Big Data II
Curso 2017-18

La asignatura “Big Data II” comprende el estudio de tecnologías de analítica y procesamiento de datos masivos (big data).

Big Data Analytics

Francisco Herrera
herrera@decsai.ugr.es

Jesús Maillo
jesusmh@decsai.ugr.es

Diego García
djgarcia@decsai.ugr.es

Contenido

- M6, Mar. 17:30 – T1 Introducción a Big data y Big Data Analytics
- M8, Jue. 15:30 – P1. Spark. Spark Packages. Algoritmo kNN-IS
- M13, Ma4. 17:30 P2. Plataforma Spark Apache
- A03, Mar. 17:30 T2 Big Data Analytics: Algoritmos exactos vs aproximados
- A05, Jue. 15:30 T3 Big Data Analytics: Clasificación
- A10, Mar. 17:30 P3 MLLib y Spark Package. Algoritmo de ensemble
- A12, Jue. 15:30 T4 Smart Data: Big data Preprocessing I
- A17, Mar. 17:30 T5 Smart Data: Big data Preprocessing II
- A19, Jue. 15:30 P4 MLLib y Spark Package. Algoritmos de ruido y selección de instancias

Practicas con el cluster: hadoop.ugr.es

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

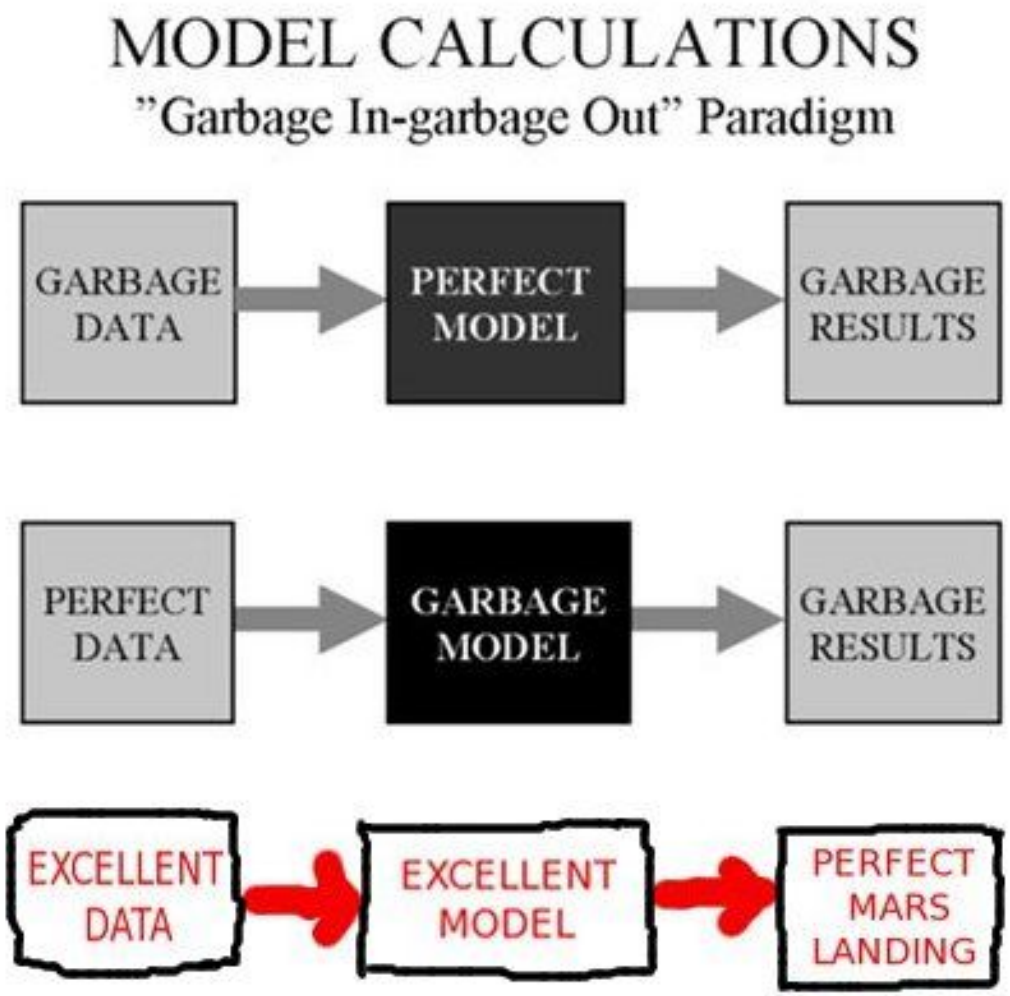
From Big Data to Smart Data: Big Data Preprocessing

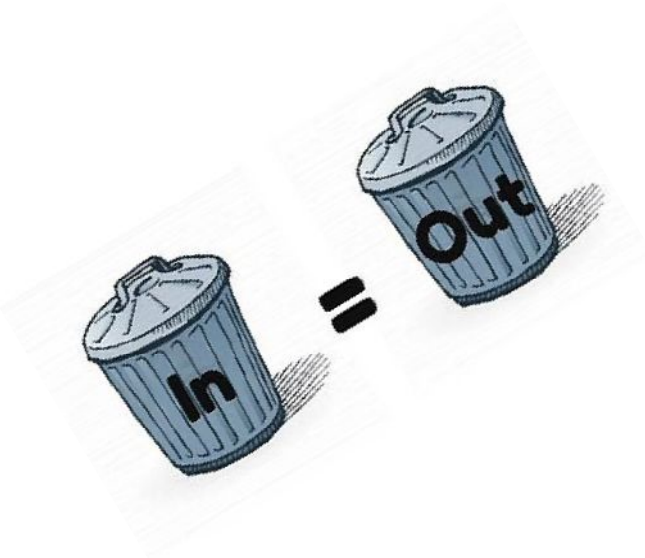
1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Instance Reduction
6. Imperfect Data
7. Discretization
8. Feature Selection
9. Final Comments

From Big Data to Smart Data: Big Data Preprocessing

1. **Smart Data: Towards Quality Data**
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. Discretization
9. Final Comments

Big Data: Towards Quality Data





Big Data: Towards Quality Data

An important statement

Challenges of Big Data analysis

Jianqing Fan[1,*], Fang Han[2] and Han Liu[1]

National Science Review
1: 293–314, 2014
doi: 10.1093/nsr/nwt032
Advance access publication 6 February 2014

What are the challenges of analyzing Big Data?

