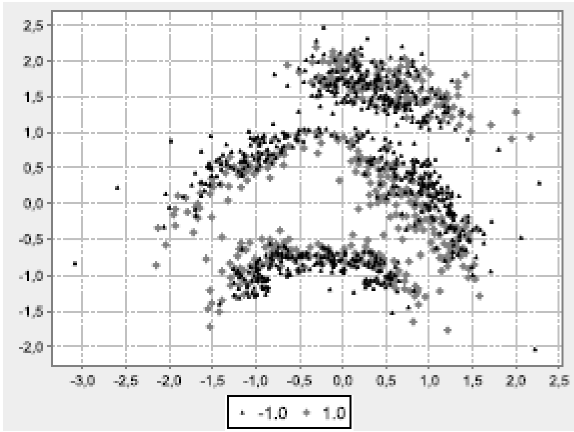
Instance Selection



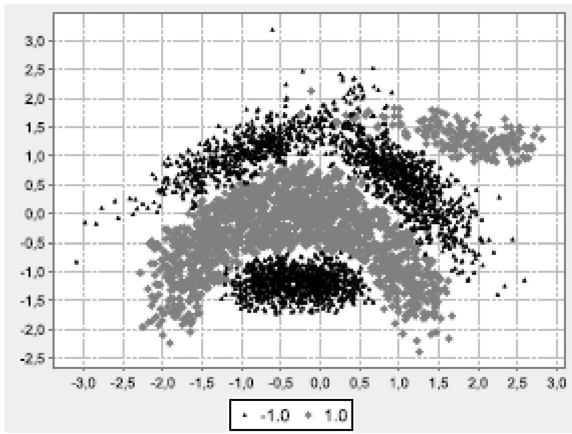
Graphical illustrations (Condensation vs Edition):



(a) Banana Original $(0.8751, 0.7476)$



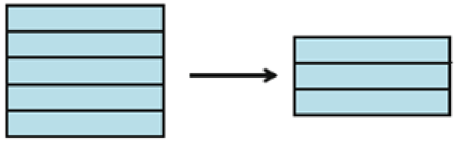
(b) CNN $(0.7729, 0.8664, 0.7304)$



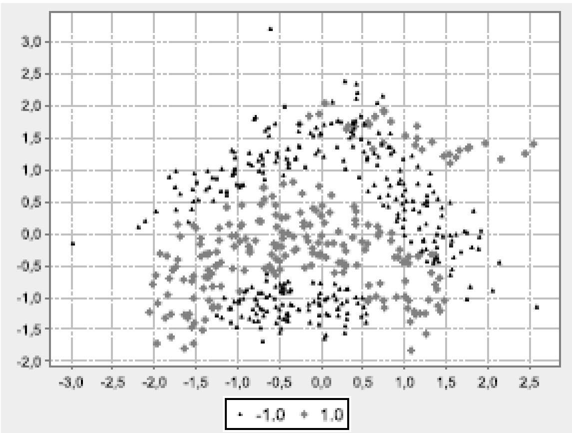
(h) AllKNN $(0.1758, 0.8934, 0.7831)$

Banana data set with 5,300 instances and two classes. Obtained subset with CNN and AllKNN (iterative application of ENN with k=3, 5 y 7).

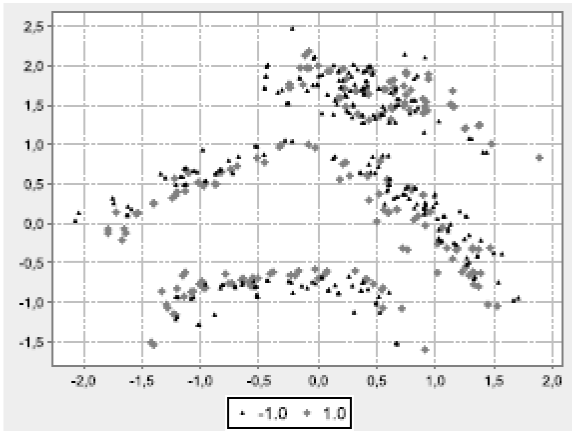
Instance Selection



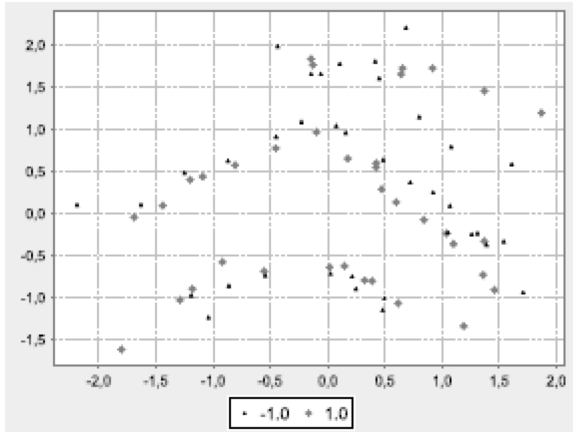
Graphical illustrations:



(k) RMHC $(0.9000, 0.8972, 0.7915)$



(e) DROP3 $(0.9151, 0.8696, 0.7356)$



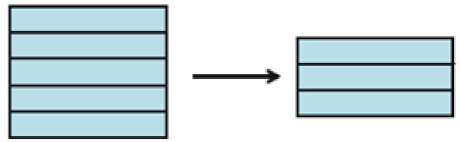
(l) SSMA $(0.9879, 0.8964, 0.7900)$

RMHC is an adaptive sampling technique based on local search with a fixed final rate of retention.

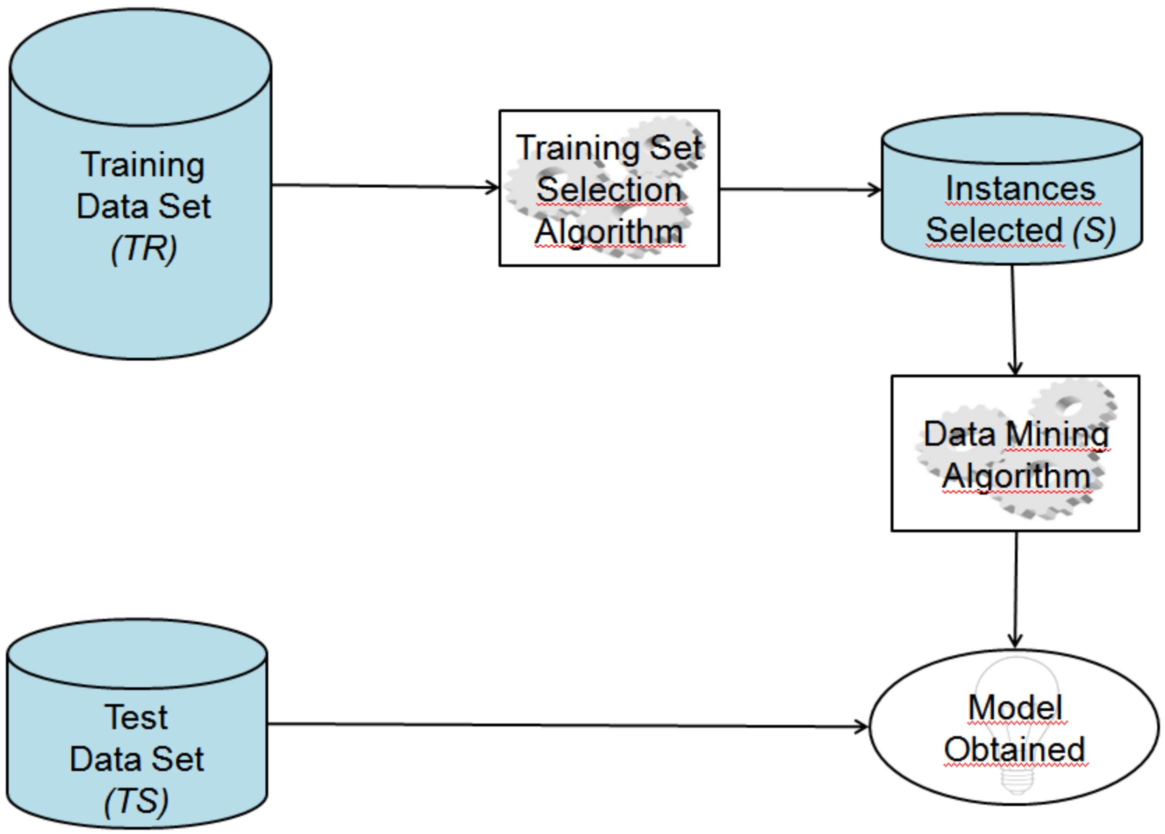
DROP3 is the most-known hybrid technique very use for kNN.

SSMA is an evolutionary approach based on memetic algorithms..

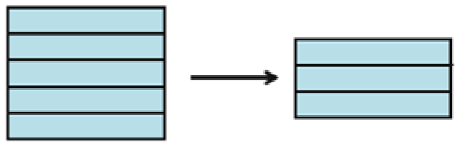
Instance Selection



Training Set Selection



Example Instance Selection and Decision Tree modeling

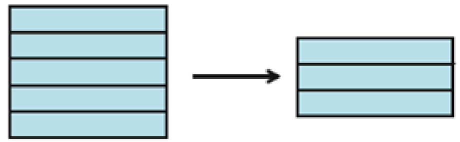


Kdd Cup'99. Strata Number: 100

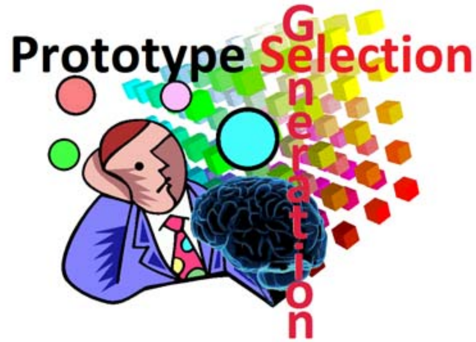
	No. Rules	% Reduction	C4.5	
			%Ac Trn	%Ac Test
C4.5	252		99.97%	99.94%
Cnn Strat	83	81.61%	98.48%	96.43%
Drop1 Strat	3	99.97%	38.63%	34.97%
Drop2 Strat	82	76.66%	81.40%	76.58%
Drop3 Strat	49	56.74%	77.02%	75.38%
Ib2 Strat	48	82.01%	95.81%	95.05%
Ib3 Strat	74	78.92%	99.13%	96.77%
Icf Strat	68	23.62%	99.98%	99.53%
CHC Strat	**9**	**99.68%**	98.97%	**97.53%**

Bibliography: J.R. Cano, F. Herrera, M. Lozano, **Evolutionary Stratified Training Set Selection for Extracting Classification Rules with Trade-off Precision-Interpretability**. *Data and Knowledge Engineering 60 (2007) 90-108, doi:10.1016/j.datak.2006.01.008*.

Instance Selection



WEBSITE:
http://sci2s.ugr.es/pr/index.php



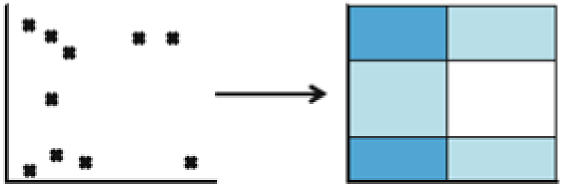
Bibliography:

S. García, J. Derrac, J.R. Cano and F. Herrera,
Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study.
IEEE Transactions on Pattern Analysis and Machine Intelligence 34:3 (2012) 417-435 doi: 10.1109/TPAMI.2011.142

S. García, J. Luengo, F. Herrera. **Data Preprocessing in Data Mining**, Springer, 15, 2015

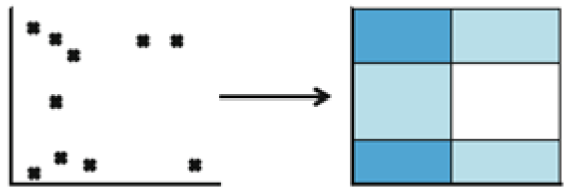
Source Codes (Java):

Discretization



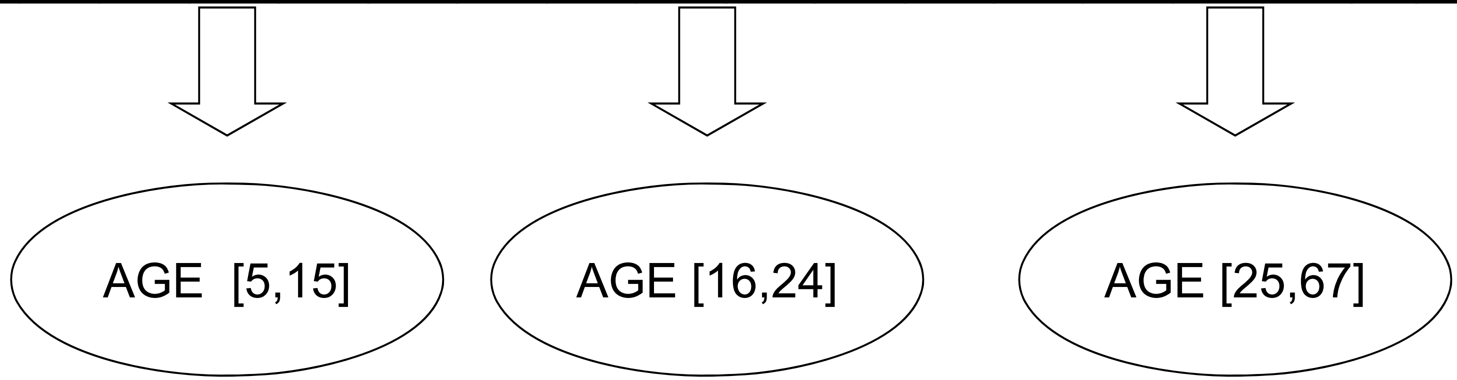
- Discrete values are very useful in Data Mining.
- They represent more concise information, they are easier to understand and closer to the representation of knowledge.
- The discretization is focused on the transformation of continuous values with an order among in nominal/categorical values without ordering. It is also a quantification of numerical attributes.
- Nominal values are within a finite domain, so they are also considered as a data reduction technique.

Discretization

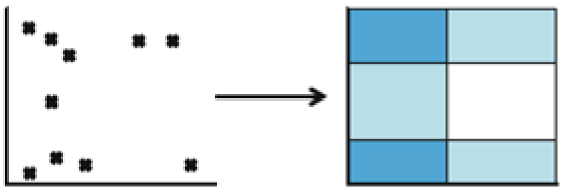


- Divide the range of numerical (continuous or not) attributes into intervals.
- Store the labels of the intervals.
- Is crucial for association rules and some classification algorithms, which only accept discrete data.

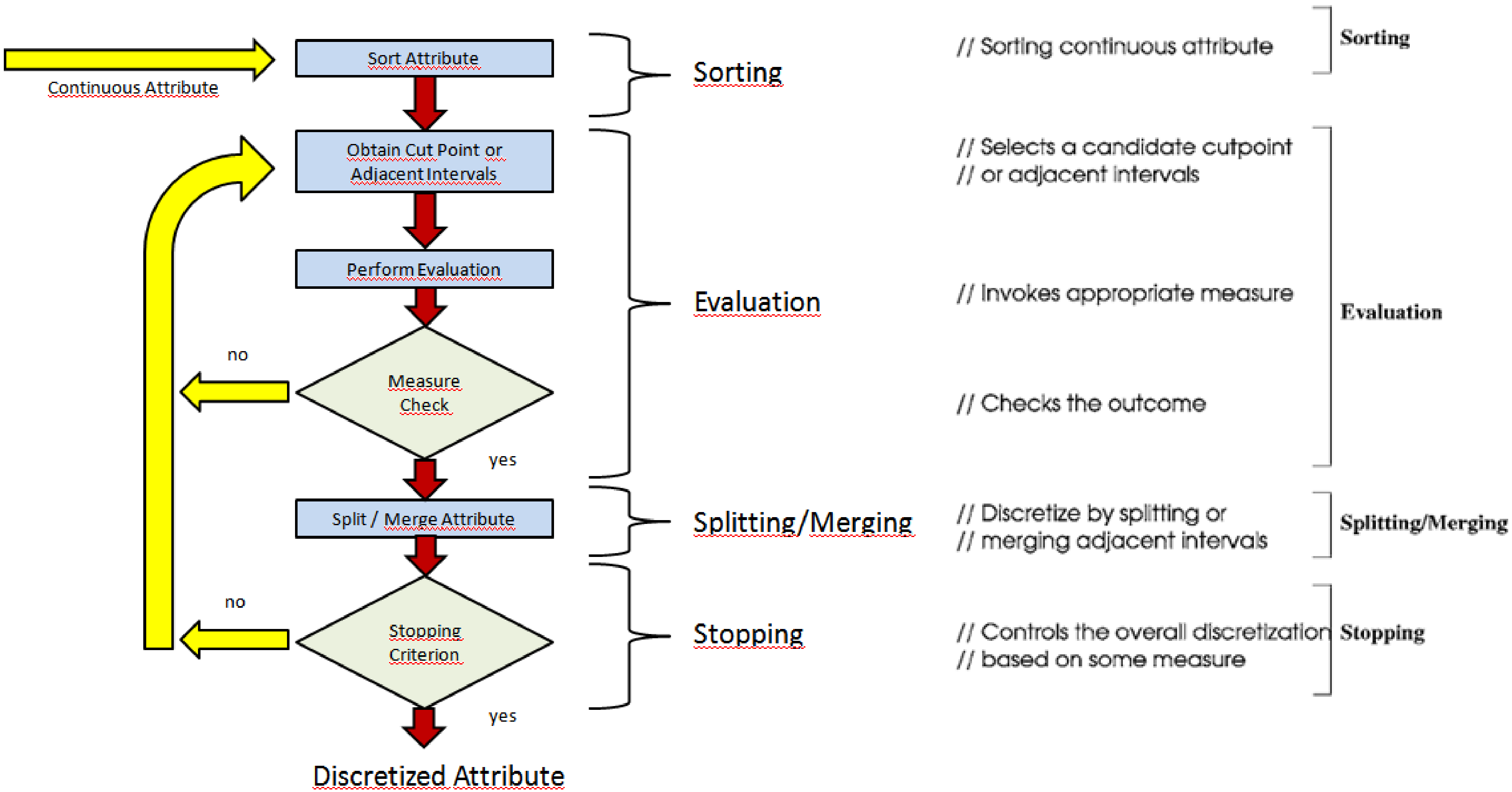
Age	5	6	6	9	...	15	16	16	17	20	...	24	25	41	50	65	...	67
Owner of a Car	0	0	0	0	...	0	1	0	1	1	...	0	1	1	1	1	...	1



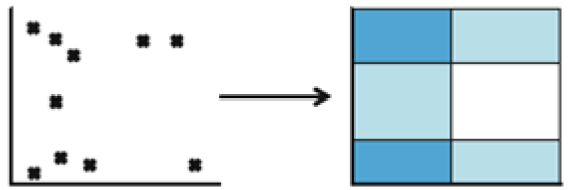
Discretization



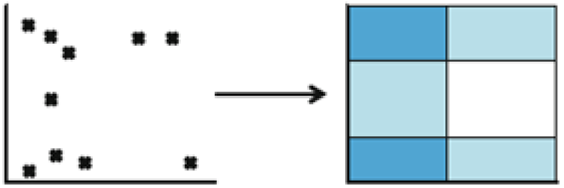
Stages in the discretization process



Discretization



- **Discretization has been developed in several lines according to the neccesities:**
- Supervised vs. unsupervised: Whether or not they consider the objective (class) attributes.
- Dinamical vs. Static: Simultaneously when the model is built or not.
- Local vs. Global: Whether they consider a subset of the instances or all of them.
- Top-down vs. Bottom-up: Whether they start with an empty list of cut points (adding new ones) or with all the possible cut points (merging them).
- Direct vs. Incremental: They make decisions all together or one by one.



Discretization

- **Unsupervised algorithms:**
  - Equal width
  - Equal frequency
  - Clustering …..

- Supervised algorithms:

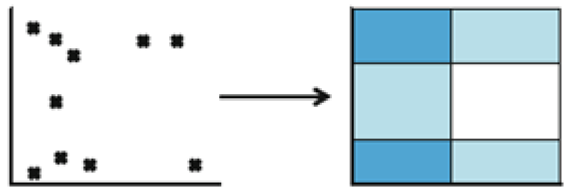
  - Entropy based [Fayyad & Irani 93 and others]

    [Fayyad & Irani 93] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Proc. 13th Int. Joint Conf. AI (IJCAI-93)*, 1022-1027. Chamberry, France, Aug./ Sep. 1993.
  - Chi-square [Kerber 92]

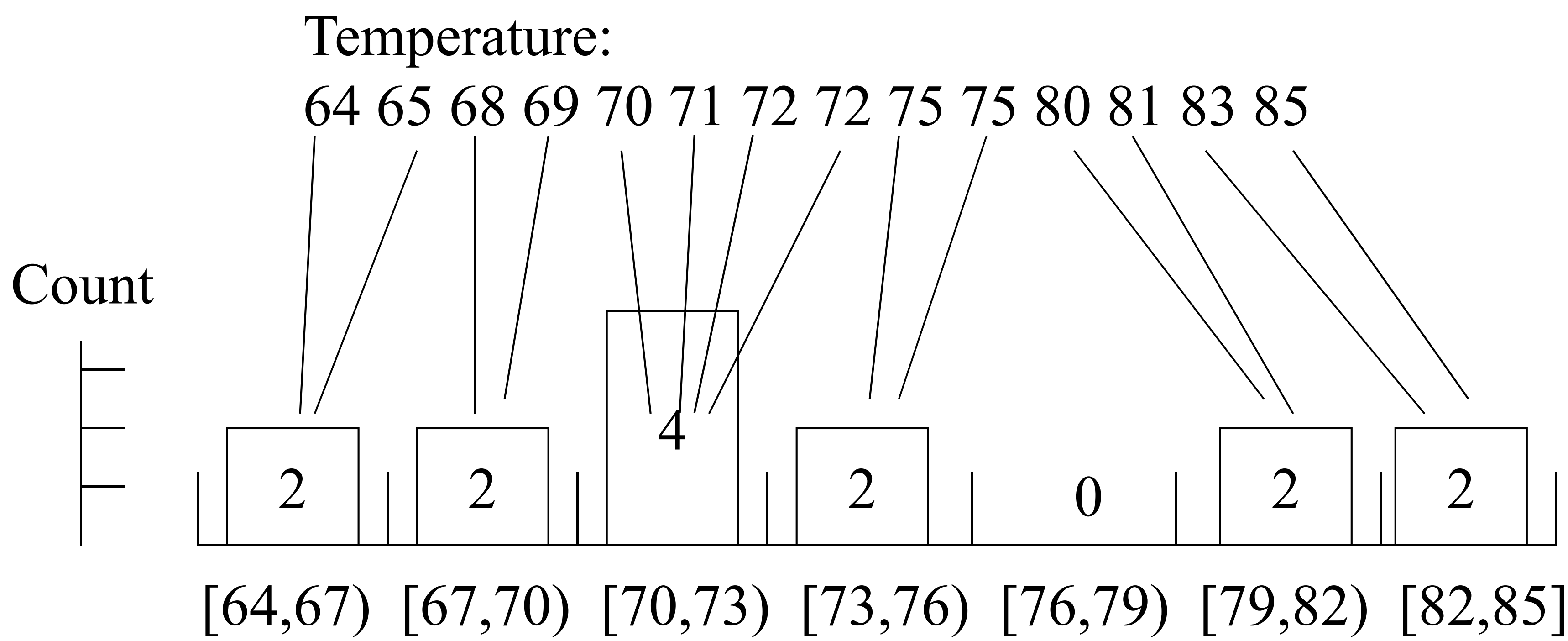
    [Kerber 92] R. Kerber. ChiMerge: Discretization of numeric attributes. *Proc. 10th Nat. Conf. AAAI,* 123-128. 1992.
  - … (lots of proposals)

Bibliography: **S. García, J. Luengo, José A. Sáez, V. López, F. Herrera, A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning**.
IEEE Transactions on Knowledge and Data Engineering 25:4 (2013) 734-750, *doi: 10.1109/TKDE.2012.35*.

Discretization

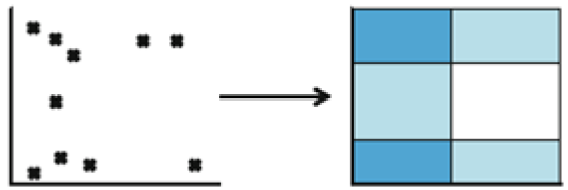


Example Discretization: Equal width



Equal width

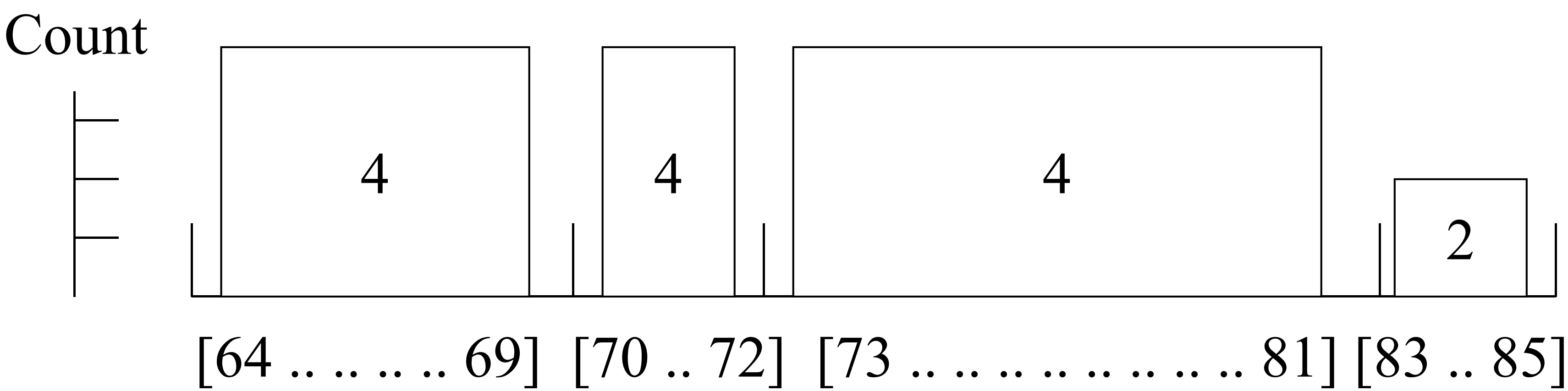
Discretization



Example discretization: Equal frequency

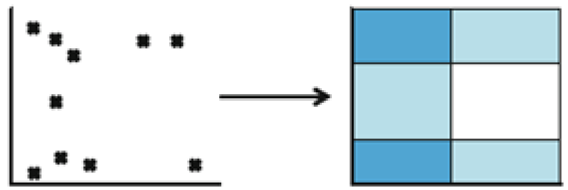
Temperature

64 65 68 69 70 71 72 72 75 75 80 81 83 85



Equal frequency (height) = 4, except for the last box

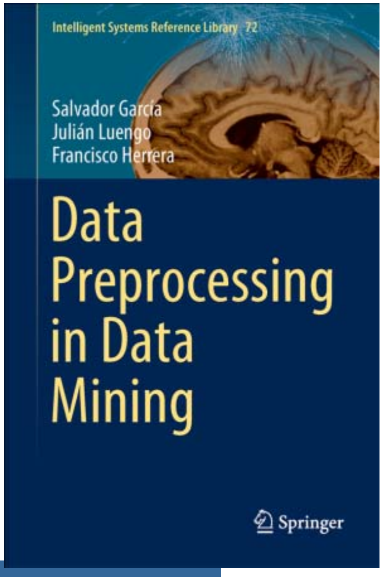
Discretization



- *Which discretizer will be the best?.*

- As usual, it will depend on the application, user requirements, etc.
- Evaluation ways:
  - Total number of intervals
  - Number of inconsistencies
  - Predictive accuracy rate of classifiers

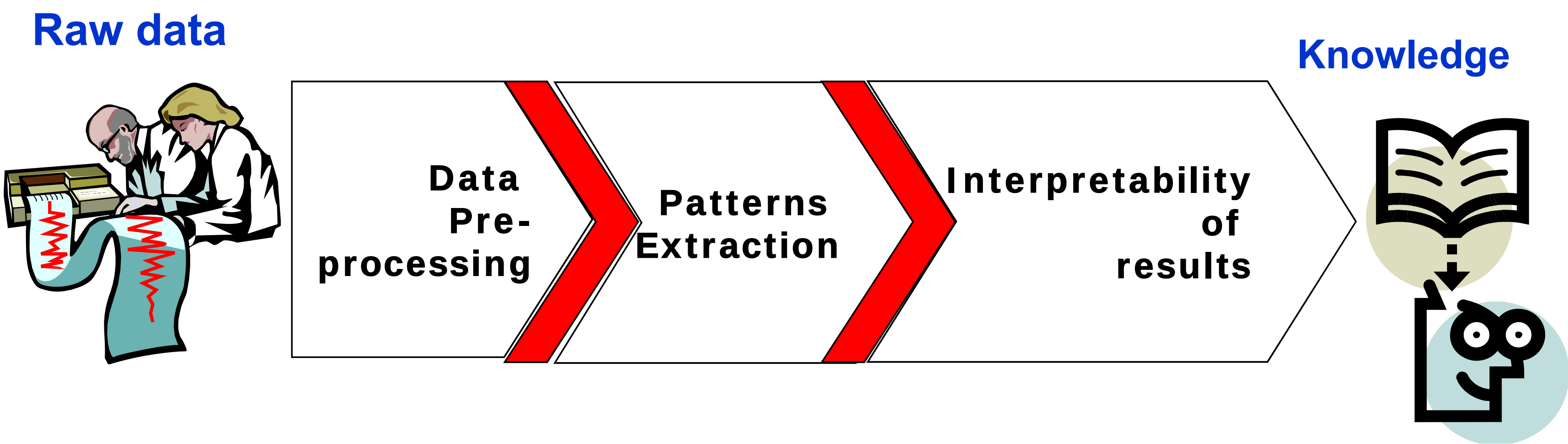
Data Preprocessing



1. Introduction. Data Preprocessing
2. Integration, Cleaning and Transformations
3. Imperfect Data
4. Data Reduction
5. Final Remarks

Final Remarks

Data preprocessing is a necessity when we work with real applications.



Final Remarks

Real data could be dirty and could drive to the extraction of useless patterns/rules.

Data preprocessing can generate a smaller data set than the original, which allows us to improve the efficiency in the Data Mining process.

No quality data, no quality mining results!

Quality decisions must be based on quality data!

Final Remarks

Data Preprocessing Advantage: Data preprocessing allows us to apply Learning/Data Mining algorithms easier and quicker, obtaining more quality models/patterns in terms of accuracy and/or interpretability.

Final Remarks

Data Preprocessing Advantage: Data preprocessing allows us to apply Learning/Data Mining algorithms easier and quicker, obtaining more quality models/patterns in terms of accuracy and/or interpretability.

A drawback: Data preprocessing is not a structured area with a specific methodology for understanding the suitability of preprocessing algorithms for managing a new problems.

Every problem can need a different preprocessing process, using different tools.

The design of automatic processes of use of the different stages/techniques is one of the data mining challenges.

Final Remarks

Website including slides, material, links …

(under preparation)

http://sci2s.ugr.es/books/data-preprocessing

Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big Data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Preprocessing for Big Data analytics

Tasks to discuss:

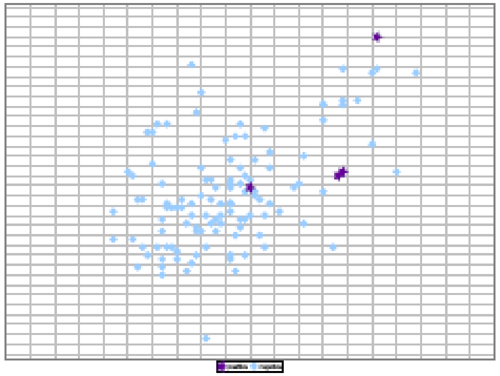
1. **Scalability of the proposals (Algorithms redesign!!)**

2. **Reduce phase: How must we combine the output of the maps? (Fundamental phase to use MapReduce for Big Data Preprocessing!!)**

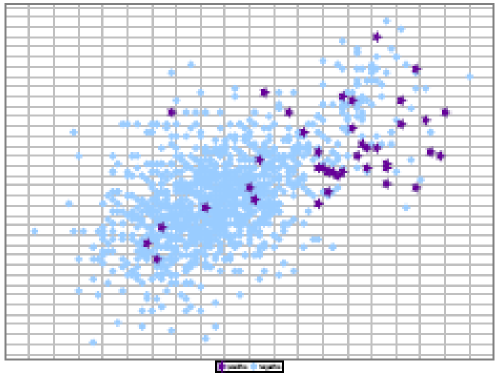
1. **Appearance of small disjuncts with the MapReduce data fragmentation.**
   This problem is basically associated to imbalanced classification: Lack of Data/lack of density

Appearance of small disjuncts with the MapReduce data fragmentation

Density: Lack of data



(a) 10 % of training instances



(b) 100 % of training instances

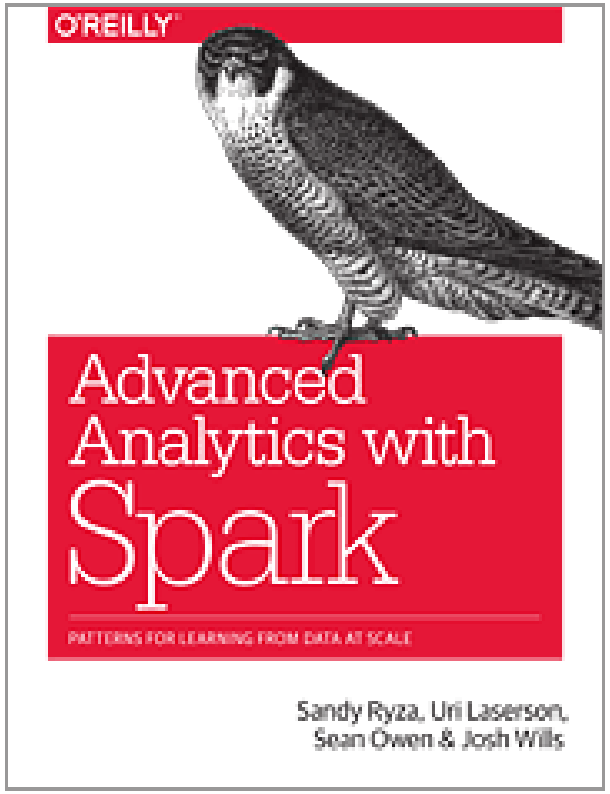
Figure 11: Lack of density or small sample size on the yeast4 dataset

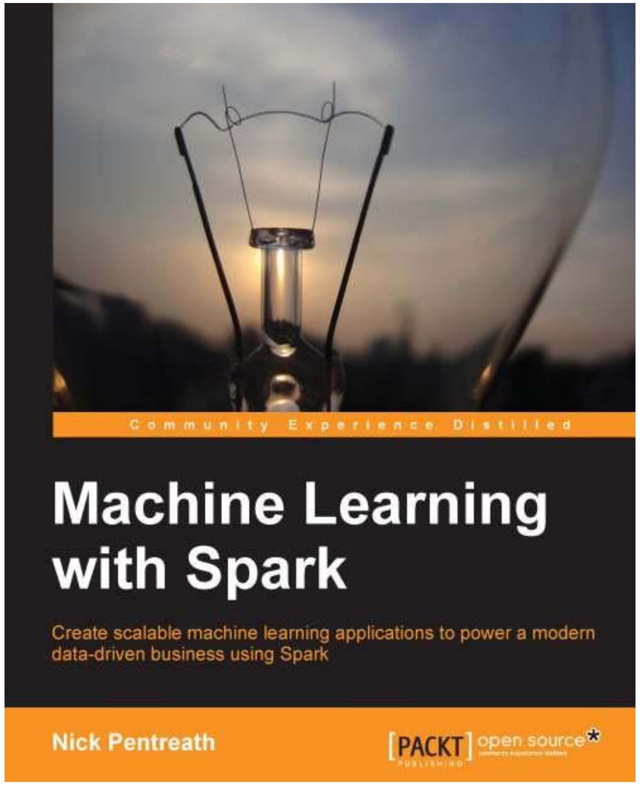
The lack of density in the training data may also cause the introduction of small disjuncts.

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.

Big Data Preprocessing

Bird's eye view



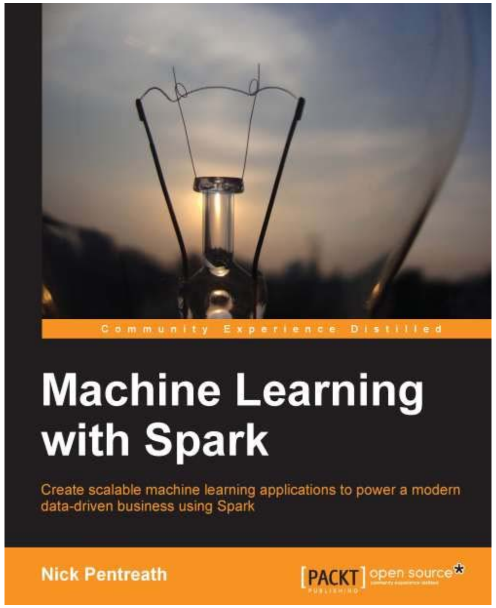


9 cases of study

10 chapters giving a quick glance on Machine Learning with Spakr

Big Data Preprocessing

Bird's eye view



A short introduction to data preparation with Spark – Chapter 3

Chapter 3: Obtaining, Processing, and Preparing Data with Spark

Big Data Preprocessing

Bird's eye view https://spark.apache.org/docs/latest/mllib-guide.html

MLlib - Feature Extraction and Transformation

Spark MLlib

- 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)

ChiSqSelector

ChiSqSelector stands for Chi-Squared feature selection. It operates on labeled data with categorical features. ChiSqSelector orders features based on a Chi-Squared test of independence from the class, and then filters (selects) the top features which are most closely related to the label.

Model Fitting

ChiSqSelector has the following parameter in the constructor:

- numTopFeatures number of top features that the selector will select (filter).

Big Data Preprocessing

Bird's eye view http://sci2s.ugr.es/BigData

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

The web is organized according to the following summary:

Throughout this Website, we have also included the **source code for the algorithms** associated with the former papers, as well as **new approaches that are under development.** Readers may find the implementations in the corresponding *Github and Spark Packages links* placed in those sections devoted to describe each framework. Both are marked with the corresponding logo:

Big Data Preprocessing

Bird's eye view http://sci2s.ugr.es/BigData

Home » Thematic Web 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 Spark
3. Big Data preprocessing
4. Imbalanced Big Data classification
5. Big Data classification with fuzzy models
6. Dataset Repository
7. Literature review: surveys and overviews
8. Keynote slides
9. Links of interest

Big Data Preprocessing

1. Introduction to Data Preprocessing
2. Feature Selection
3. Feature Weighting
4. Discretization
5. Prototype Generation

Feature Selection

MapReduce based Evolutionary Feature Selection

GitHub triguero / **MR-EFS**

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

This repository includes the MapReduce implementations used in [1]. This implementation is based on Apache Mahout 0.8 library. The Apache Mahout (http://mahout.apache.org/) project's goal is to build an environment for quickly creating scalable performant machine learning applications.

[1] 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, In press, 2015.

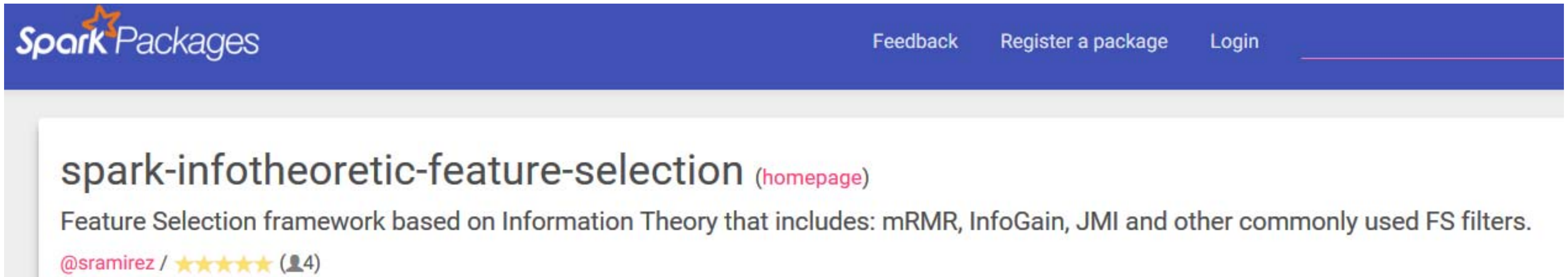
Feature Selection

An Information Theoretic Feature Selection Framework for Spark

GitHub sramirez / **spark-infotheoretic-feature-selection**

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

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



Feature Selection

Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm

GitHub sramirez / fast-mRMR

https://github.com/sramirez/fast-mRMR

An Information Theoretic Feature Selection Framework for Big Data under Apache Spark

Sergio Ramírez-Gallego, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera

Fast-mRMR

This is an improved implementation of the classical feature selection method: minimum Redundancy and Maximum Relevance (mRMR); presented by Peng in (*Hanchuan Peng, Fuhui Long, and Chris Ding "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 8, pp.1226-1238, 2005*).
This includes several optimizations such as: cache marginal probabilities, accumulation of redundancy (greedy approach) and a data-access by columns.

Feature Weighting

GitHub triguero / **ROSEFW-RF**

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

This project contains the code used in the ROSEFW-RF algorithm, including:

Evolutionary Feature Weighting
RandomForest
Random Oversampling

I. Triguero, S. 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, in press. doi: 10.1016/j.knosys.2015.05.027

Feature weighting is a feature importance ranking technique where weights, not only ranks, are obtained. When successfully applied relevant features are attributed a high weight value, whereas irrelevant features are given a weight value close to zero.
Feature weighting can be used not only to improve classification accuracy but also to discard features with weights below a certain threshold value and thereby increase the resource efficiency of the classifier.

Prototype Generation

MapReduce based Prototype Reduction

GitHub triguero / **MRPR**

https://github.com/triguero/MRPR

This repository includes the MapReduce implementation proposed for Prototype Reduction for the algorithm MRPR.

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. doi: 10.1016/j.neucom.2014.04.078

Discretization

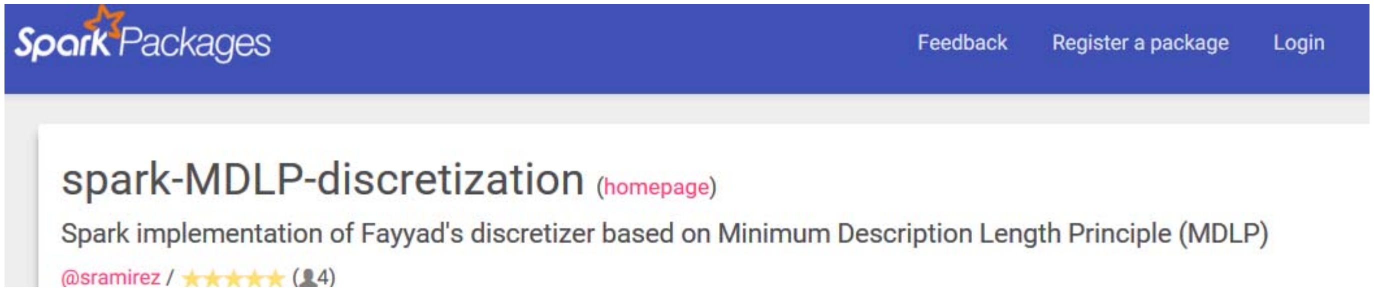
Distributed Minimum Description Length Discretizer for Spark

GitHub sramirez / **spark-MDLP-discretization**

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

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

S. Ramírez-Gallego, S. García, H. Mouriño-Talin, D. Martínez-Rego, V. Bolón, A. Alonso-Betanzos, J.M. Benitez, F. Herrera. Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark. IEEE BigDataSE Conference, Helsinki, August, 2015.



Processing Imbalanced data sets

Imbalanced Data Preprocessing for Hadoop

GitHub saradelrio / **hadoop-imbalanced-preprocessing**

https://github.com/saradelrio/hadoop-imbalanced-preprocessing

MapReduce implementations of random oversampling, random undersampling and "Synthetic Minority Oversampling TEchnique" (SMOTE) algorithms using Hadoop, used in:

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.

Big Data Preprocessing

Bird's eye view http://sci2s.ugr.es/BigData

Our approaches:

https://github.com/sramirez

Isaac Triguero

Sergio Ramírez

fast-mRMR
An improved implementation of the classical f…

spark-infotheoretic-feature-sel…
This package contains a generic implementati…

spark-MDLP-discretization
Spark implementation of Fayyad's discretizer…

https://github.com/triguero

Popular repositories

MR-EFS
This project includes the implementation of evolutionary feature s

MRPR
This repository includes the MapReduce implementation propose

ROSEFW-RF
This project contains the code used in the ROSEFW-RF paper.