Big data are characterized by high dimensionality and large sample size. These two features raise three unique challenges: (i) high dimensionality brings noise accumulation, spurious correlations and incidental

Noise accumulation

Spurious correlation

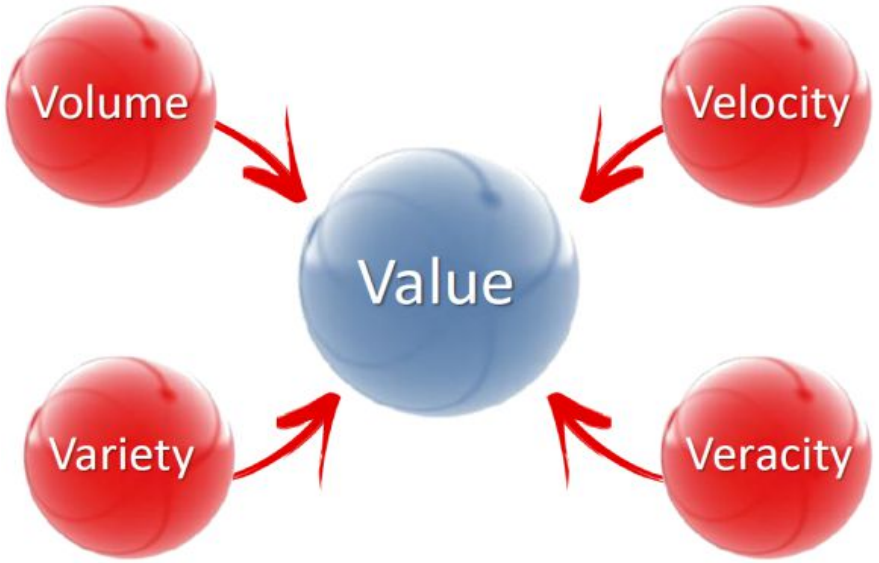
Large sample brings

Duplicity

Redundancy

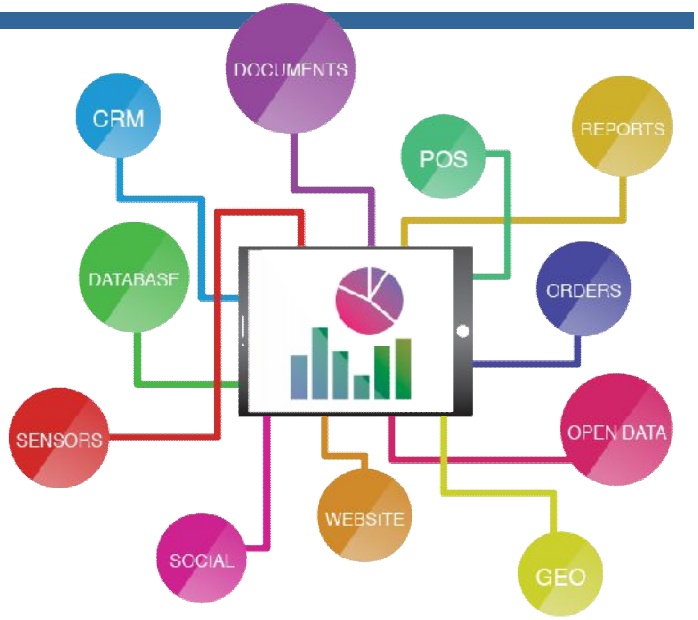
Smart Data

- RECALL: Big data as a concept is defined around five aspects:
- Data volume,
- Data velocity,
- Data variety,
- Data veracity and
- Data value.



Smart Data

a) While the volume, variety and velocity aspects refer to data generation process and how to capture and store the data,



b) Veracity and value aspects deal with the quality and the usefulness of the data leading to the point.

Smart Data

Smart Data (veracity and value) aims to filter out the noise and hold the valuable data, which can be effectively used by enterprises and governments for planning, operation, monitoring, control, and intelligent decision making.

What makes data smart? Three key attributes do for data to be smart, it must be accurate, actionable, and agile.

Big Data (Analytics) → Smart Data

The key is to explore how Big Data can become Smart Data.