Sara Del Río

https://github.com/saradelrio

hadoop-imbalanced-preprocessi ng
MapReduce implementations of random oversampling, random undersampling
(SMOTE) algorithms using Hadoop

Big Data Preprocessing

Describing some Approaches:

- **MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-ITFS: Filtering Feature Selection For Big Data**
- **Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm**
- **Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark**

Big Data Preprocessing

Describing some Approaches:

- **MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-ITFS: Filtering Feature Selection For Big Data**
- **Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm**
- **Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark**

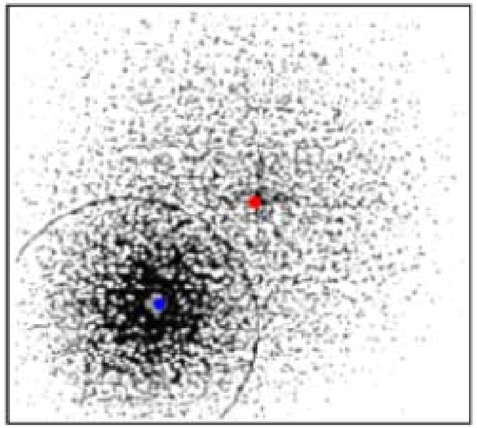
Big Data Preprocessing:MRPR

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

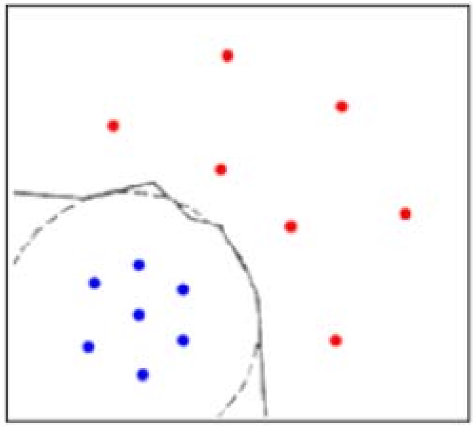
I. Triguero, D. Peralta, J. Bacardit, S.García, F. Herrera. A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation.
IEEE CEC Conference, 2014.

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 EPG.



I. Triguero

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

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 use 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:

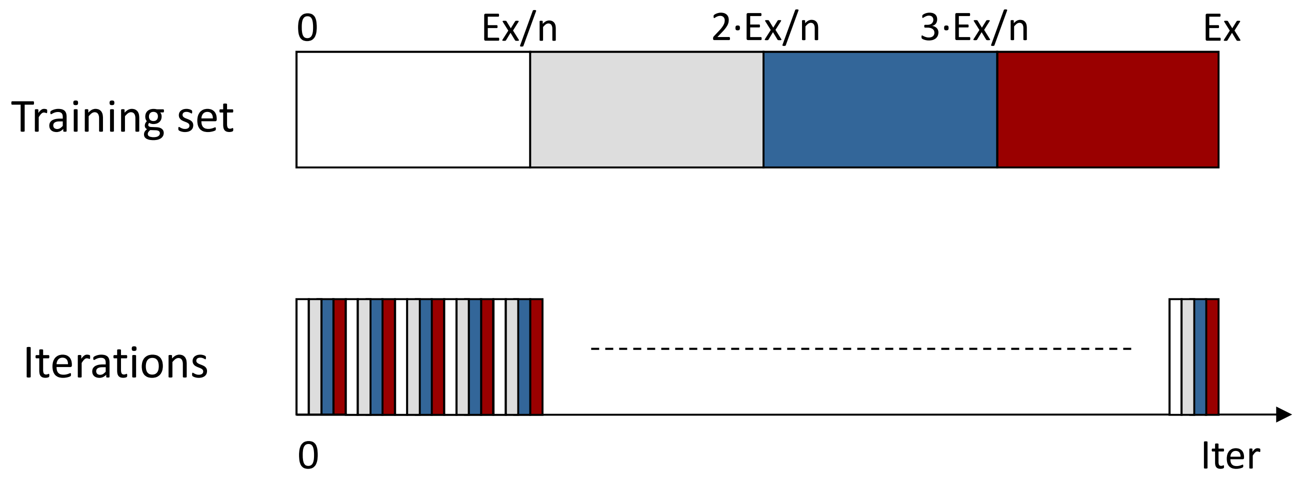
- Join: Concatenates all the resulting generated sets.
  - This process does not guarantee that the final generated set does not contain irrelevant or even harmful instances
- Fusion: This variant eliminates redundant prototypes by fusion of prototypes. Centroid-based PG methods: ICPL2 (Lam et al).

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

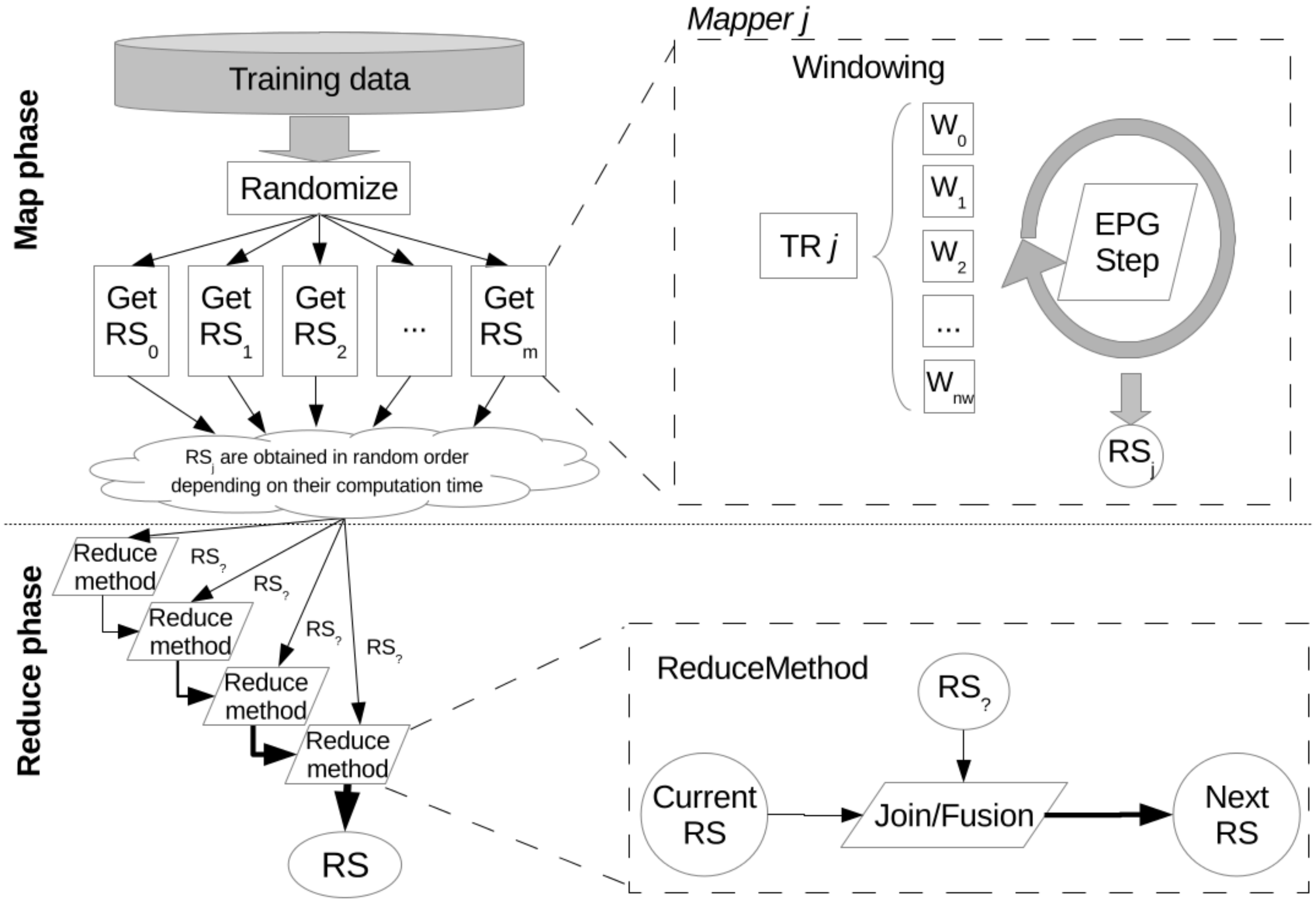
✓This mechanism also introduces some generalization pressure

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)



I. Triguero

Big Data Preprocessing: MRPR

Experimental Study

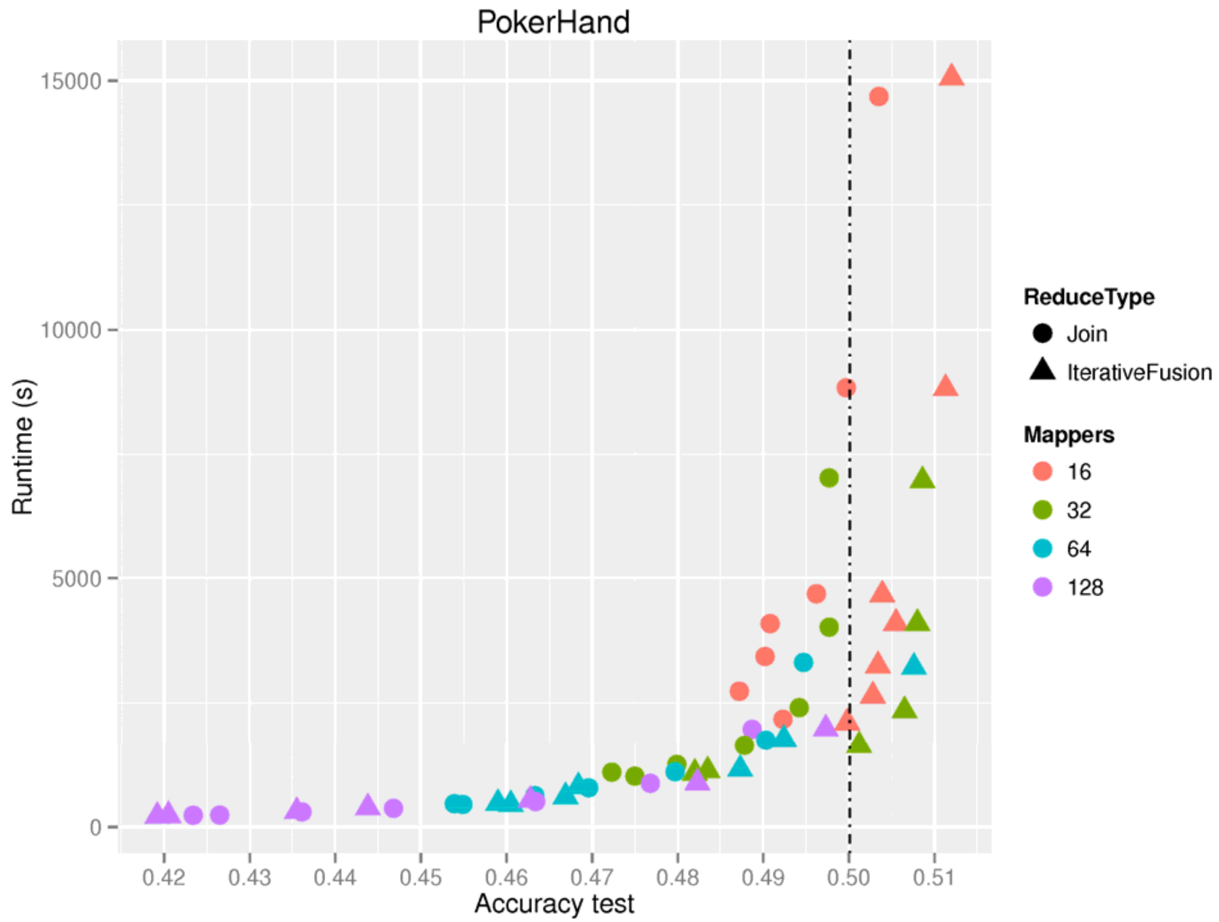
- PokerHand data set. 1 million of instances, 3x5 fcv.
- Performance measures: Accuracy, reduction rate, runtime, test classification time and speed up.
- PG technique tested: IPADECS.

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	PopulationSize = 10, iterations of Basic DE = 500 iterSFGSS =8, iterSFHC=20, Fl=0.1, Fu=0.9
ICLP2 (Fusion)	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

I. Triguero

Big Data Preprocessing: MRPR

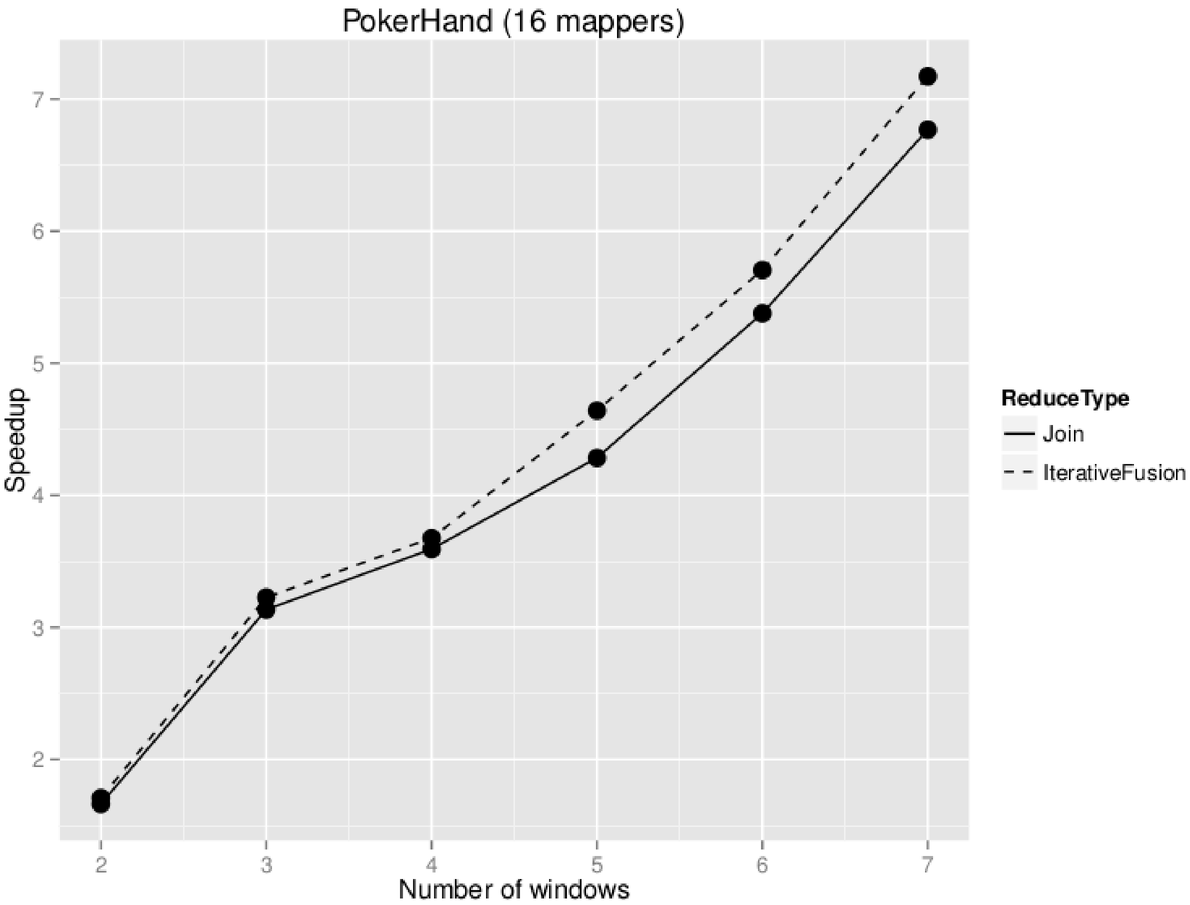
Results

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

#Windows nw	#Mappers	Training		Test		Runtime		Reduction rate		Classification
		Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.	time (TS)
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



I. Triguero

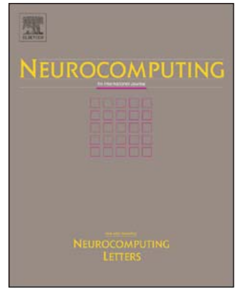
Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

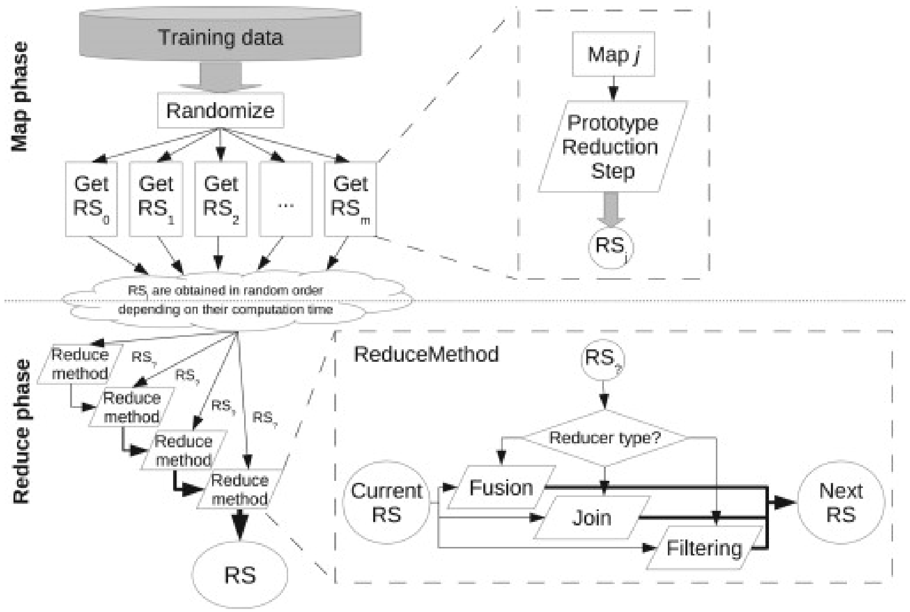
- There is a good synergy between the windowing and MapReduce approaches. They complement themselves in the proposed two-level scheme.
- Without windowing, evolutionary prototype generation could not be applied to data sets larger than approximately ten thousands instances
- The application of this model 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



- **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.*



Including:
ENN algorithm for Filtering

https://github.com/triguero/MRPR

Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

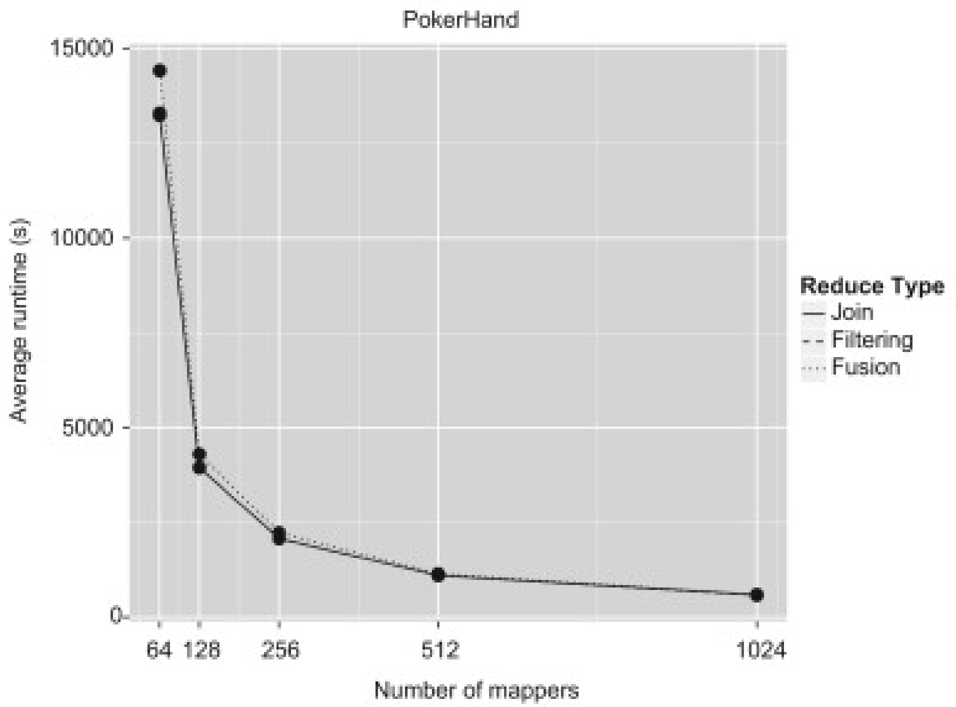


Fig. 6 Average runtime obtained by MRPR. (a) PokerHand

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

Describing some Approaches:

- MRPG: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-ITFS: Filtering Feature Selection For Big Data
- Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm
- Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark

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, 2015, In press.

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.

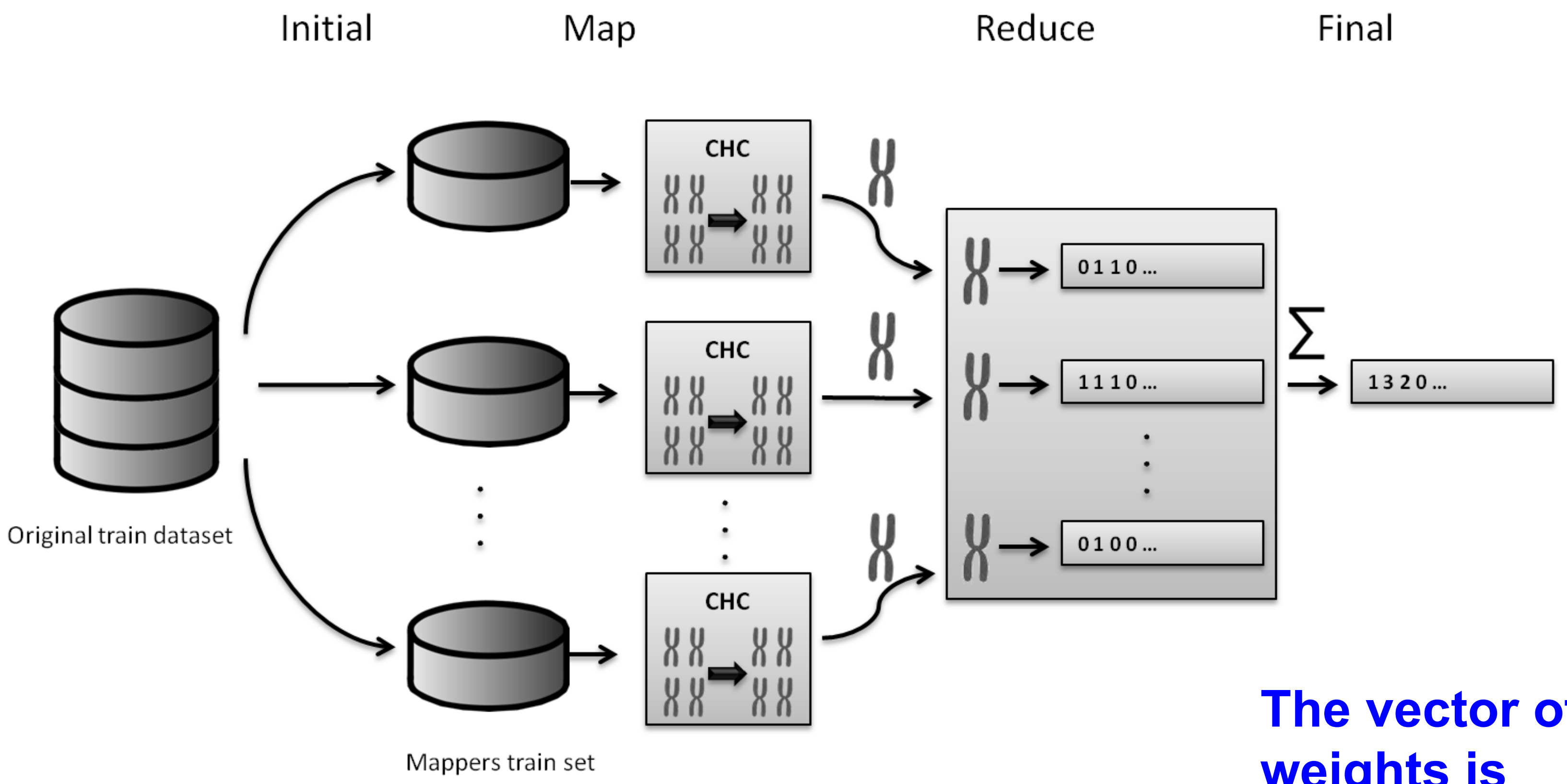
Reduce phase

- 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.

D. Peralta

Big Data Preprocessing: MR-EFS

MapReduce EFS process

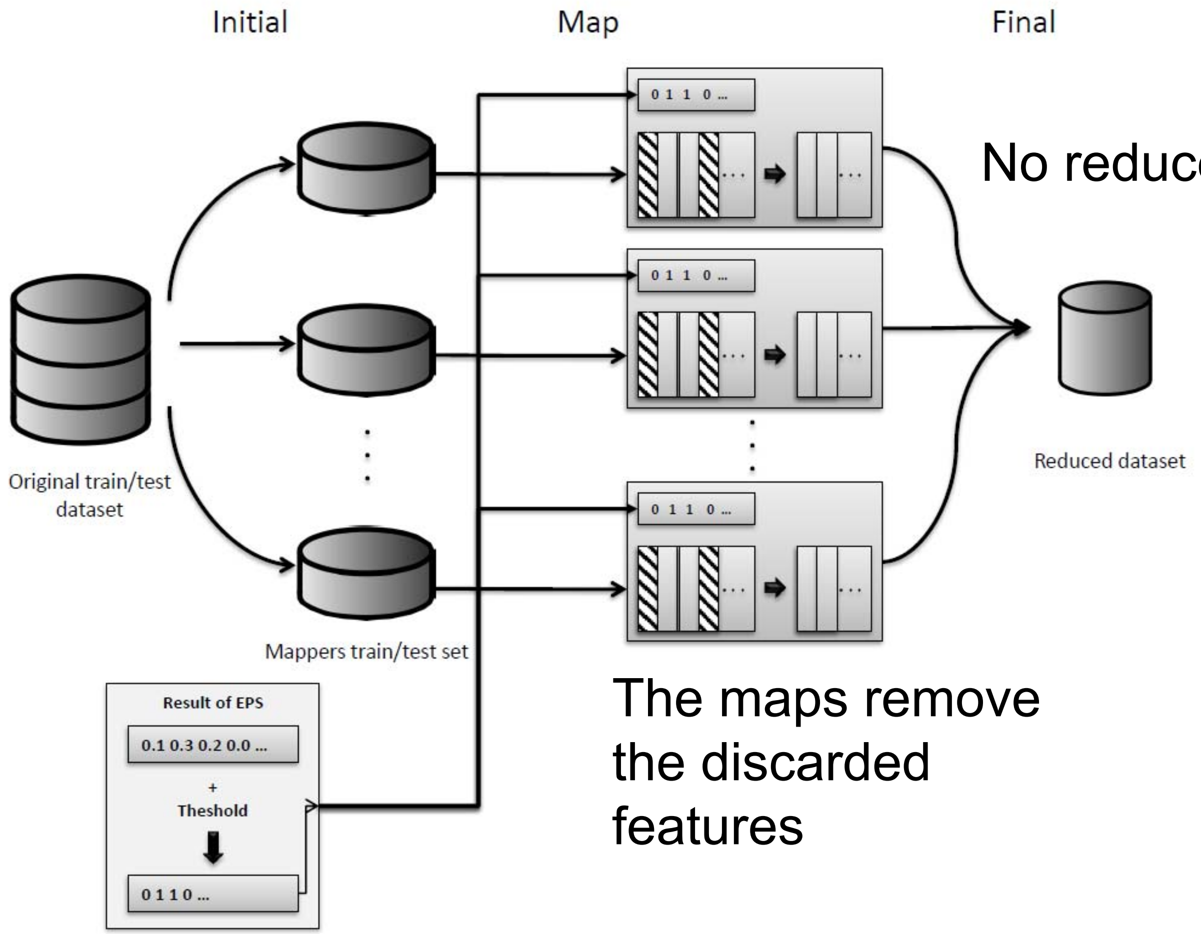


The vector of weights is binarized with a threshold

D. Peralta

Big Data Preprocessing: MR-EFS

Dataset reduction



No reduce phase

The maps remove the discarded features

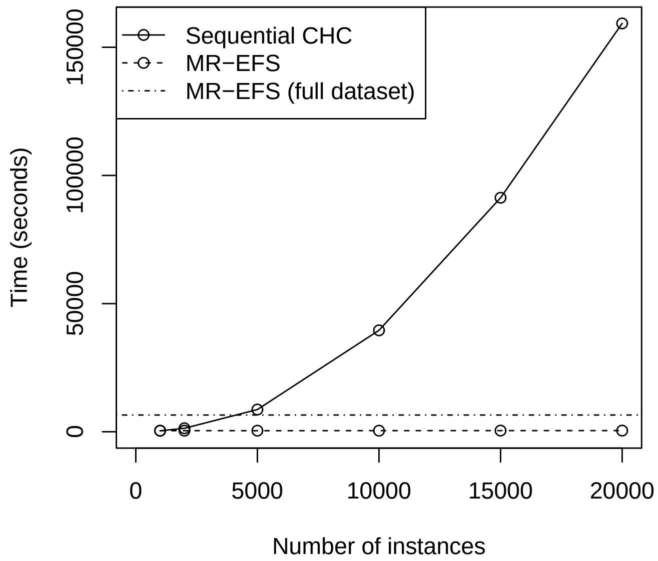
D. Peralta

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
- Three classifiers in Spark
  - SVM
  - Logistic Regression
  - Naïve Bayes
- The reduction rate is controlled with the weight threshold
- 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

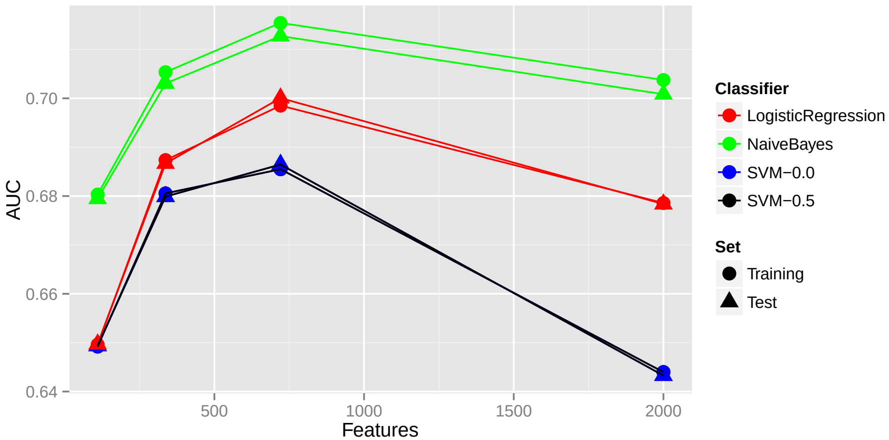
D. Peralta

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



D. Peralta

Big Data Preprocessing: MR-EFS

Experimental Study: Feature selection scalability

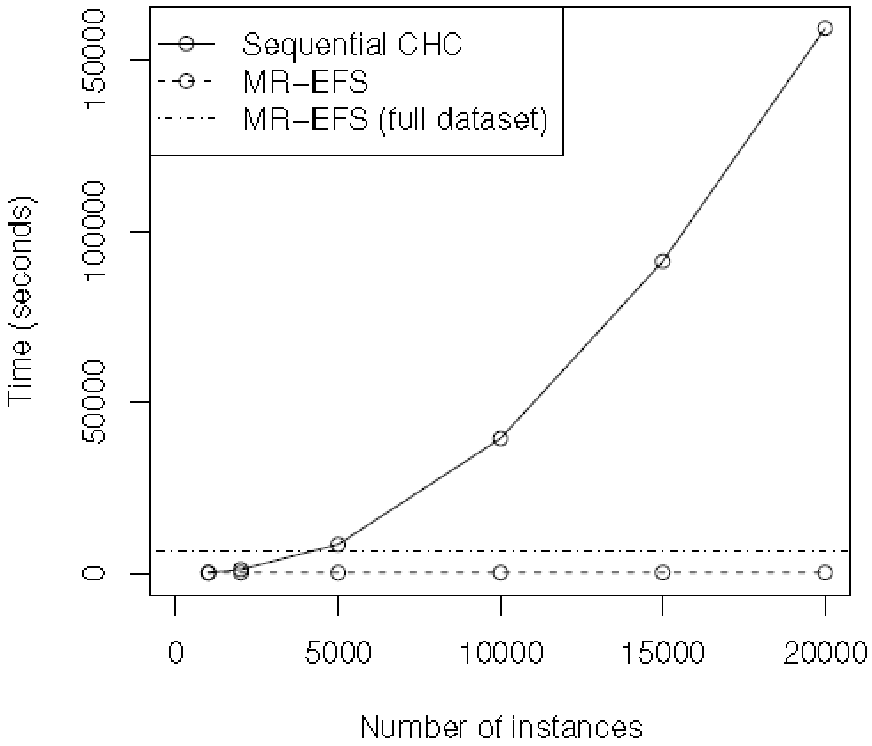


Figure 5: Execution times of the sequential CHC and MR-EFS.

D. Peralta

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

D. Peralta

Describing some Approaches:

- MRPG: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-ITFS: Filtering Feature Selection For Big Data
- Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm
- Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

- Many filtering methods are based on information theory. These are based on a quantitative criterion or index that measures its usefulness.
- **Relevance (self-interaction)**: mutual information of a feature with the class. Importance of a feature.

$$
\begin{aligned}
I(A;B) &= H(A) - H(A|B) \\
&= \sum_{a\in A}\sum_{b\in B} p(a|b)\log\frac{p(a|b)}{p(a)p(b)}.
\end{aligned}
$$

- **Redundancy (multi-interaction):** conditional mutual information between two input features. Features that carry similar information.

$$
\begin{aligned}
I(A;B|C) &= H(A|C) - H(A;B|C) \\
&= \sum_{c\in C} p(c)\sum_{a\in A}\sum_{b\in B} p(ab|c)\log\frac{p(ab|c)}{p(a|c)p(b|c)}.
\end{aligned}
$$

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

- There are a wide range of method in the literature built on these information theoretic measures.
- To homogenize the use of all these criterions, Gavin Brown proposed a **generic expression** that allows to ensemble many of these criterions in an unique FS framework:

$$J = I(X_i;Y) - \beta \sum_{j \in S} I(X_j;X_i) + \gamma \sum_{j \in S} I(X_j;X_i|Y),$$

- It is based on a greedy optimization which assesses features based on a simple scoring criterion. Through some independence assumptions, it allows to transform many criterions as linear combinations of Shannon entropy terms.

Brown G, Pocock A, Zhao MJ, Luján M (2012) Conditional likelihood max-imisation: A unifying framework for information theoretic feature selection. J Mach Learn Res 13:27–66

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

Criterion name	
Original proposal	**Brown's reformulation**
Mutual Information Maximisation (MIM) [22]	
$J_{mim}(X_i) = I(X_i;Y)$	$J_{mim} = I(X_i;Y) - 0\sum_{j\in S} I(X_j;X_i) + 0\sum_{j\in S} I(X_i;X_j\|Y)$
Mutual Information FS (MIFS) [4]	
$J_{mifs}(X_i) = I(X_i;Y) - \beta\sum_{X_j\in S} I(X_i;X_j)$	$J_{mifs} = I(X_i;Y) - \beta\sum_{j\in S} I(X_j;X_i) + 0\sum_{j\in S} I(X_i;X_j\|Y)$
Joint Mutual Information (JMI) [33]	
$J_{jmi}(X_i) = \sum_{X_j\in S} I(X_iX_j;Y)$	$J_{jmi} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{j\in S} I(X_j;X_i) + \frac{1}{\|S\|}\sum_{j\in S} I(X_i;X_j\|Y)$
Conditional Mutual Information (CMI)	
$J_{cmi} = I(X_i;Y\|S)$	$J'_{cmi} = I(X_i;Y) - \sum_{j\in S} I(X_j;X_i) + \sum_{j\in S} I(X_i;X_j\|Y)$
Minimum-Redundancy Maximum-Relevance (mRMR) [27]	
$J_{mrmr} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{j\in S} I(X_j;X_i)$	$J_{mrmr} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{j\in S} I(X_j;X_i) + 0\sum_{j\in S} I(X_i;X_j\|Y)$
Conditional Mutual Information Maximization (CMIM) [14]	
$J_{cmim} = \min_{X_j\in S}[I(X_i;Y\|X_j)]$	$J_{cmim} = I(X_i;Y) - \max_{j\in S}[I(X_j;X_i) - I(X_i;X_j\|Y)]$
Informative Fragments (IF) [31] (equivalent to CMIM)	
$J_{if} = \min_{X_j\in S}[I(X_iX_j;Y) - I(X_j;Y)]$	$J_{if} = J_{cmim} = I(X_i;Y) - \max_{j\in S}[I(X_j;X_i) - I(X_i;X_j\|Y)]$
Interaction Capping (ICAP) [20]	
$J_{icap} = I(X_i;Y) - \sum_{X_j\in S}\max[0, I(X_i;X_j) - I(X_i;X_j\|Y)]$	$J_{icap} = I(X_i;Y) - \sum_{X_j\in S}\max[0, I(X_i;X_j) - I(X_i;X_j\|Y)]$

S. Ramirez

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

- We propose a distributed version of this framework based on a greedy approach (each iteration the algorithm selects one feature).
- As **relevance** values do not change, we compute them first and cache to reuse.
- Then, **redundancy** values are calculated between the non-selected features and the last one selected.

$$J = I(X_i;Y) - \beta \sum_{j \in S} I(X_j;X_i) + \gamma \sum_{j \in S} I(X_j;X_i|Y),$$

An Information Theoretic Feature Selection Framework for Big Data under Apache Spark

Sergio Ramírez-Gallego, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera

Algorithm 1 Main FS Algorithm

Input: D — Data set
Input: $|S_\theta|$ — Number of features to select
Output: S_θ — Index list of selected features

$calcs \leftarrow calculateRelevances(D)$
$criterions \leftarrow initCriterions(calcs)$
$p_{best} \leftarrow extractTopRelevant(criterions, 1)$
$S \leftarrow Set(p_{best})$
while $|S| < |S_\theta|$ **do**
    $calcs \leftarrow calculateMIAndCMI(criterions, p_{best})$
    $criterions \leftarrow updateCriterions(criterions, calcs)$
    $p_{best} \leftarrow extractTopCriterions(criterions, 1)$
    $S \leftarrow addTo(p_{best}, S)$
end while
return S

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

- The **most challenging** part is to **compute the mutual and conditional information results.**
- It supposes to compute all combinations necessary for **marginal and joint probabilities**.
- This imply to run several **Map-Reduce phases** to distribute and joint probabilities with its correspondent feature.

An Information Theoretic Feature Selection Framework for Big Data under Apache Spark

Sergio Ramírez-Gallego, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera

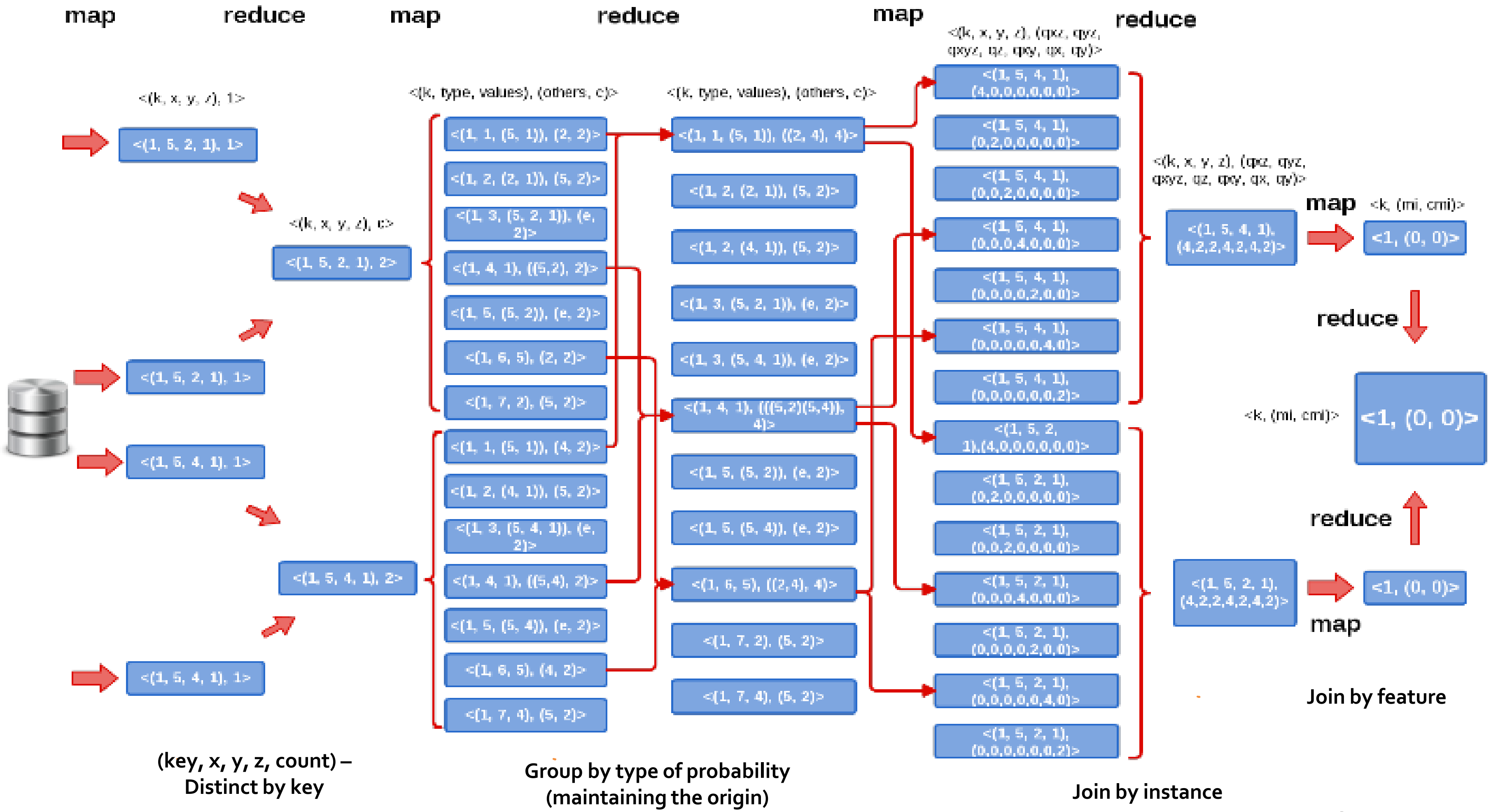
Algorithm 3 Calculate MI and CMI

Input: D	Data set
Input: V_1	Features indexes for primary varia
Input: V_2	Feature index for secondary varial
Input: V_c	Feature index for conditional varia

Output: $MI = I(V_1; V_2)$
Output: $CMI = I(V_1; V_2 | V_c)$
$genCombinations \leftarrow forall(k \in V_1)\ EMIT < (k, E_i(k), E_i(V_2), E_i(V_c)), 1 >$
$comb \leftarrow map(data,\ genCombinations)$
$comb \leftarrow reduce(comb,\ sumValues)$
$freqsByProb \leftarrow map(comb,\ splitByProb)$
$reduceByProb \leftarrow (addToSet(others1,\ others2),\ q1 + q2)$
$freqsByProb \leftarrow reduce(freqsByProb,\ reduceByProb)$
$combByOrigin \leftarrow map(freqsByProb,\ mapByOrigin)$
$combByOrigin \leftarrow reduce(combByOrigin,\ addVectors)$
$miCalcs \leftarrow map(combByOrigin,\ calcMIandCMI)$
$miCmi \leftarrow reduce(sumValues)$
return $miCmi$

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data



S. Ramirez

Big Data Preprocessing: Spark-ITFS

Experimental Framework

- **Datasets:** Two huge datasets (*ECBDL14 and epsilon*)

Data Set	#Train Ex.	#Test Ex.	#Atts.	#Total	#Cl.
epsilon	400 000	100 000	2000	800 000 000	2
ECBDL14 (ROS)	65 003 913	2 897 917	631	41 017 469 103	2

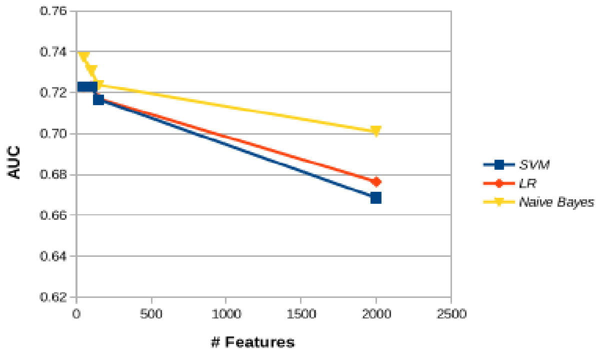
- **Parameters:**

Method	Parameters
Naive Bayes	lambda = 1.0, iterations = 100
LR	stepSize = 0.2, batchFraction = 1.0, iterations = 100
SVM	stepSize = 1.0, batchFraction = 1.0, regularization = 1.0, iterations = 100
mRMR	pool size = 0

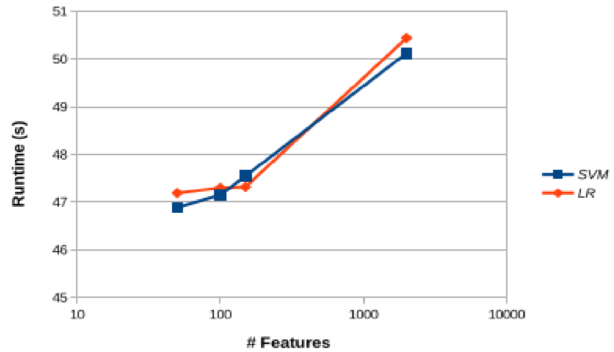
- **Measures:** AUC, selection and classification time.
- **Hardware:** 16 nodes (12 cores per node), 64 GB RAM.
- **Software**: Hadoop 2.5 and Apache Spark 1.2.0.