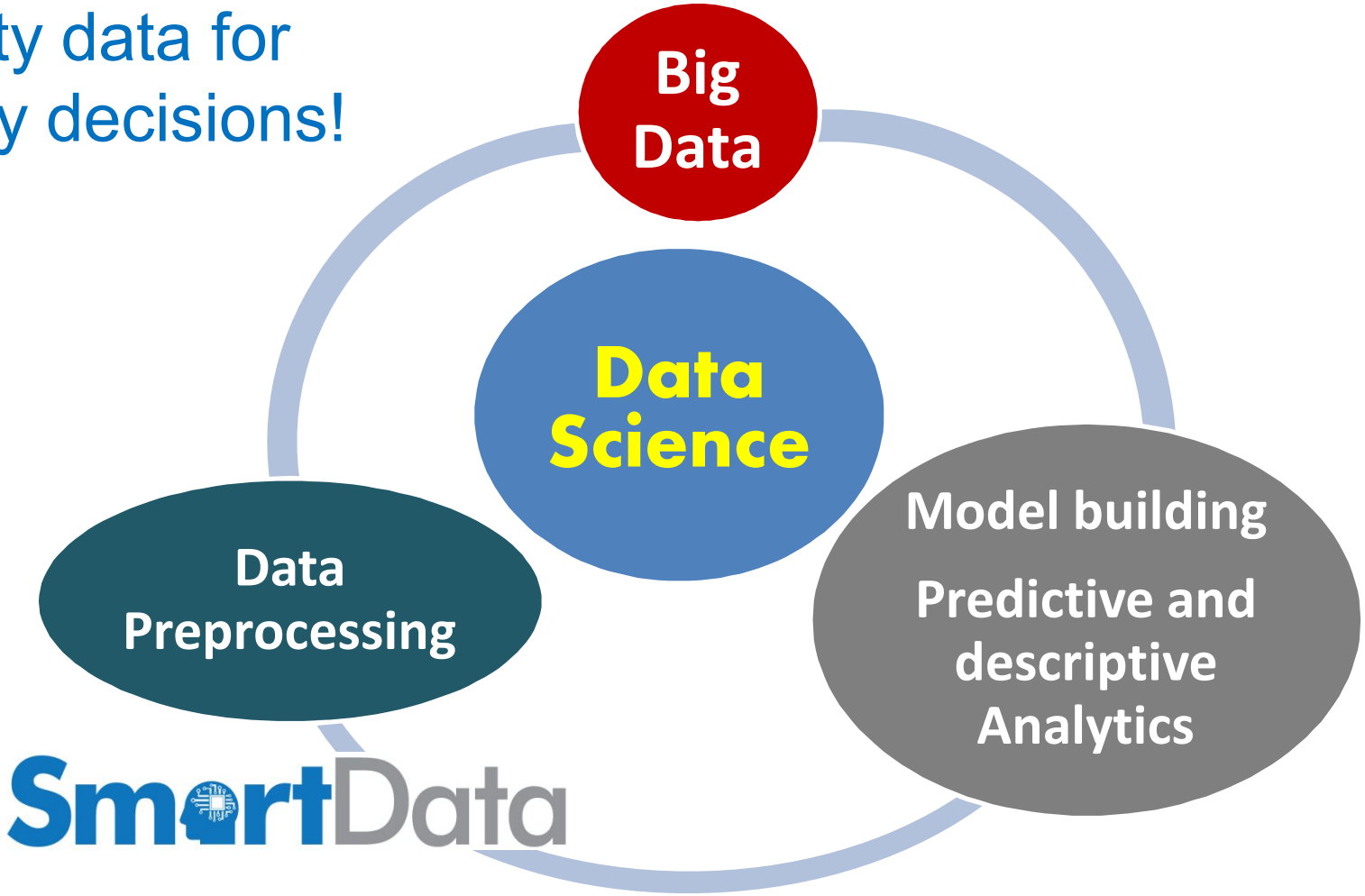
"Here's a list of 100,000 warehouses full of data. I'd like you to condense them down to one meaningful warehouse."

BIG DATA Preprocessing

Big Data
+ Analytics
= Smart Data

Big Data (Analytics) → Smart Data

Quality data for
quality decisions!



From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. **Data Preprocessing. Importance**
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. Discretization
9. Final Comments

Data Preprocessing

D. Pyle, 1999, pp. 90:

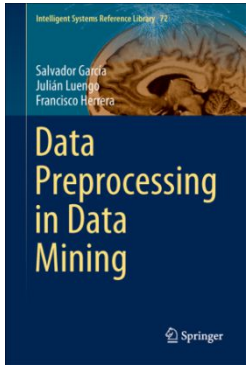
"The fundamental purpose of *data preparation* is to manipulate and transform raw data so that the information content enfolded in the data set can be exposed, or made more easily accesible."

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

Data Preprocessing

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

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



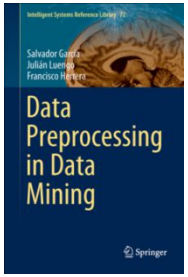
S. García, J. Luengo, F. Herrera
Data Preprocessing in Data Mining
Springer, January 2015
Website:
http://sci2s.ugr.es/books/data-preprocessing

Data Preprocessing

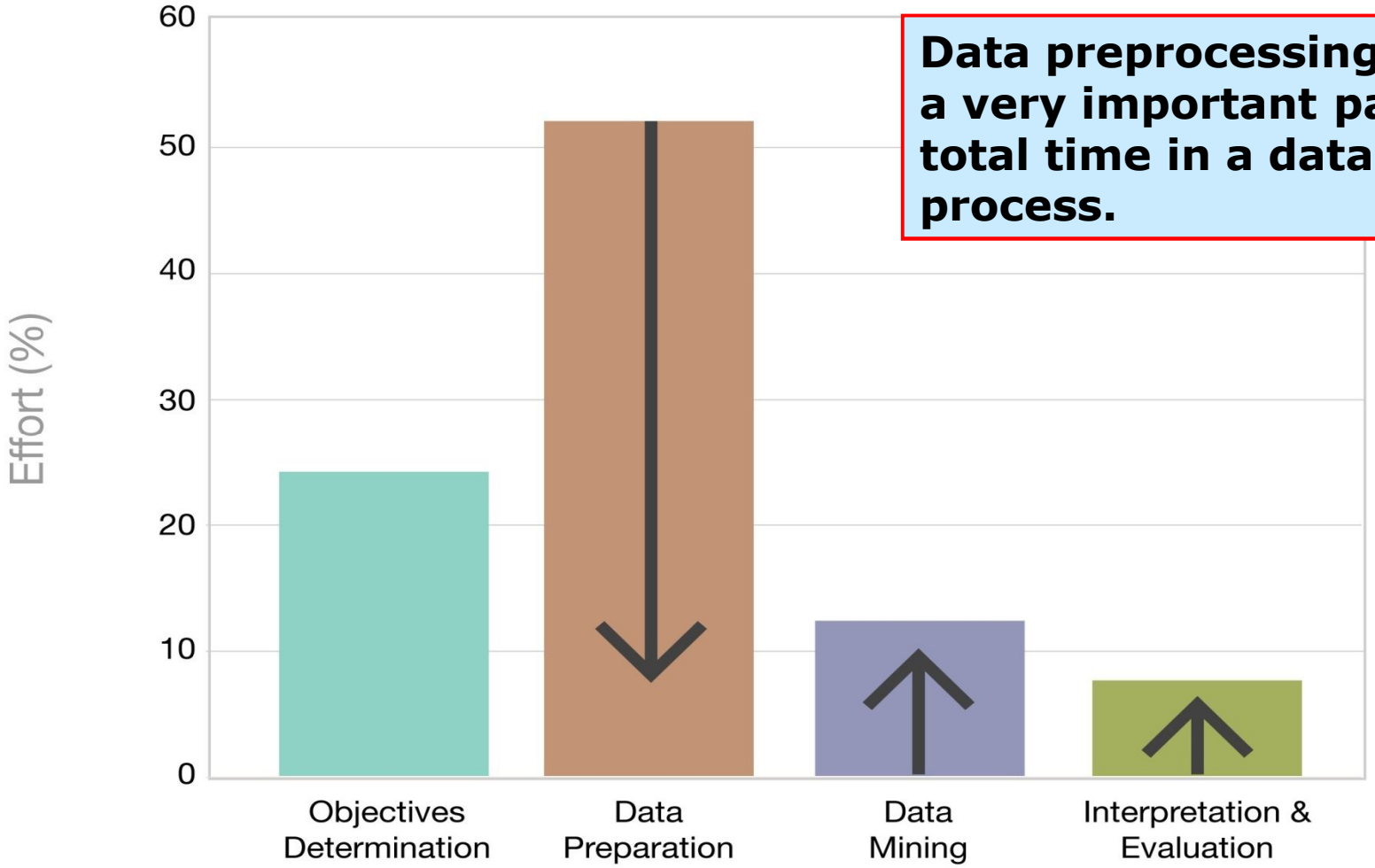
S. García, J. Luengo, F. Herrera, 2015, Preface viii:

There are many advantages that data preprocessing provides:

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



Data Preprocessing



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

Data Science Process

Data Processing

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

Data Preprocessing

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

> 70% time!

Data Analytics

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

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.

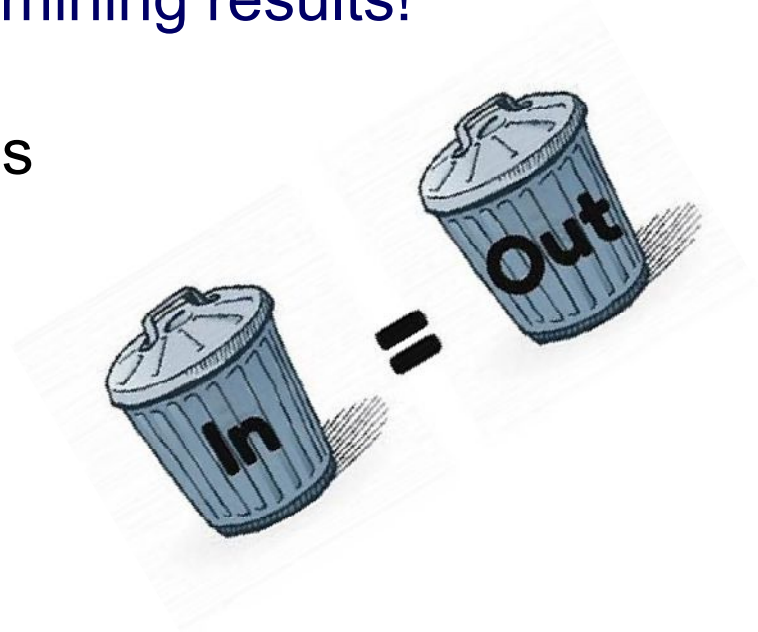
   This performing includes Data Reduction techniques:
   Feature selection, sampling or instance selection, discretization.

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!

From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. **Data Preparation and Data Reduction**
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. Discretization
9. Final Comments

Data Preprocessing

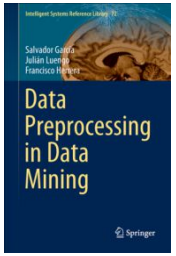
What is included in data preprocessing?

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

"Data preprocessing includes data preparation, compounded by integration, cleaning, normalization and transformation of data; and data reduction tasks; such as feature selection, instance selection, discretization, etc.

…

The result expected after a reliable chaining of data preprocessing tasks is a final dataset, which can be considered correct and useful for further data mining algorithms."



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

Data Preprocessing

What is included in data preprocessing?
Data preparation versus Data Reduction

We refer to **data preparation** as the set of techniques that initialize the data properly to serve as input for a certain DM algorithm.

Data reduction comprises the set of techniques that, in one way or another, obtain a reduced representation of the original data.

- **For us, the distinction of data preparation techniques is those that are needed to appropriately suit the data as input of a Data Mining task. This means that if data preparation is not properly conducted, the Data Mining algorithms will not be run or will surely report wrong results after running.**
- **In the case of data reduction, the data produced usually maintains the essential structure and integrity of the original data, but the amount of data is downsized.**

Data Preprocessing

What is included in data preprocessing?