Big Data Preprocessing: Spark-ITFS

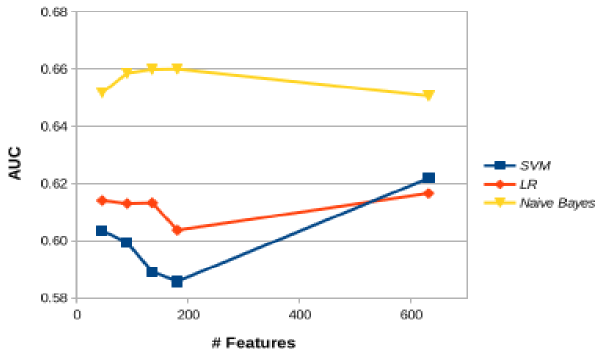
Experimental Results: AUC and classification time



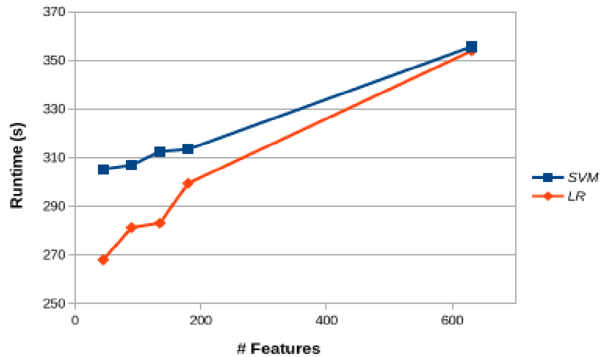
(a) AUC values (epsilon)



(b) Runtime values for epsilon (in logarithm scale). Times for Naive Bayes are not shown as they are too small



(c) AUC values (ECBDL14)



(d) Runtime values for ECBDL14 (in logarithm scale). Times for Naive Bayes are not shown as they are too small

S. Ramirez

Big Data Preprocessing: Spark-ITFS

Experimental Results: Selection Time (in seconds)

# Dataset	# Features selected	Total time	Average time/feature
ECBDL14	200	29108.86	145.54
epsilon	1000	12541.54	12.54

Big Data Preprocessing: Spark-ITFS

Filtering Feature Selection For Big Data

Info-Theoretic Framework 2.0:

- **Data column format:** Row/Instance data are transformed to a column-wise format. Each feature computation (X) is isolated and parallelized in each step.
- **Cached marginal and joint probabilities:** in order to reuse them in next iterations.
- **Broadcasted variables:** Y and Z features and its marginal/joint values are broadcasted in each iteration.
- Support for **high-dimensional and sparse problems**: zero values are calculated from the non-zero values avoiding explosive complexity. Millions of features can be processed.

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

S. Ramirez

Big Data Preprocessing

Describing some Approaches:

- **MRPG: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-ITFS: Filtering Feature Selection For Big Data**
- **Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm**
- **Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark**

Big Data Preprocessing: Fast-mRMR

Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm

Original proposal (mRMR):

- Rank features based on their relevance to the target, and at the same time, the redundancy of features is also penalized.
- Maximum dependency: mutual information (MI) between a feature set *S* with *xi features* and the targe class *c:* $max\ D(S,c)\ ;\ D = \frac{1}{|S|}\sum_{x_i \in S} I(x_i;c)$
- Minimum redundancy: MI between features. $min\ R(S)\ ;\ R = \frac{1}{|S|^2}\sum_{x_i,x_j \in S} I(x_i,x_j)$
- Combining two criterions: $\Phi = D - R$

Hanchuan Peng, Fulmi Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 27(8):1226–1238, 2005.

Big Data Preprocessing: Fast-mRMR

Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm

Improvements:

- **Accumulating Redundancy** (greedy approach): in each iteration only compute MI between last selected feature and those non-selected. Select one feature by iteration.
- **Data-access pattern**: column-wise format (more natural approach).
- **Caching marginal probabilities:** computed once at the beginning, saving extra computations.

$$\mathrm{mRMR} = \max_{S}\left[\frac{1}{|S|}\sum_{f_i \in S} I(f_i;c) - \frac{1}{|S|^2}\sum_{f_i,f_j \in S} I(f_i;f_j)\right].$$

Big Data Preprocessing: Fast-mRMR

Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm

Software package with several versions for:

- **CPU**: implemented in C++. For small and medium datasets.
- **GPU**: mapped MI computation problem to histogramming problem in GPU. Parallel version.
- **Apache Spark:** for Big Data problems.

Code: https://github.com/sramirez/fast-mRMR

S. Ramirez

Big Data Preprocessing: Fast-mRMR

Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm

DataSets	mRMR	fast-mRMR	speedup
lung	23.27	0.06	387.83
nci	39.41	2.02	19.51
colon	40.24	0.37	108.76
leuk	43.54	1.51	28.83
lym	50.81	0.95	53.48

(a) mRMR vs. fast-mRMR (time in seconds). 200 features selected.

# Samples	CPU	GPU	speedup
160	0.08	0.50	0.16
1600	0.28	0.67	0.42
16000	0.53	0.75	0.71
160000	2.64	1.04	2.54
1600000	22.49	5.28	4.26
3200000	42.34	7.96	5.32

(b) fast-mRMR: CPU vs. GPU version (time in seconds). 1,000 features and 100 selected.

# Datasets.	# Features	# Instances	CPU	Spark	Speedup
ECBDL14	631	65,003,913	11,281.27	1,980.46	5,69
epsilon	2,000	400,000	2,553.79	253.72	10,07

(c) fast-mRMR: CPU vs. Spark version (time in seconds). 100 features selected.

S. Ramirez

Describing some Approaches:

- **MRPG: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-ITFS: Filtering Feature Selection For Big Data**
- **Fast-mRMR: an optimal implementation of minimum Redundancy Maximum Relevance algorithm**
- **Spark-MDLP: Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark**

Big Data Preprocessing: Spark-MDLP

Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark

Sergio Ramírez-Gallego, Salvador García, Héctor Mouriño-Talín, David Martínez-Rego ,
Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, Francisco Herrera
Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark.
IEEE BigDataSE, 2015.

Big Data Preprocessing: Spark-MDLP

Introduction

- The astonishing rate of data generation on the Internet nowadays has caused that many classical knowledge extraction techniques have become obsolete.
- Data reduction (discretization) techniques are required in order to reduce the complexity order held by these techniques.
- In spite of the great interest, only a few simple discretization techniques have been implemented in the literature for Big Data.
- We propose a distributed implementation of the entropy minimization discretizer proposed by Fayyad and Irani using **Apache Spark** platform.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. NSDI 2012. April 2012.

U. M. Fayyad and K. B. Irani, “Multi-interval discretization of continuous-valued attributes for classification learning,” in International Joint Conference On Artificial Intelligence, vol. 13. Morgan Kaufmann, 1993, pp. 1022–1029.

Big Data Preprocessing: Spark-MDLP

Discretization: An Entropy Minimization Approach

- **Discretization**: transforms numerical attributes into discrete or nominal attributes with a finite number of intervals.
- **Main objective**: find the best set of points/intervals according to a quality measure (e.g.: inconsistency or entropy).
- Optimal discretization is NP-complete, determined by the number of **candidate cut points** (all distinct values in the dataset for each attribute).
- A possible optimization: use only **boundary points** in the whole set (midpoint between two values between two classes).

S. García, J. Luengo, and F. Herrera, Data Preprocessing in Data Mining. Springer, 2015.

U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in International Joint Conference On Artificial Intelligence, vol. 13. Morgan Kaufmann, 1993, pp. 1022–1029.

Big Data Preprocessing: Spark-MDLP

Discretization: An Entropy Minimization Approach

- **Minimum Description Length Discretizer (MDLP)** implements this optimization and multi-interval extraction of points (which improves accuracy and simplicity respect to ID-3).
- **Quality measure**: class entropy of partitioning schemes (S1, S2) for attribute A. Find the best cut point.

$$EP(A, b_\alpha, S) = \frac{|S_1|}{|S|} E(S_1) + \frac{|S_2|}{|S|} E(S_2),$$

- **Stop criterion:** MDLP criterion

$$G(A, b_\alpha, S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A, b_\alpha, S)}{N},$$

$$G(A, b_\alpha, S) = E(S) - EP(A, b_\alpha, S)$$

$$\Delta(A, b_\alpha, S) = log_2(3^c) - [cE(S) - c_1 E(S_1) - c_2 E(S_2)]$$

U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in International Joint Conference On Artificial Intelligence, vol. 13. Morgan Kaufmann, 1993, pp. 1022–1029.

Big Data Preprocessing: Spark-MDLP

Distributed MDLP Discretization: a complexity study

- **Determined by two time-consuming operations (repeated for each attribute):**
  - **Sorting:** O(|A| log |A|), assuming A is an attribute with all its points distinct.
  - **Evaluation:** quadratic operation (boundary points).
- **Proposal:**
  - Sort all points in the dataset using a single distributed operation.
  - Evaluates boundary points (per feature) in an parallel way.
- **Main Primitives:** using **Apache Spark**, a large-scale processing framework based on in-memory primitives. Some primitives used:
  - **SortByKey:** sort tuples by key in each partition. Partitions are also sorted.
  - **MapPartitions:** an extension of Map-Reduce operation for partition processing.

S. Ramirez

Big Data Preprocessing: Spark-MDLP

Distributed MDLP Discretization: main procedure

1) Distinct points: it creates tuples where key is formed by the feature index and the value. Value: frequency vector for classes.

2) Sorting: distinct points are sorted by key (feature, value).

3) Boundary points: boundary points are calculated evaluating consecutives points.

4) Points grouped by feature: grouped and new key: only the attribute index.

5) Attributes divided by size: depends on the number of boundary points (> 10,000 points)

6) Points selection and MLDP evaluation

Algorithm 1 Main discretization procedure

Input: S Data set
Input: M Feature indexes to discretize
Input: $maxbins$ Maximum number of cut points to select
Input: $maxcand$ Maximum number of candidates per partition
Output: Cut points by feature

$comb \leftarrow$
map $s \in S$
  $v \leftarrow zeros(k)$
  $v(c) \leftarrow 1$
  for all $A \in M$ **do**
    $EMIT < (A, A(s)), v >$
  end for
end map
$distinct \leftarrow reduce(comb,\ sum\_vectors)$
$sorted \leftarrow sort\_by\_key(distinct)$
$first \leftarrow first\_by\_part(sorted)$
$boundaries \leftarrow get\_boundary\_points(sorted, first)$
$boundaries \leftarrow$
map $b \in boundaries$
  $< (att, point,)q > \leftarrow b$
  $EMIT < (att, (point, q)) >$
end map
$(small, big) \leftarrow divide\_attributes(boundaries, maxcand)$
$sthresholds \leftarrow select\_thresholds(small, maxbins, maxcand)$
for all $att \in keys(big)$ **do**
  $bthresholds \leftarrow bthresholds + select\_thresholds(big(att), maxbins, maxcand)$
end for
$return(union(bthresholds, sthresholds))$

S. Ramirez

Big Data Preprocessing: Spark-MDLP

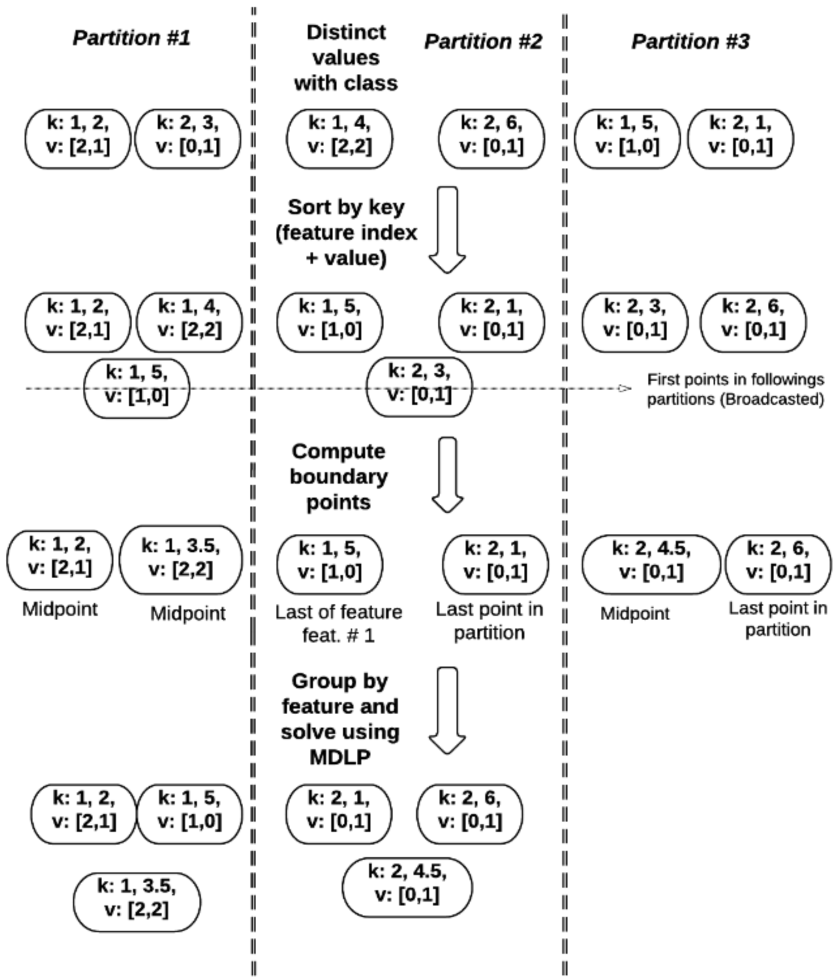
Distributed MDLP Discretization: main procedure

1) **Distinct points:** already calculated in step #1 from raw data.

2) **Sorting:** distinct points are sorted by key. For the next step, the first points in each partition are sent to the following partition.

3) **Boundary points:** midpoints are generated when two points of the same attribute are on the border. Last points in a partition and in a feature are also added.

4) **MLDP selection and evaluation:** parallelized by feature



S. Ramirez

Big Data Preprocessing: Spark-MDLP

Distributed MDLP Discretization: attribute division

- Once boundary points are generated, for each attribute do:
  - **Points < limit:** group in a local list.
  - **Point > limit:** associate a list of partitions (distributed) → *unusual*.
- Then, the points are **evaluated recursively,** depending of the aforementiond parameter. In case of partitions, the process is iterative, whereas for list of points, it is distributed.

Algorithm 3 Function to select the best cut points for a given feature *(select_thresholds)*

Input: $candidates$ A RDD/array of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Input: $maxbins$ Maximum number of intervals or bins to select
Input: $maxcand$ Maximum number of candidates to eval in a partition
Output: An array of thresholds for a given feature

```
stack ← enqueue(stack, (candidates, ()))
result ← ()
while |stack| > 0 & |result| < maxbins do
  (subset, lth) ← dequeue(stack)
  if |subset| > 0 then
    if type(subset) = 'array' then
      bound ← arr_select_thresholds(subset, lth)
    else
      bound ← rdd_select_thresholds(subset, lth, maxcand)
    end if
    if bound <> () then
      result ← result + bound
      (left, right) ← divide_partitions(subset, bound)
      stack ← enqueue(stack, (left, bound))
      stack ← enqueue(stack, (right, bound))
    end if
  end if
end while
return(sort(result))
```

S. Ramirez

Big Data Preprocessing: Spark-MDLP

Distributed MDLP Discretization: MDLP evaluation (small)

1)Prerequisites: Points must be sorted.

2)Compute the total frequency for all classes

3)For each point:

- Left accumulator: computed from the left.
- Right accumulator: computed using the left one and the total,

4) The point is evaluated as:

(point, frequency, left, right)

Algorithm 4 Function to select the best cut point according to MDLP criterion (single-step version) *(arr_select_thresholds)*

Input: $candidates$ An array of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Input: lth Last threshold selected.
Output: The minimum-entropy cut point

$total \leftarrow sum\_freqs(candidates)$
$leftacc \leftarrow ()$
for $< point, q > \in candidates$ **do**
  $leftacc \leftarrow leftacc + q$
  $freqs \leftarrow freqs + (point, q, leftacc, total - leftacc)$
end for
$return(select\_best\_cut(candidates, freqs))$

S. Ramirez

Big Data Preprocessing: Spark-MDLP

Distributed MDLP Discretization: MDLP evaluation (big)

1)Prerequisites: Points and partitions must be sorted

2)Compute the total frequency by partition

3) Compute the accumulated frequency

4)For each partition:

- Left accumulator: computed from the left.
- Right accumulator: computed using the left one and the total,

5) The point is evaluated as: (point, frequency, left, right).

Algorithm 5 Function that selects the best cut points according to MDLP criterion (RDD version) *(rdd_select_thresholds)*

Input: $candidates$ An RDD of tuples ($< point, q >$), where $point$ represents a candidate point to evaluate and q the class counter.
Input: lth Last threshold selected
Input: $maxcand$ Maximum number of candidates to eval in a partition
Output: The minimum-entropy cut point

$npart \leftarrow round(|candidates|/maxcand)$
$candidates \leftarrow coalesce(candidates, npart)$
$totalpart \leftarrow$
map partitions $partition \in candidates$
    $return(sum(partition))$
end map
$total \leftarrow sum(totalpart)$
$freqs \leftarrow$
map partitions $partition \in candidates$
    $index \leftarrow get\_index(partition)$
    $lefttotal \leftarrow ()$
    $freqs \leftarrow ()$
    for $i = 0$ $until$ $index$ **do**
        $lefttotal \leftarrow lefttotal + totalpart(i)$
    end for
    for all $< point, q > \in partition$ **do**
        $freqs \leftarrow freqs + (point, q, leftotal + q, total - leftotal)$
    end for
    $return(freqs)$
end map
$return(select\_best\_cut(candidates, freqs))$

S. Ramirez

Big Data Preprocessing: Spark-MDLP

Experimental Framework

- **Datasets:** Two huge datasets (*ECBDL14 and epsilon*)

TABLE I
SUMMARY DESCRIPTION FOR CLASSIFICATION DATASETS

Data Set	#Train Ex.	#Test Ex.	#Atts.	#Total	#Cl.
epsilon	400 000	100 000	2000	800 000 000	2
ECBDL14 (ROS)	65 003 913	2 897 917	631	41 017 469 103	2

- **Parameters:** 50 intervals and 100,000 max candidates per partition.
- **Classifier:** Naive Bayes from MLLib, lambda = 1, iterations = 100.
- **Measures:** discretization time, classification time and classification accuracy.
- **Hardware:** 16 nodes (12 cores per node), 64 GB RAM.
- **Software**: Hadoop 2.5 and Apache Spark 1.2.0.

Big Data Preprocessing: Spark-MDLP

Performance results: Classification accuracy

- **Clear advantage** on using discretization in both datasets.
- Specially important for **ECBDL14**.

TABLE II
CLASSIFICATION ACCURACY VALUES

	NB		NB-disc	
Dataset	*Train*	*Test*	*Train*	*Test*
ECBDL14	0.5260	0.6276	**0.6659**	**0.7260**
epsilon	0.6542	0.6550	**0.7094**	**0.7065**

Big Data Preprocessing: Spark-MDLP

Performance results: Classification time modelling

- **Light improvement** between both versions. It seems that using discretization does not affect too much the modelling performance.
- Despite of being **insignificant**, the time value for discretization is a bit better than for the other one.

TABLE III
CLASSIFICATION TIME VALUES (WITH DISCRETIZATION VS. W/O DISCRETIZATION) IN SECONDS

Dataset	NB	NB-disc
ECBDL14	31.06	**26.39**
epsilon	5.72	**4.99**

Big Data Preprocessing: Spark-MDLP

Performance results: Discretization time

- **High speedup** between the sequential version and the distributed one for both datasets.
  - For ECBDL14, our version is almost **300 times faster**.
  - Even for the medium-size dataset (*epsilon*), there is a clear advantage.
- **The bigger the dataset, the higher the improvement.**

TABLE IV
MDLP TIME VALUES (SEQUENTIAL VS. DISTRIBUTED) IN SECONDS

Dataset	Sequential (estimation)	Distributed
ECBDL14	295 508	**1 087**
epsilon	5 764	**476**

S. Ramirez

Spark-MDLP: Final Comments

- We have proposed a sound **multi-interval discretization method** based on entropy minimization for large-scale discretization. This has implied a **complete redesign** of the original proposal.
- Adapting discretization methods to Big Data is **not a trivial task** (only few simple techniques implemented).
- The experimental results has demonstrated **the improvement in accuracy and time** (both classification and discretization) with respect to the sequential proposal.

https://github.com/sramirez/spark-MDLP-discretization

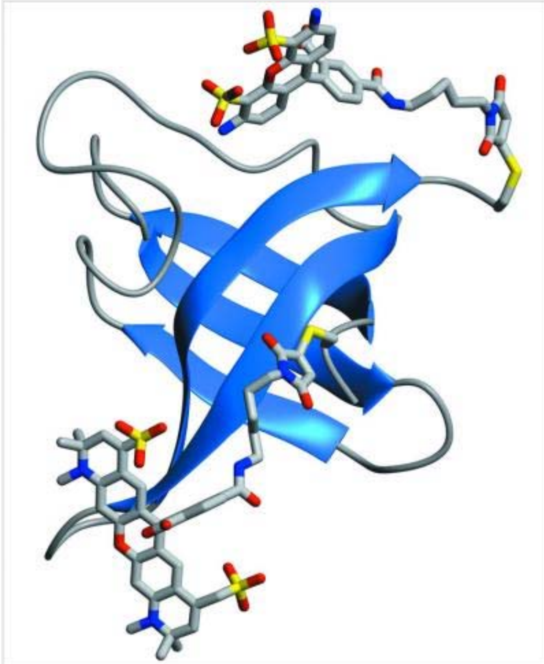
Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big data Preprocessing
- Imbalanced Big Data Classification: Data preprocessing
- Challenges and Final Comments

Evolutionary Computation for Big Data and Big Learning Workshop

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

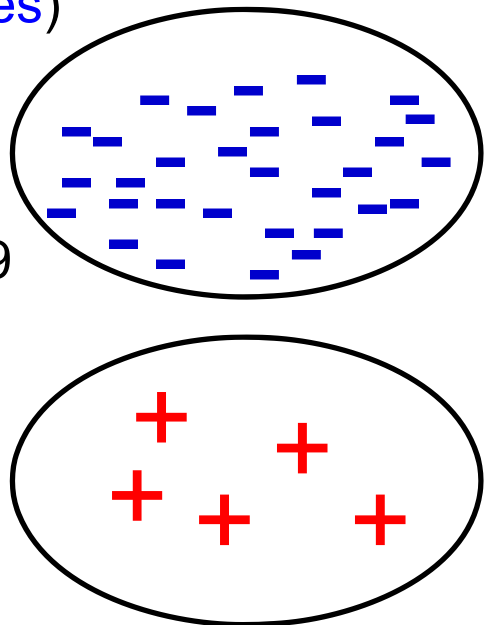
Evolutionary Computation for Big Data and Big Learning Workshop

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



Imbalanced Big Data Classification
Introduction

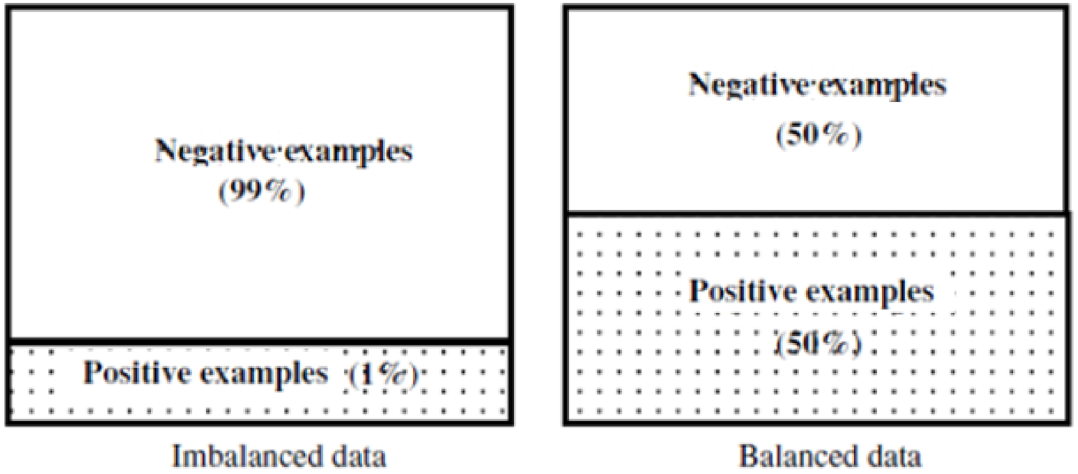


Fig. 1. Imbalanced and balanced data sets.