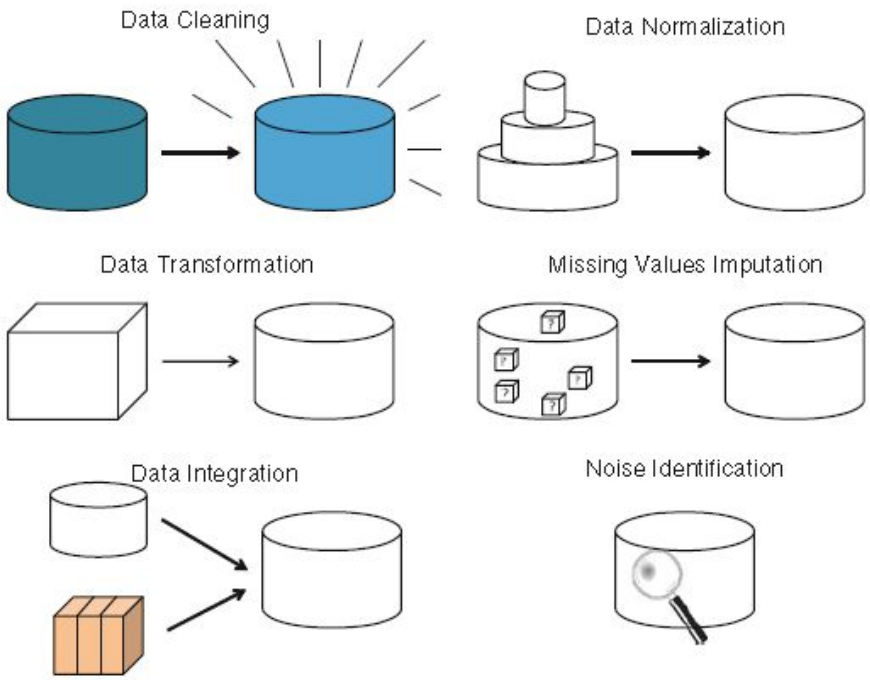


Fig. 1.3 Forms of data preparation

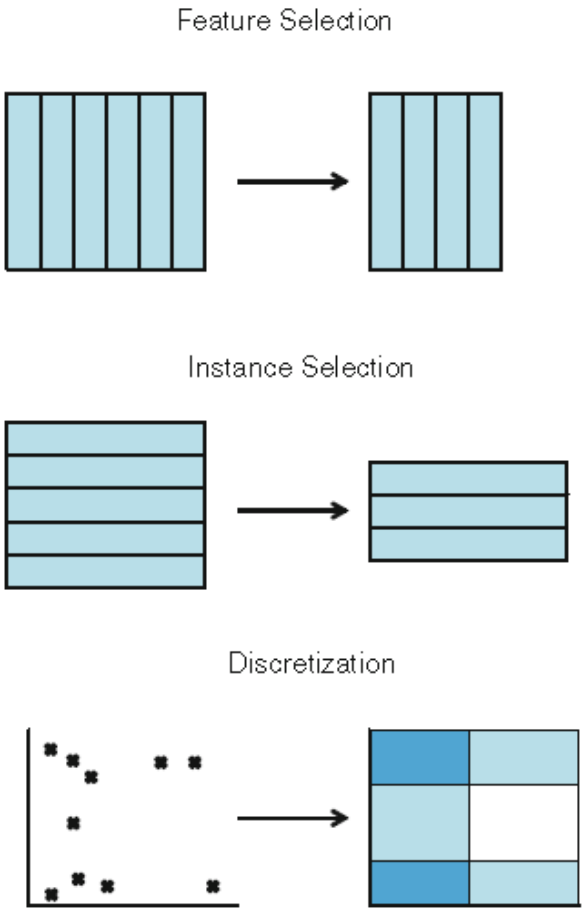


Fig. 1.4 Forms of data reduction

Snapshot on Data Preparation

Data preparation is normally a mandatory step.
It converts prior useless data into new data that fits a Data Mining process.

What are the basic issues that must be resolved in data preparation? Here, we provide a list of questions accompanied with the correct answers involving each type of process that belongs to the data preparation family,

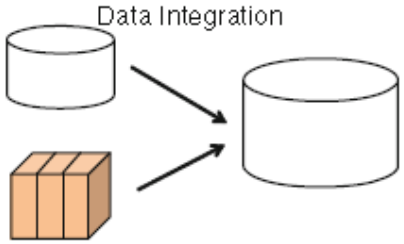
- How do I clean up the data?
- How do I provide accurate data?
- How do I incorporate and adjust data?
- How do I unify and scale data?
- How do I handle missing data?
- How do I detect and manage noise?.

Snapshot on Data Preparation

- How do I clean up the data?—**Data Cleaning**

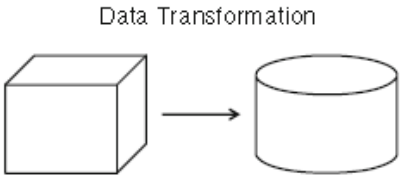
- How do I incorporate and adjust data?

—**Data Integration**



How do I provide accurate data?

—**Data Transformation**



- How do I unify and scale data?