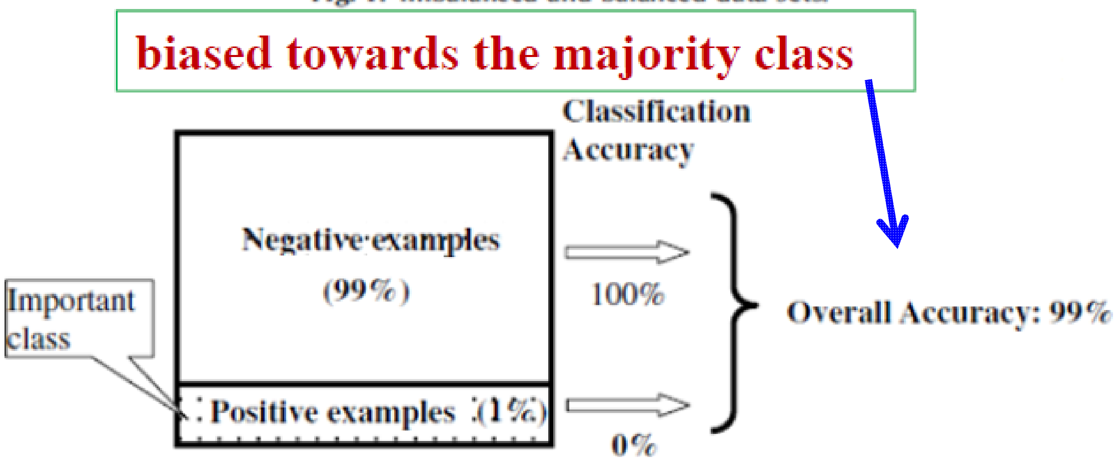


Fig. 2. The illustration of class imbalance problems.

Imbalanced classes problem: **standard learners are often biased towards the majority class.**

We need to change the way to evaluate a model performance!

Imbalanced Big Data Classification
Introduction

Strategies to deal with imbalanced data sets

Motivation

Over-Sampling
- Random
- Focused

Retain influential examples
Balance the training set

Under-Sampling
- Random
- Focused

Remove noisy instances in the decision boundaries
Reduce the training set

- Cost Modifying (cost-sensitive)
- **Algorithm-level approaches: A commont strategy to deal with the class imbalance is to choose an appropriate inductive bias.**
- **Boosting approaches: ensemble learning, AdaBoost, …**

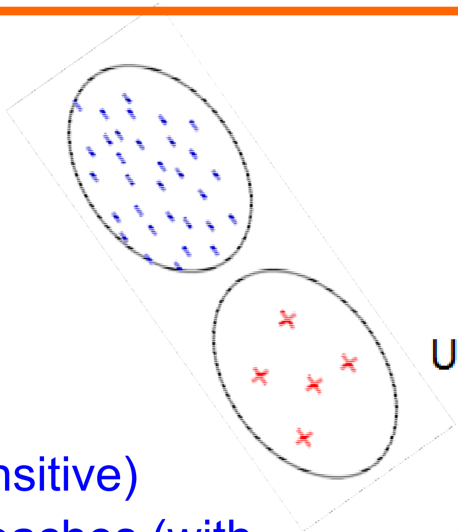
Imbalanced Big Data Classification

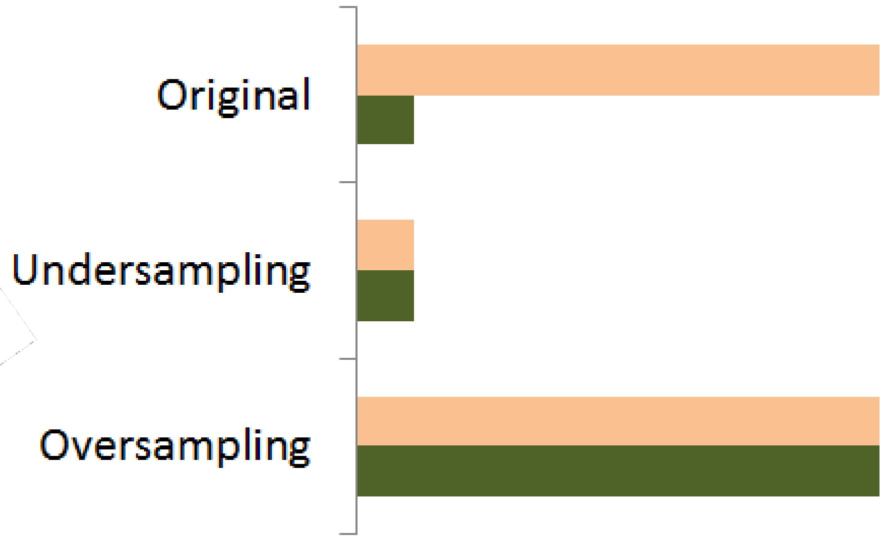
A MapReduce Approach

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.*

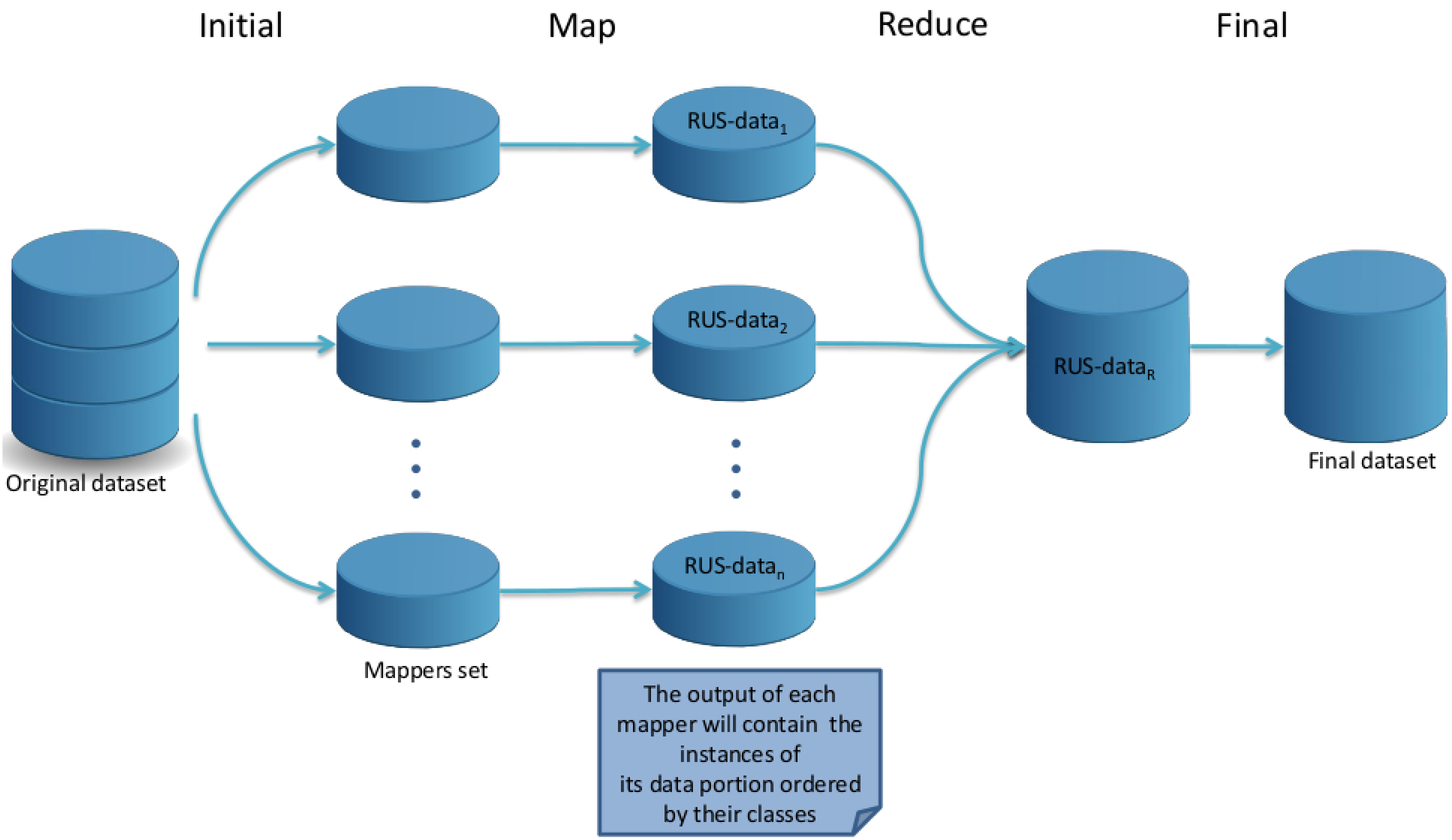
- Over-Sampling
  - Random
  - Focused
- Under-Sampling
  - Random
  - Focused
- Cost Modifying (cost-sensitive)
- Boosting/Bagging approaches (with preprocessing)





Imbalanced Big Data Classification

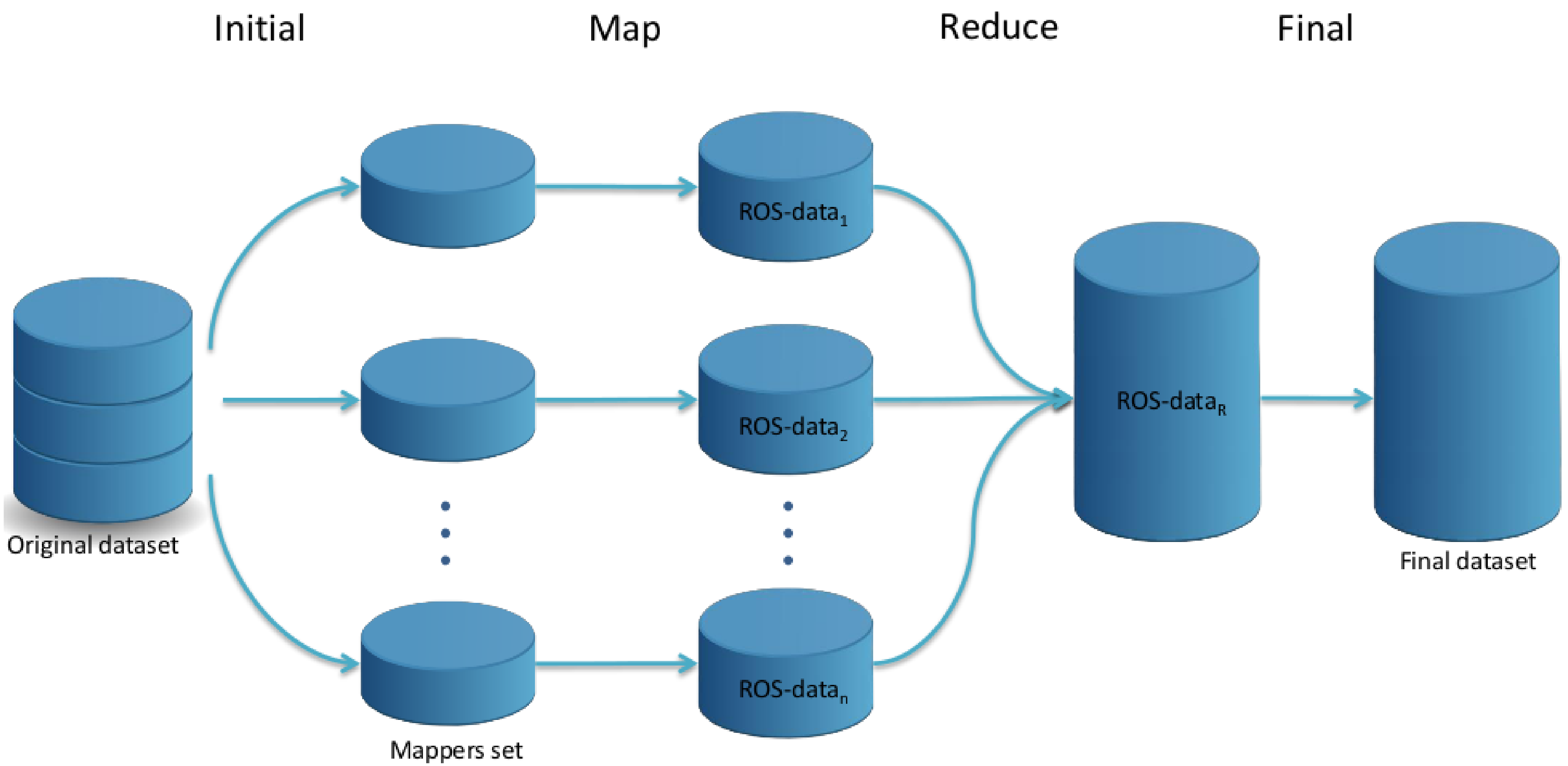
A MapReduce Approach for Random Undersampling



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.

Imbalanced Big Data Classification

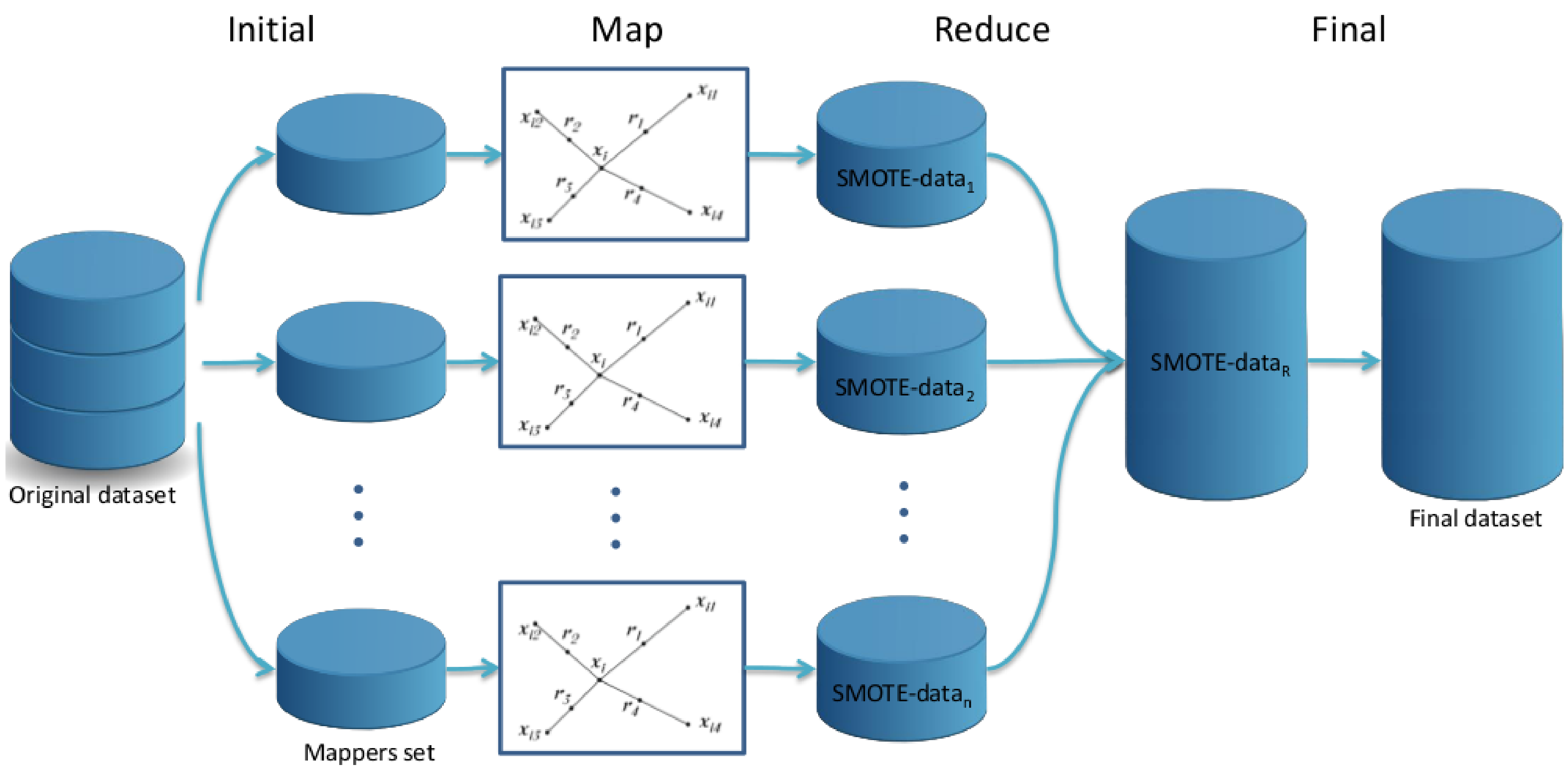
A MapReduce Approach for Random Oversampling



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.

Imbalanced Big Data Classification

A MapReduce Approach for Adapting the generation of synthetic minority samples



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.

Imbalanced Big Data Classification

Dataset	Average (kddcup)							
	8 mappers		16 mappers		32 mappers		64 mappers	
	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}	GM_{tr}	GM_{tst}
Big data versions								
RF-BigData	0.7620	0.7505	0.6985	0.6976	0.6852	0.6836	0.6626	0.6598
RF-BigDataCS	0.9404	0.9305	0.9480	0.9651	0.9173	0.9328	0.9372	0.9286
ROS+RF-BigData	1.0000	0.9661	0.9999	**0.9696**	0.9999	**0.9773**	0.9999	**0.9857**
RUS+RF-BigData	0.9869	**0.9843**	0.9490	0.9336	0.7103	0.7104	0.7049	0.7048
SMOTE+RF-BigData	0.9477	0.9140	0.9381	0.9191	0.9445	0.9091	0.8994	0.8722

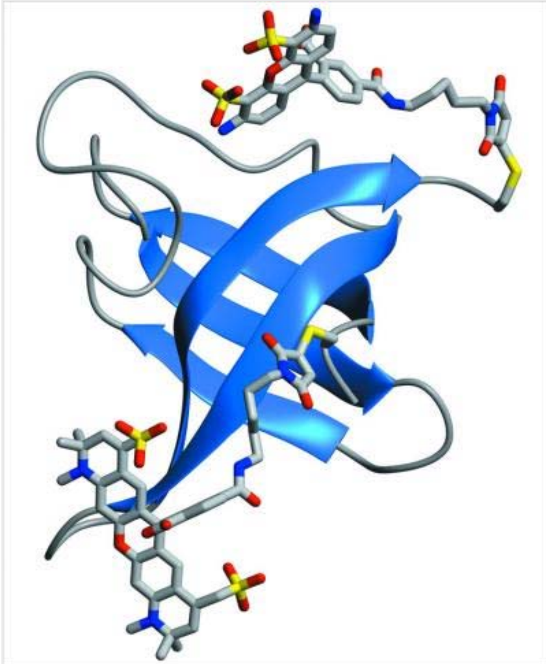
Analysis of the effectiveness in classification of the approaches

Potential problem: lack of density of the positive class for RUS/SMOTE. Lack of data due to the data fragmentation for MapReduce

https://github.com/saradelrio/hadoop-imbalanced-preprocessing

Evolutionary Computation for Big Data and Big Learning Workshop

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

ECBDL'14 Big Data Competition

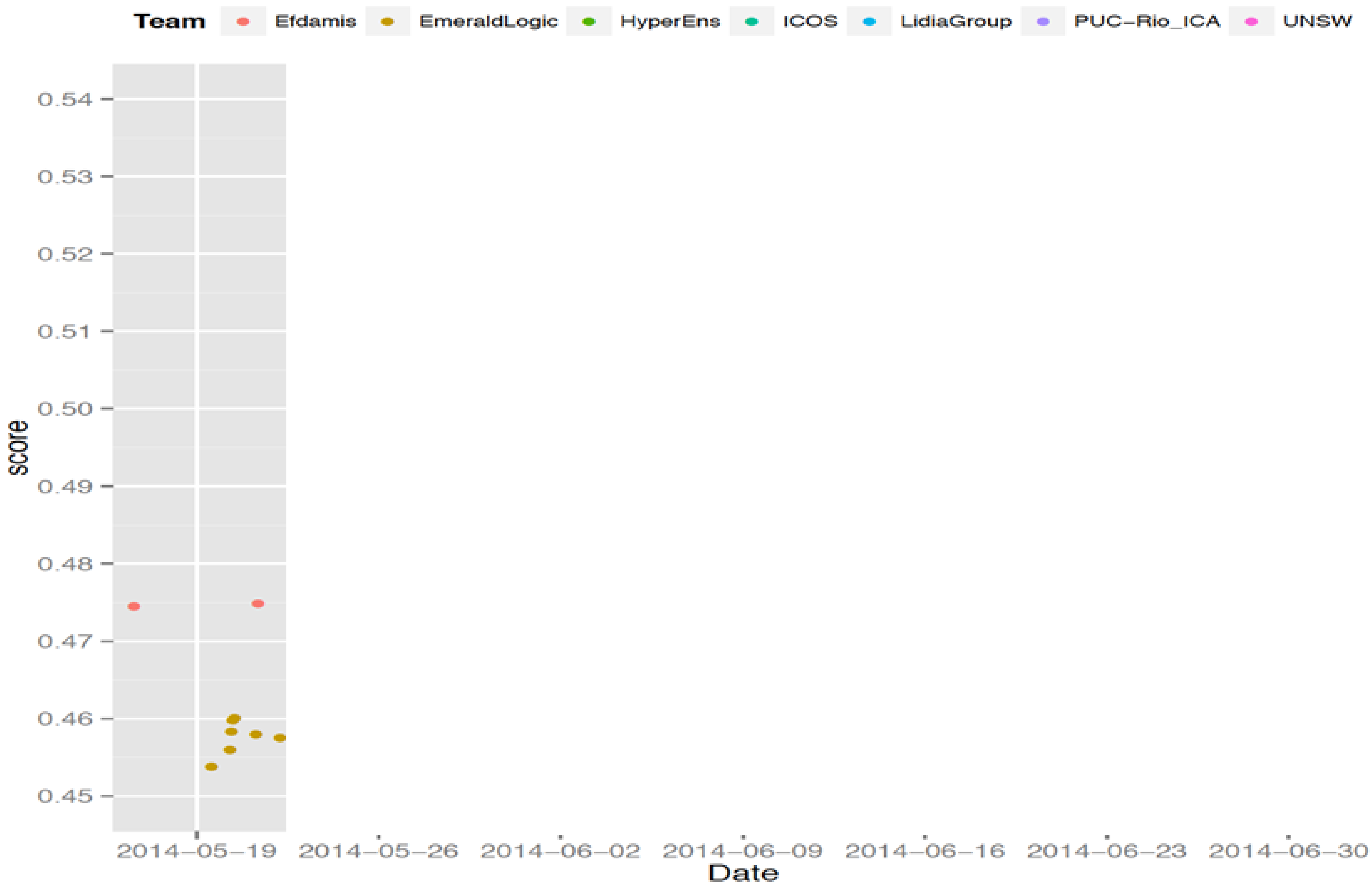
ECBDL'14 Big Data Competition 2014

Our approach:

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

2. Learning a model.
   - ❑ Random Forest

ECBDL’14 Big Data Competition



ECBDL’14 Big Data Competition

We initially focused on

- ❑ Oversampling rate: {100%}

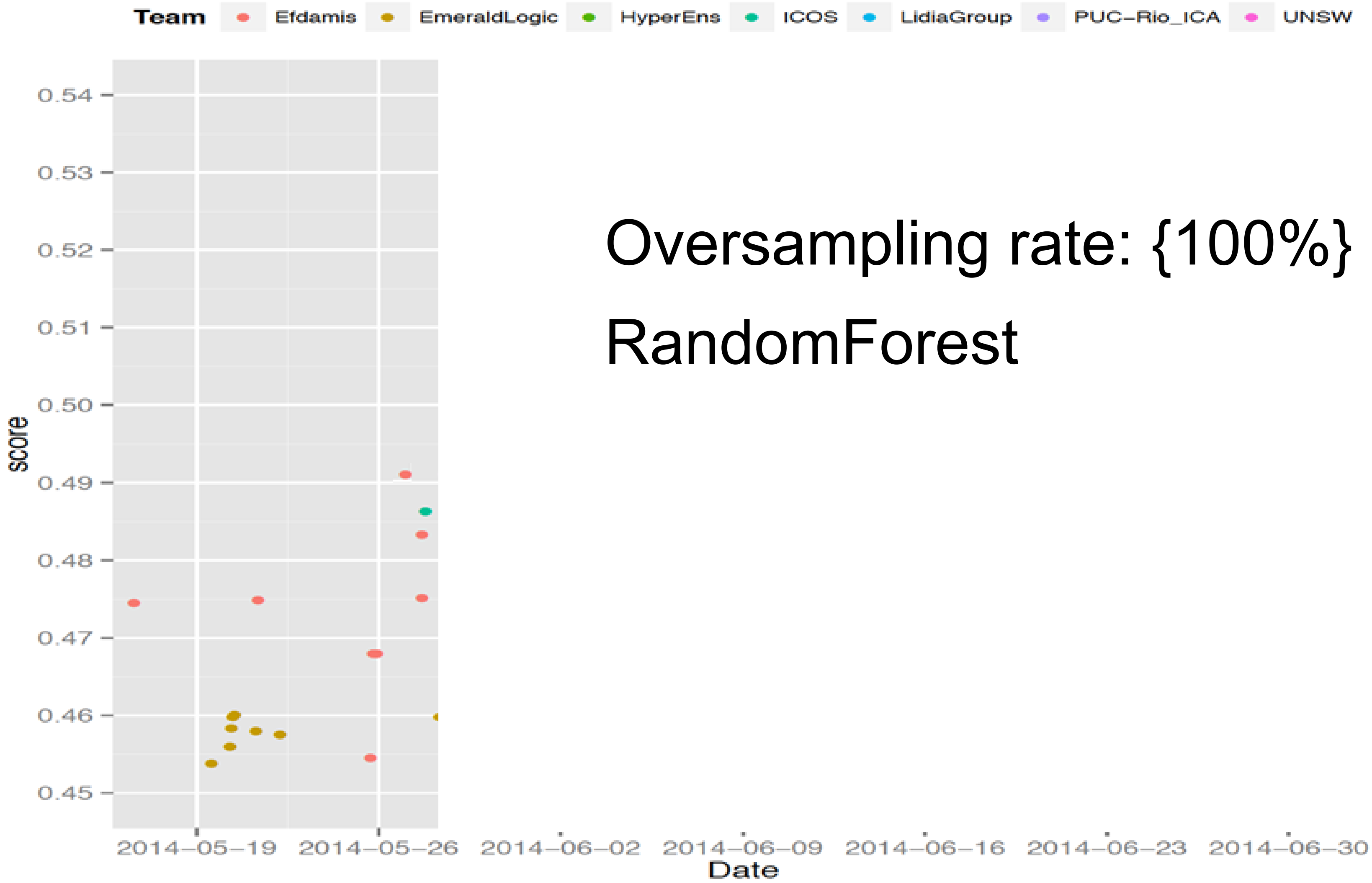
RandomForest:

- ❑ 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
190	0,635175	0,773308	0,491186
1024	0,627896	0,756297	0,474876
2048	0,624648	0,759753	0,474578

To higher mappers, the lowest TPR (relevant!)

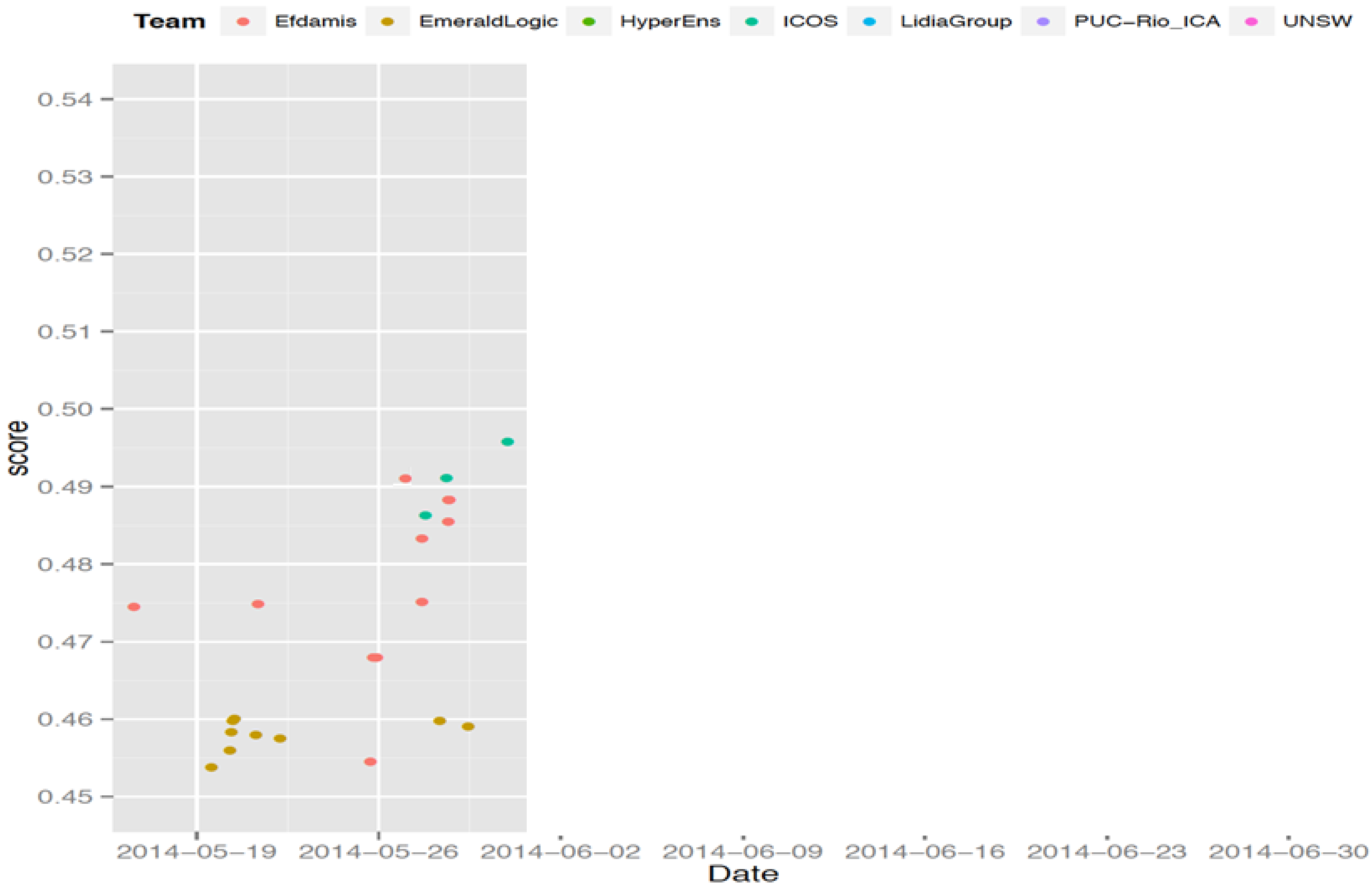
ECBDL'14 Big Data Competition



Oversampling rate: {100%}

RandomForest

ECBDL'14 Big Data Competition



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
190	**0,635175**	**0,773308**	**0,491186**

Very low TPR (relevant!)

How to increase the TPR rate?

Idea: To increase the ROS percentaje

ECBDL'14 Big Data Competition

How to increase the TPR rate?

Idea: To increase the ROS percentaje

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

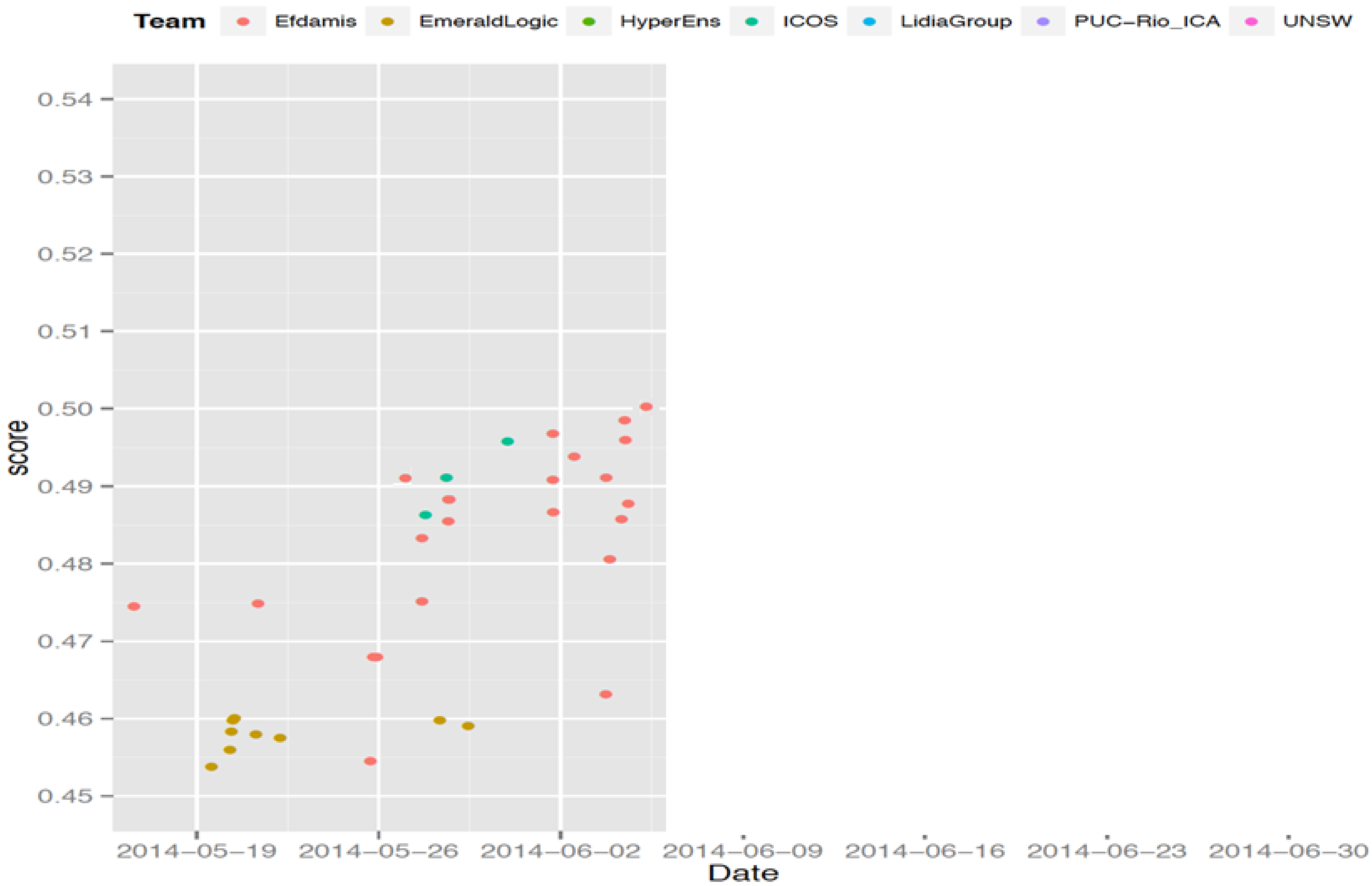
RandomForest:

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

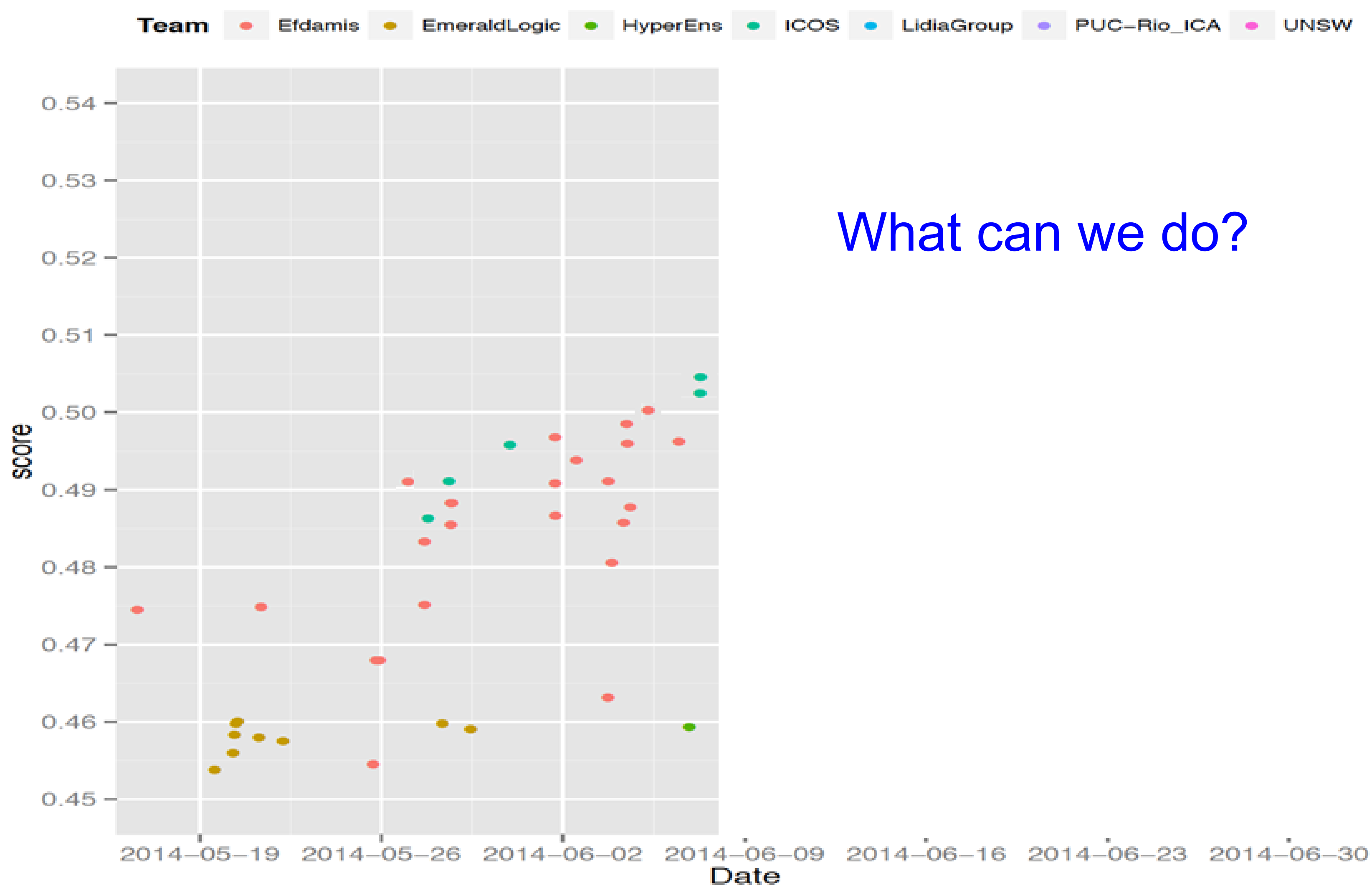
Algorithms	TPR	TNR	TNR*TPR Test
ROS+RF (RS: 100%)	**0.6351**	**0.7733**	**0.491186**
ROS+RF (RS: 105%)	0.6568	0.7555	0.496286
ROS+RF (RS: 110%)	0.6759	0.7337	0.495941
ROS+RF (RS: 115%)	**0.7041**	**0.7103**	**0.500175**
ROS+RF (RS: 130%)	0.7472	0.6609	0.493913

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

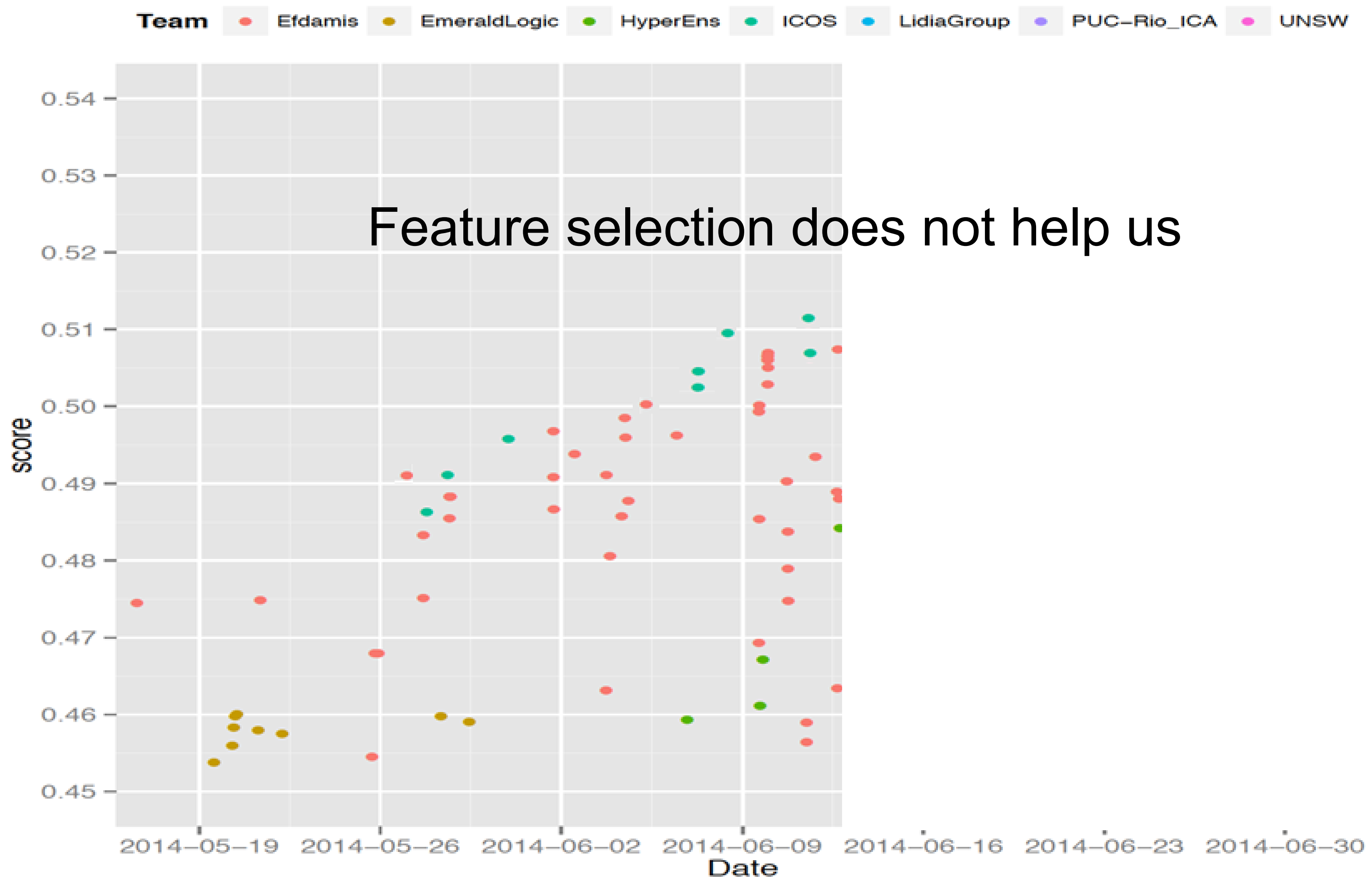
ECBDL’14 Big Data Competition 2014

Our approach:

1. Balance the original training data
   - Random Oversampling
   - (As first idea, it was extended)
2. Learning a model.
   - Random Forest
3. Detect relevant features.
   - Evolutionary Feature Selection

Classifying test set.

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
   - (As first idea, it was extended)

2. Learning a model.
   - Random Forest

3. Detect relevant features.
   - Evolutionary Feature Weighting

Classifying test set.

ECBDL'14 Big Data Competition

How to increase the performance?