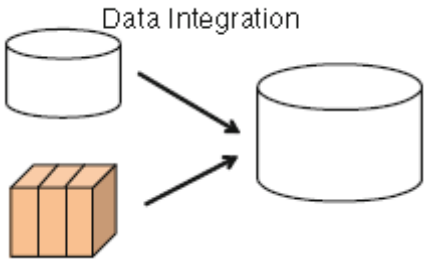
—**Data Normalization**

Data Cleaning

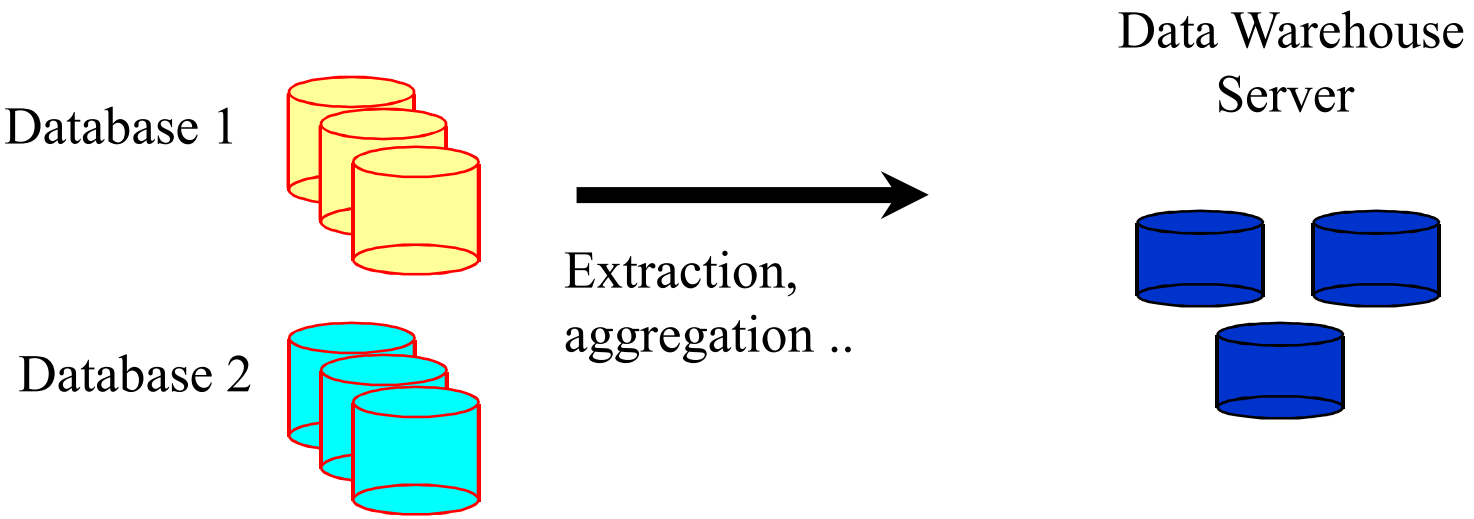
Data Cleaning: Inconsistent data

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

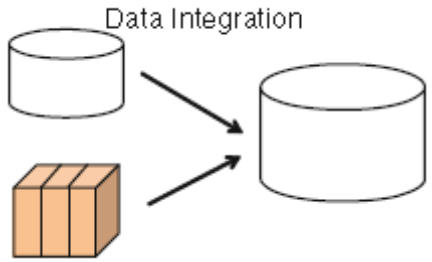
Data Integration



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



Data Integration



Examples

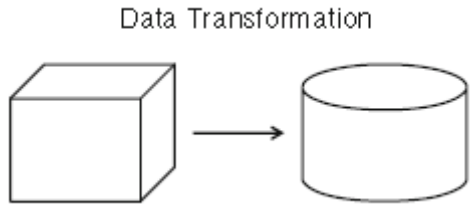
- Different scales: Salary in dollars versus euros (€)

- 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 transformation



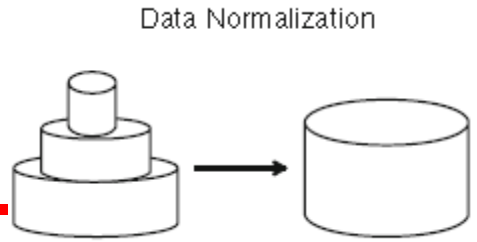
- Objective: To transform data in the best way possible to the application of Data Mining algorithms.

- 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.

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

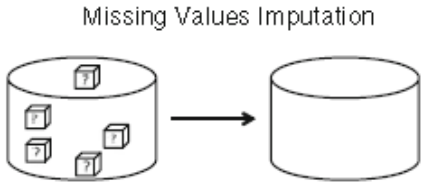
$$v' = \frac{v - \min_A}{\max_A - \min_A}(new_{\max_A} - new_{\min_A}) + new_{\min_A}$$

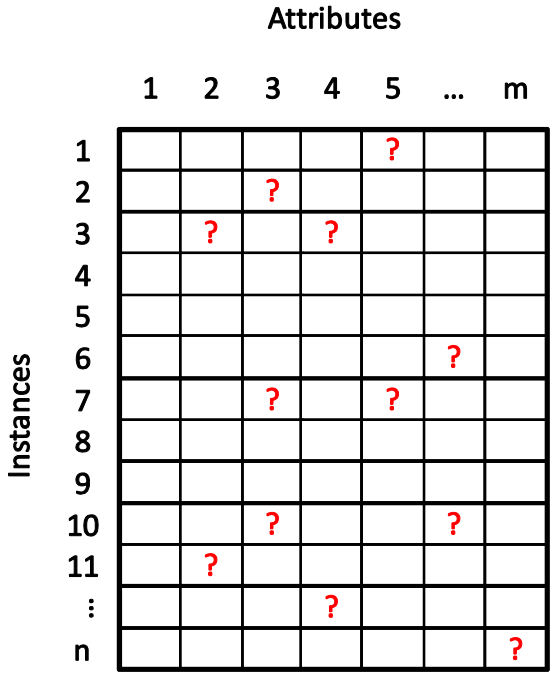
The relationships among original data are maintained.

Snapshot on Data Preparation

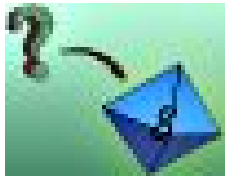
- How do I handle missing data?