Third component: MapReduce Approach for Feature Weighting

for getting a major performance over classes

Map Side

- ❑ Each map read one block from dataset.
- ❑ Perform an Evolutionary Feature Weighting step.
- ❑ Output: a real vector that represents the degree of importance of each feature.
- ❑Number of maps: 32768 (less than 1000 original data per map)

Reduce Side

- ❑ Aggregate the feature's weights
- ❑ A feature is finally selected if it overcomes a given threshold.
- ❑ Output: a binary vector that represents the final selection

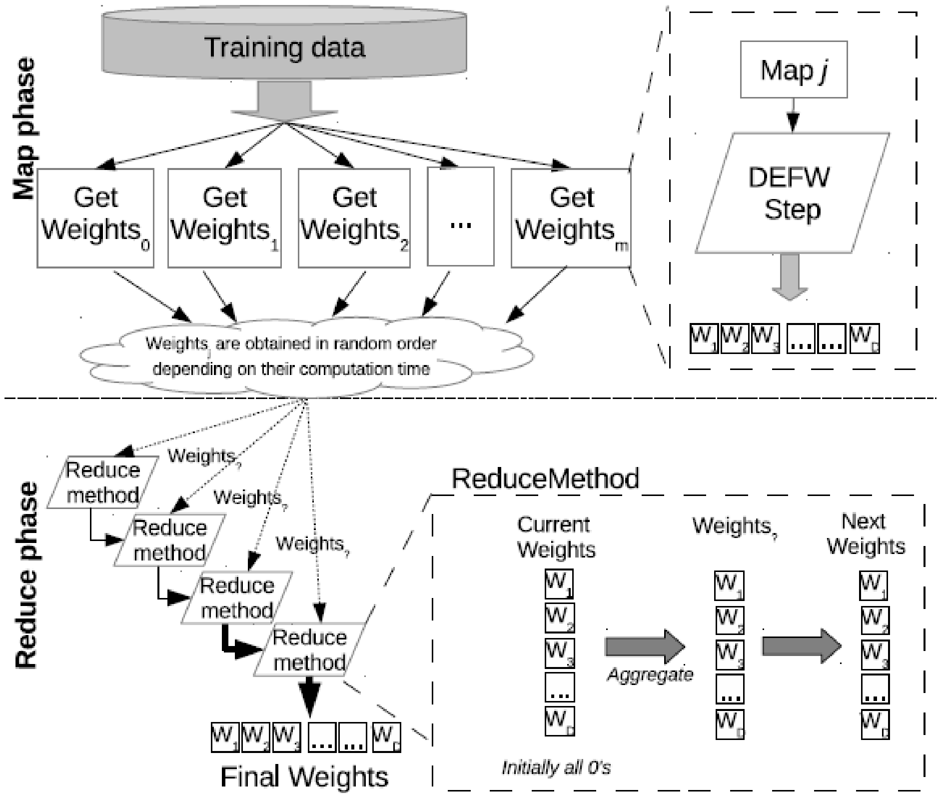
I. Triguero, J. Derrac, S. García, F. Herrera, Integrating a Differential Evolution Feature Weighting scheme into Prototype Generation. Neurocomputing 97 (2012) 332-343

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

Experimental study

Random Oversampling:

- Oversampling ratio. Analyzed values: {100 to 130)

Feature Weigthing:

- Threshold --> number of selected features.
- Set of features: {19, 63, 90, 146}
- Number of maps: 32768

RandomForest:

- Number of used features: {log NumFeatures, 2 * Log +1}
- Number of trees: {100}
- Number of maps: {32, 64,128, 190, 256, 512}

ECBDL’14 Big Data Competition

We investigate: The use of Evolutionary Feature Weighting.

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

	128 mappers			
Algorithms	**TNR*TPR Training**	**TPR**	**TNR**	**TNR*TPR Test**
ROS+RF (130% - Feature Weighting 19)	0.621638	0.684726	0.735272	0.503459
ROS+RF (115% - Feature Weighting 19)	0.628225	0.674569	0.750184	0.506051
ROS+RF (100% - Feature Weighting 19)	0.635029	0.629397	0.784132	0.493531
ROS+RF (130% - Feature Weighting 63)	**0.634843**	**0.683800**	**0.756926**	**0.517586**
ROS+RF (115% - Feature Weighting 63)	**0.639319**	**0.677015**	**0.764589**	**0.517638**
ROS+RF (100% - Feature Weighting 63)	0.648723	0.638567	0.794595	0.507402

	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**

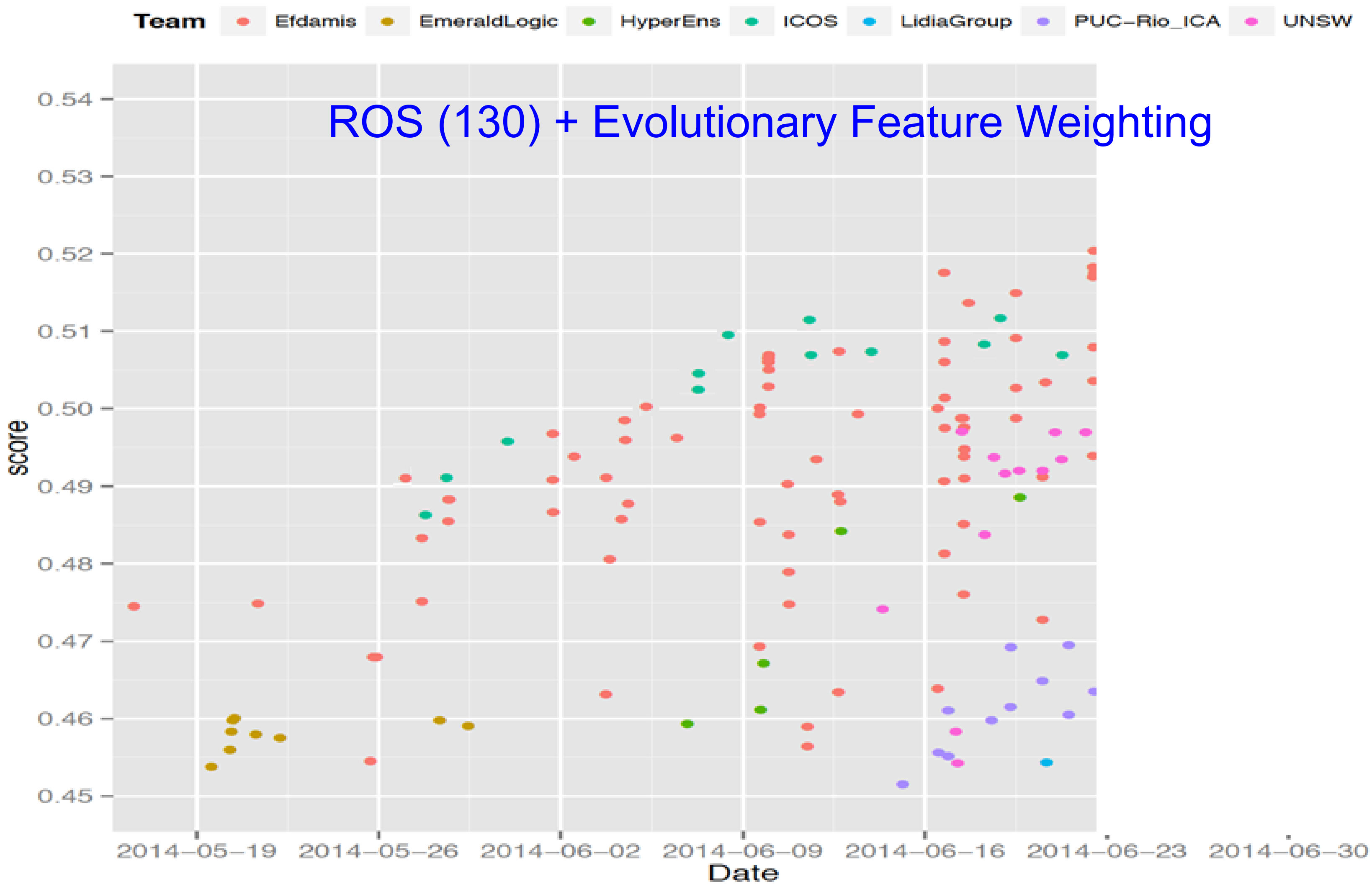
ECBDL'14 Big Data Competition

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

More features with diffferent Maps configuration

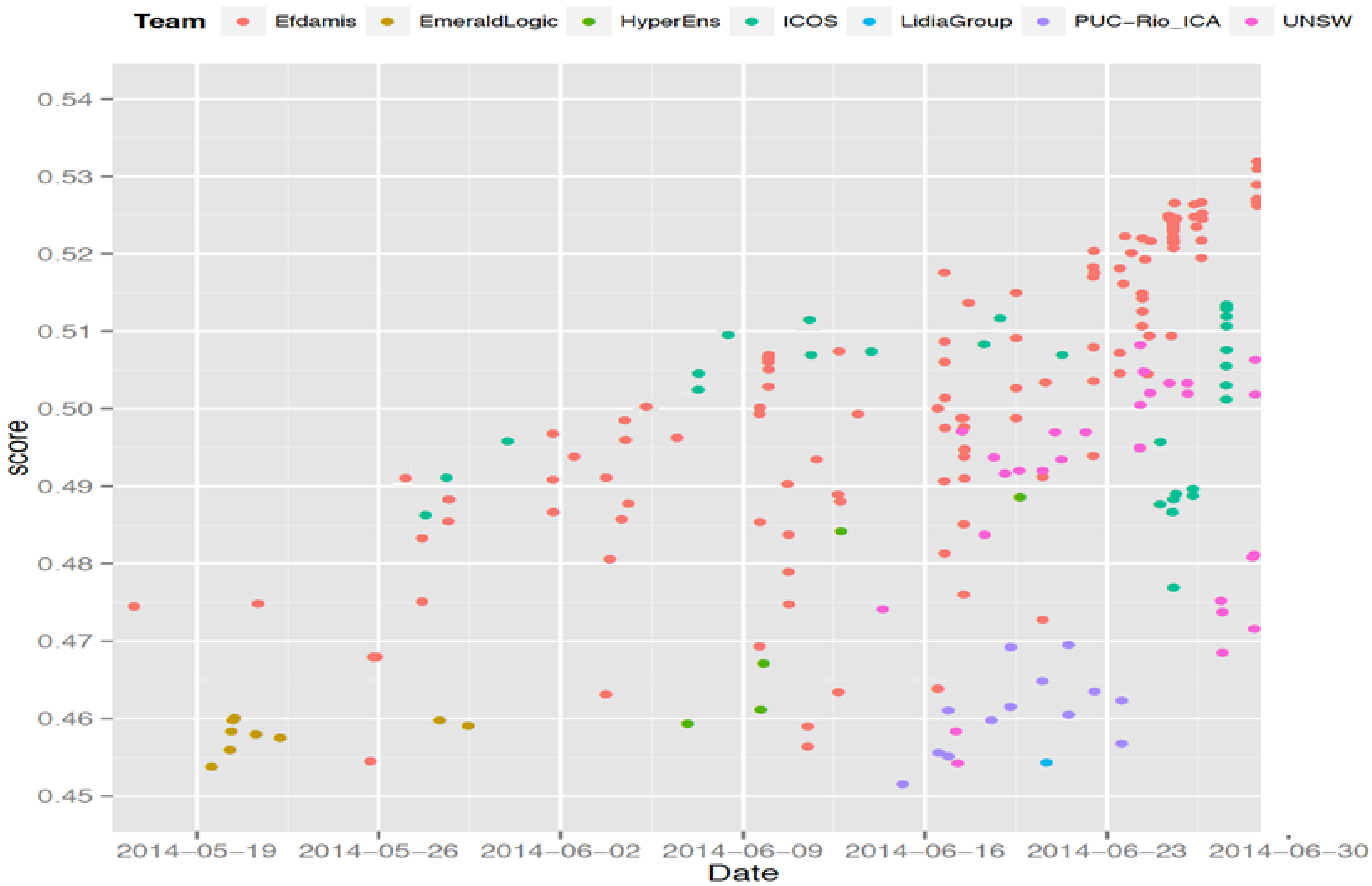
Algorithms	190 mappers TNR*TPR Training	TPR	TNR	TNR*TPR Test
ROS+ RF (140%+ FW 90+25f+200t)	**0.629273**	**0.721652**	**0.729740**	**0.526618**

Algorithms	64 mappers 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**

64 mappers and we got 0.53

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

ECBDL'14 Big Data Competition



ECBDL'14 Big Data Competition

Current state:

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

Our knowledge:

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

The less number of maps, the less TPR and the high TNR (high accuracy).

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

4 days to finish the competion:

Can we take decisions to improve the model?

ECBDL'14 Big Data Competition

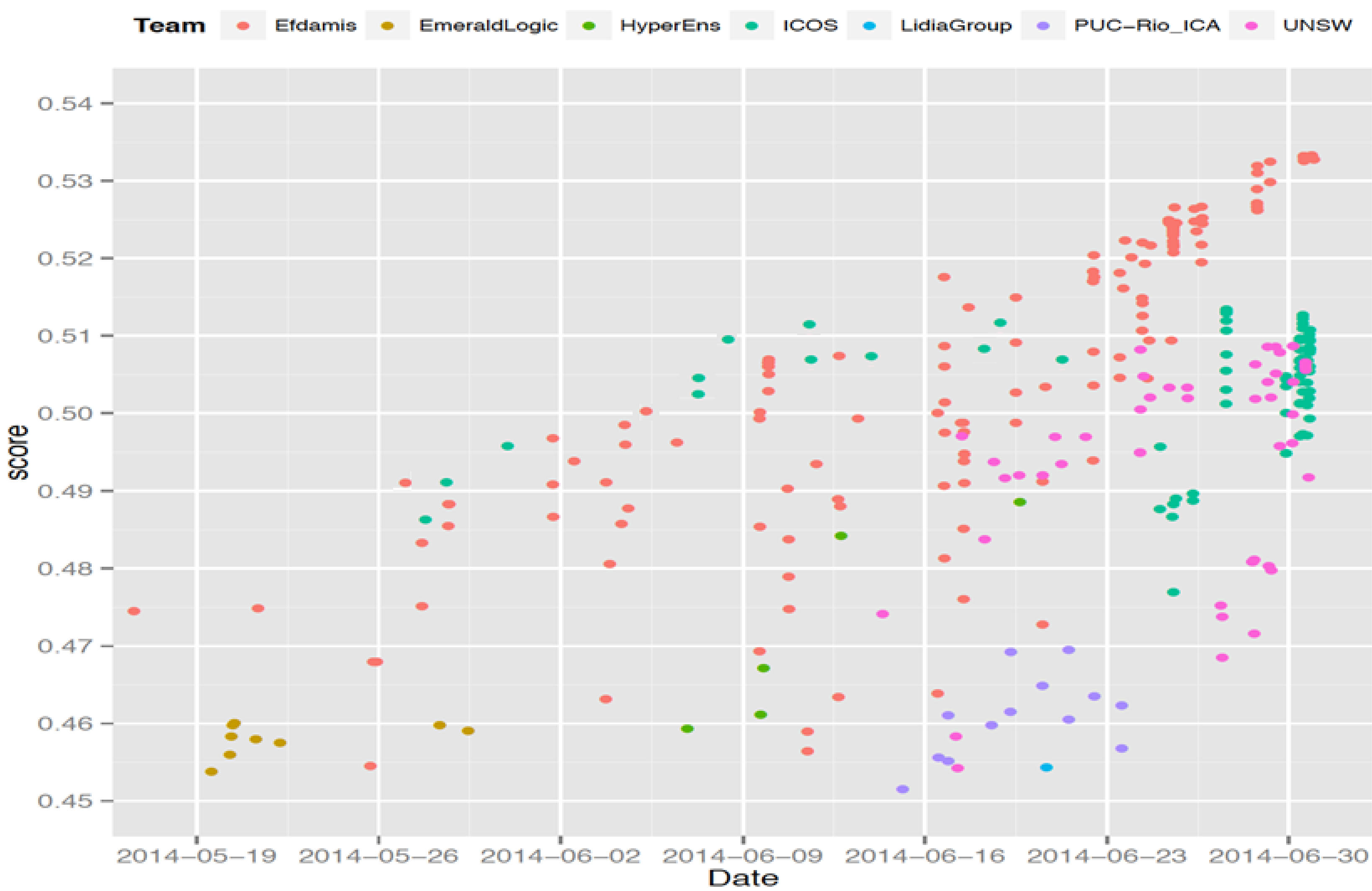
Last decision: We investigated to increase ROS until 180% with 64 mappers

Algorithms	64 mappers 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



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
Test2	0.632009	0.735545	0.733808	0.464871

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

GECCO

ECBDL'14 Big Data Competition
Final comments

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: **ROSEFW-RF algorithm**

	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

ECBDL'14 Big Data Competition

Final comments

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**

Algorithms	64 mappers			
	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

To increase ROS and to use Evolutionary feature weighting were two good decisions for getting the first position

ECBDL'14 Big Data Competition
Final comments

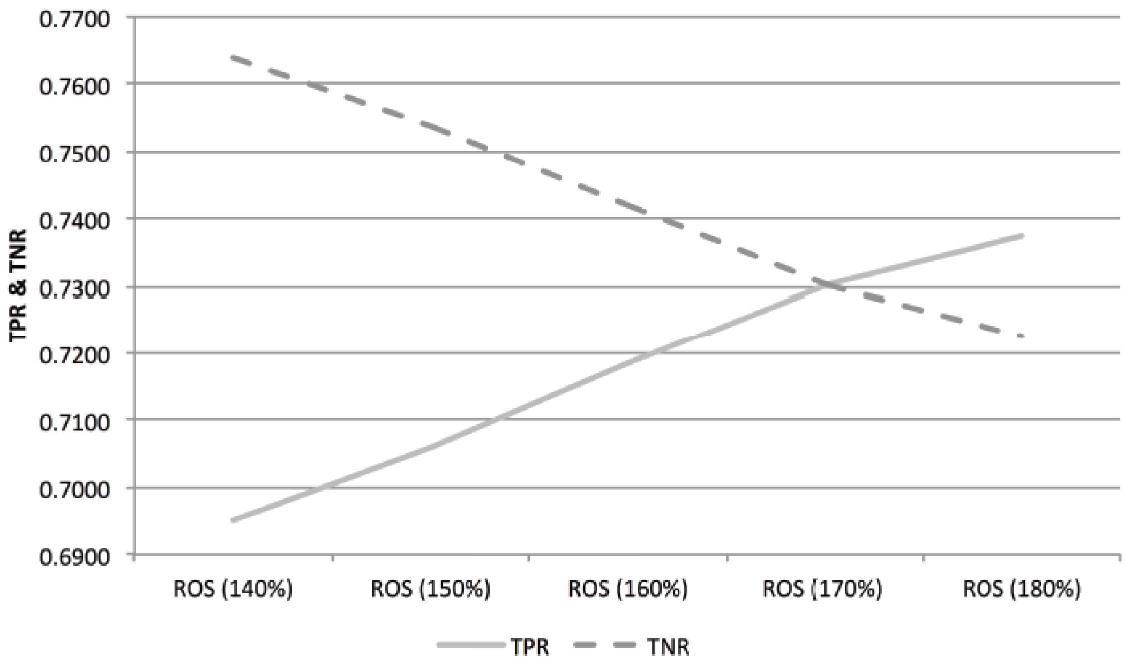


Figure 8: TPR vs. TNR varying the ROS percentage

Experiments with 64 maps

ROS 170 – 85 replications of the minority instances

Remember the initial problem. Lack of density of the minority class

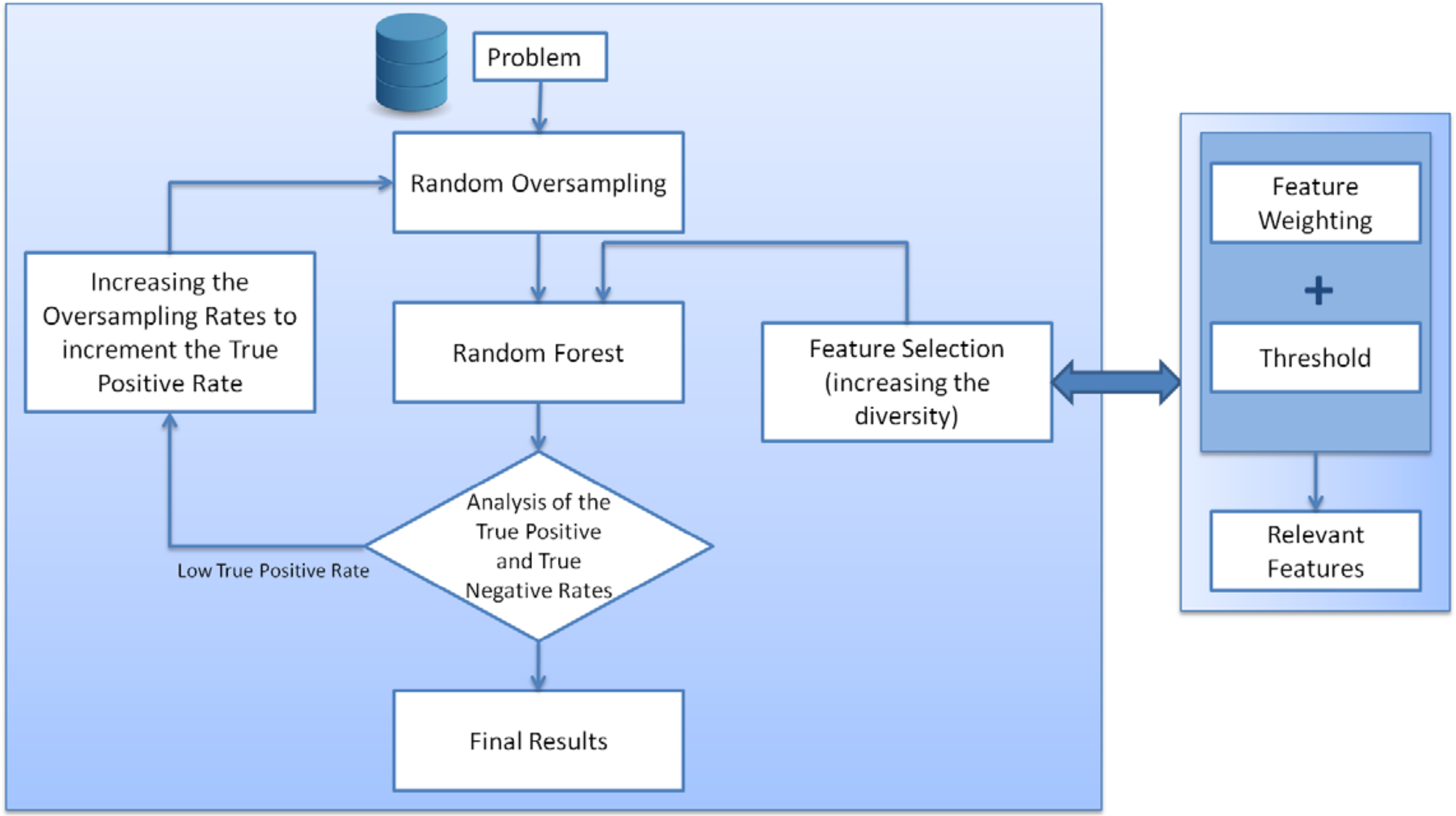
ECBDL’14 Big Data Competition

Final comments

Team Name	Learning strategy	Computational Infrastructure
Efdamis	Oversampling+FS+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

Our algorithm: ROSEFW-RF



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, 2015, In press.

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

Outline

- Big Data. Big Data Science. Data Preprocessing
- Why Big Data? MapReduce Paradigm. Hadoop Ecosystem
- Big Data Classification: Learning algorithms
- Data Preprocessing
- Big data Preprocessing
- Big Data Classification: Imbalanced classes and data preprocessing
- Challenges and Final Comments

Final Comments

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

Big Data: A small subset of algorithms

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

Final Comments

Some Challenges on Big Data Preprocessing

- **Clean Big Data**
  - Noise in data distorts
    - *Computation results*
    - *Search results*
  - Need automatic methods for "cleaning" the data
    - *Duplicate elimination*
    - *Quality evaluation*
- **Missing values**
  - Missing values management
- **Big Data Reduction**
  - To improve the efficiency in the big data analytics.
  - Quality data for quality models in big data analytics
- **Computing Model**
  - Accuracy and Approximation
  - Efficiency

Final Remarks

Big Data Preprocessing

Quality decisions must be based on quality big data!

Big data preprocessing methods are necessary to improve the quality of the processes of big data analytics.

Final Comments

Quality decisions must be based on quality big data!

Data Mining methods for Big Data Preprocessing