  —**Missing Data Imputation**





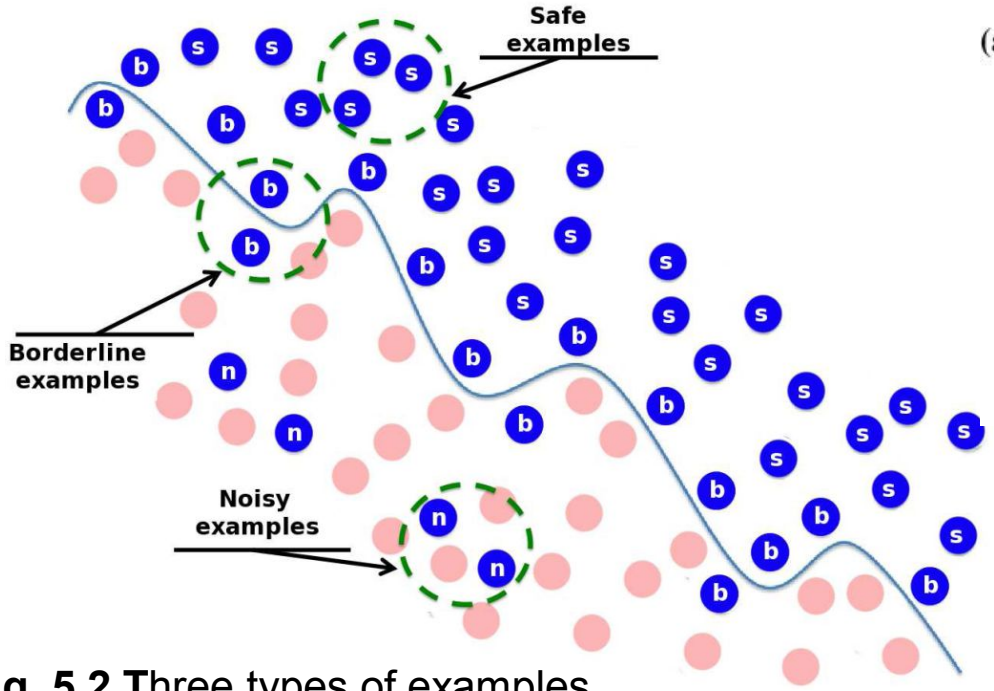
WEBSITE: http://sci2s.ugr.es/MVDM/



Snapshot on Data Preparation

- How do I detect and manage noise?

—Noise Identification (filtering and reparing)



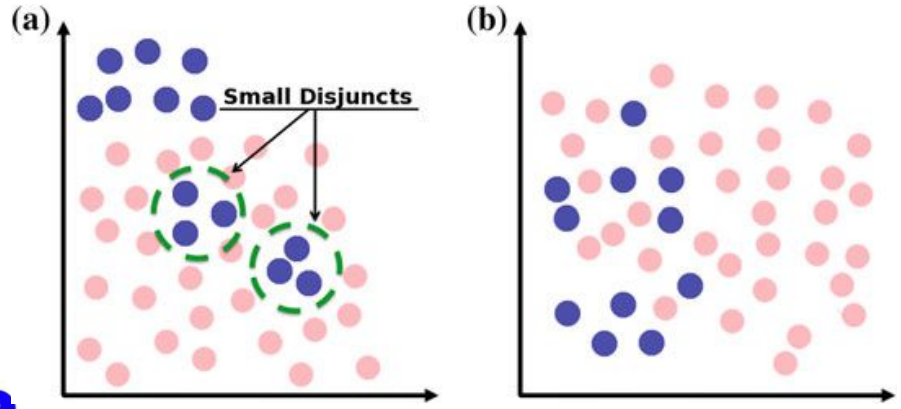
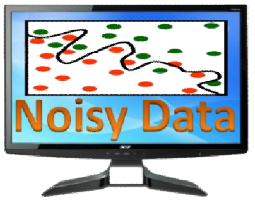


Fig. 5.1 Examples of the interaction between classes: a) small disjuncts and b) overlapping between classes

Fig. 5.2 Three types of examples 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

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



Snapshot on Data Reduction

Data reduction comprises the set of techniques that, in one way or another, obtain a reduced representation of the original data.

Can the original data be used, without applying a data reduction process, as input of a Data Mining process?
The answer is yes, but other major issues must be taken into account, being just as crucial as the issues addressed by data preparation.

Hence, at a glance, it can be considered as an optional step. However, this affirmation may be conflictive.

Snapshot on Data Reduction

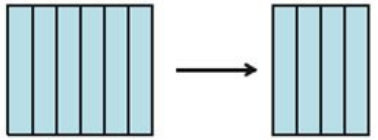
What are the basic issues that must be resolved in data reduction?

Again, we provide a series of questions accompanied with the correct answers involving each type of process that belongs to the data reduction technique.

- How do I reduce the dimensionality of data?
- How do I remove redundant and/or conflictive examples?
- How do I simplify the domain of an attribute?
- How do I fill in gaps in data?

Snapshot on Data Reduction

- How do I reduce the dimensionality of data?
  —**Feature Selection**



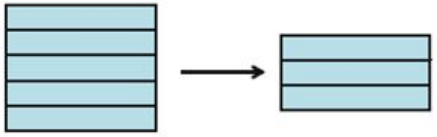
Var. 1. Var. 5 Var. 13

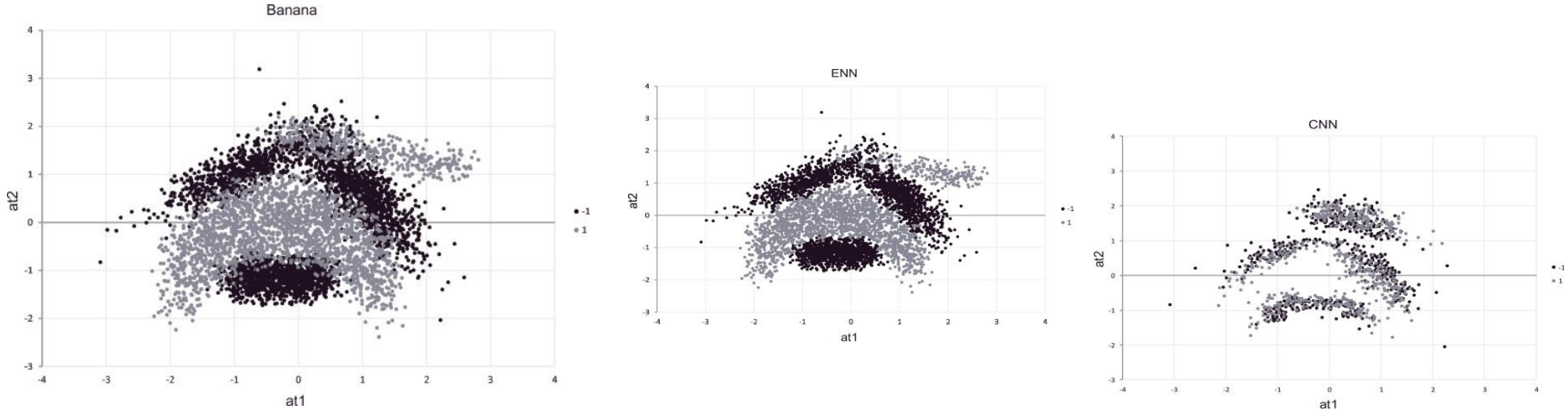
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
C	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
D	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
E	0	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0
F	1	1	1	0	1	1	0	0	1	0	1	0	0	1	0	0

- 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.

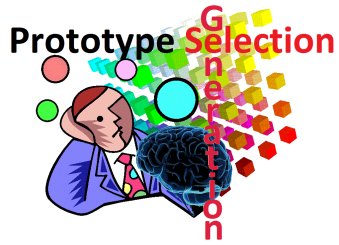
Snapshot on Data Reduction

- How do I remove redundant and/or conflictive examples?
  —**Instance Selection**
  (prototype Selection vs training set selection)



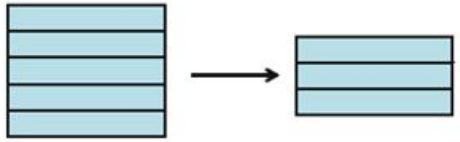


- Less data → algorithms couls learn quickly
- Higher accuracy → the algorithm better generalizes
- Simpler results → easier to understand them

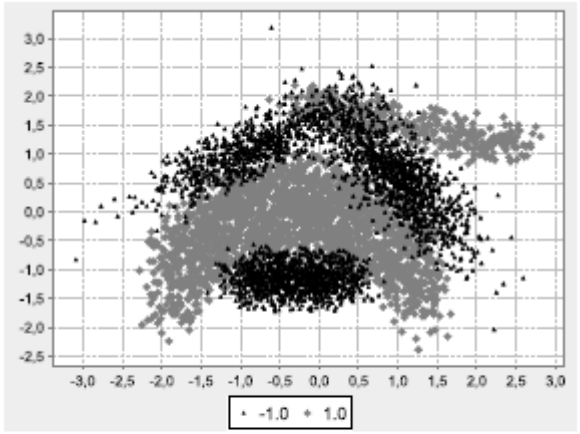


WEBSITE: http://sci2s.ugr.es/pr/index.php

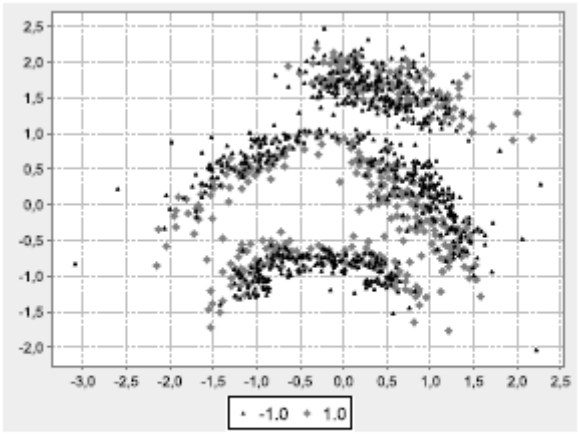
Snapshot on Data Reduction



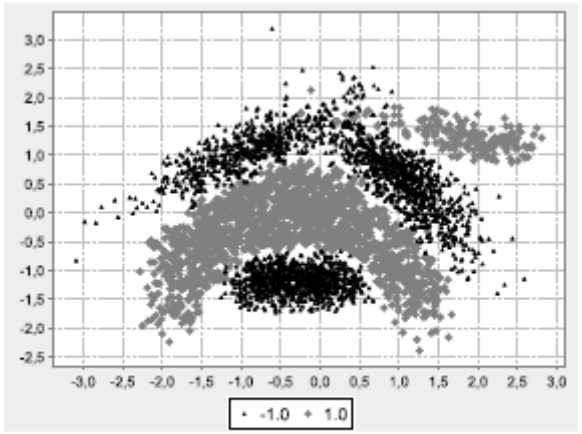
Graphical illustrations Instance Selection algorithms (Condesation vs Edition)



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



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



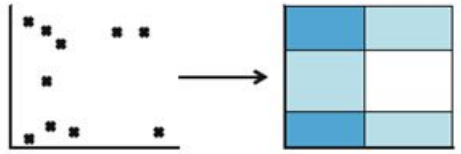
(h) AllKNN $(0.1758, 0.8934, 0.7831)$

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).

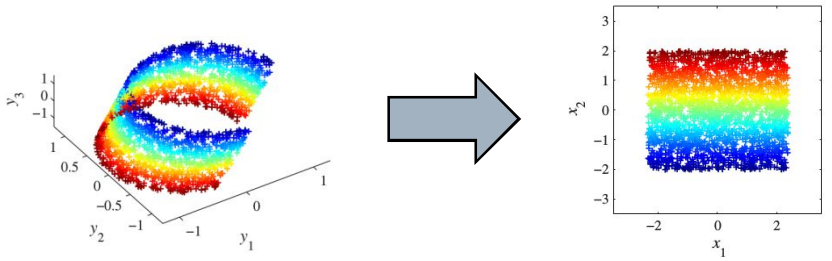
Snapshot on Data Reduction

- How do I simplify the domain of an attribute?
  —**Discretization**



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

- How do I fill in gaps in data?
  —**Feature Extraction and/or Instance Generation**



Snapshot on Data Preprocessing

Summary

- Data preprocessing is a big issue for data mining
- Data preprocessing includes
  - Data preparation: cleaning, imperfect data, transformation …
  - Data reduction
- The cooperation between data mining algorithms and data preprocessing methods is an interesting/active area.
- There are also specific preprocessing approaches for different data learning paradigms. Example: Imbalance classification (oversampling, undersampling, cost sensitive)

Snapshot on Data Preprocessing

Summary

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

Snapshot on Data Preprocessing

Summary

Advantage: Data preprocessing allows us to apply Learning/Data Mining algorithms easier and quicker, obaining 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.

Snapshot on Data Preprocessing

Summary

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

Knowledge-Based Systems

Volume 98, 15 April 2016, Pages 1–29

Tutorial on practical tips of the most influential data preprocessing algorithms in data mining

Salvador García [a, b], Julián Luengo [a], Francisco Herrera [a, b]

http://sci2s.ugr.es/most-influential-preprocessing

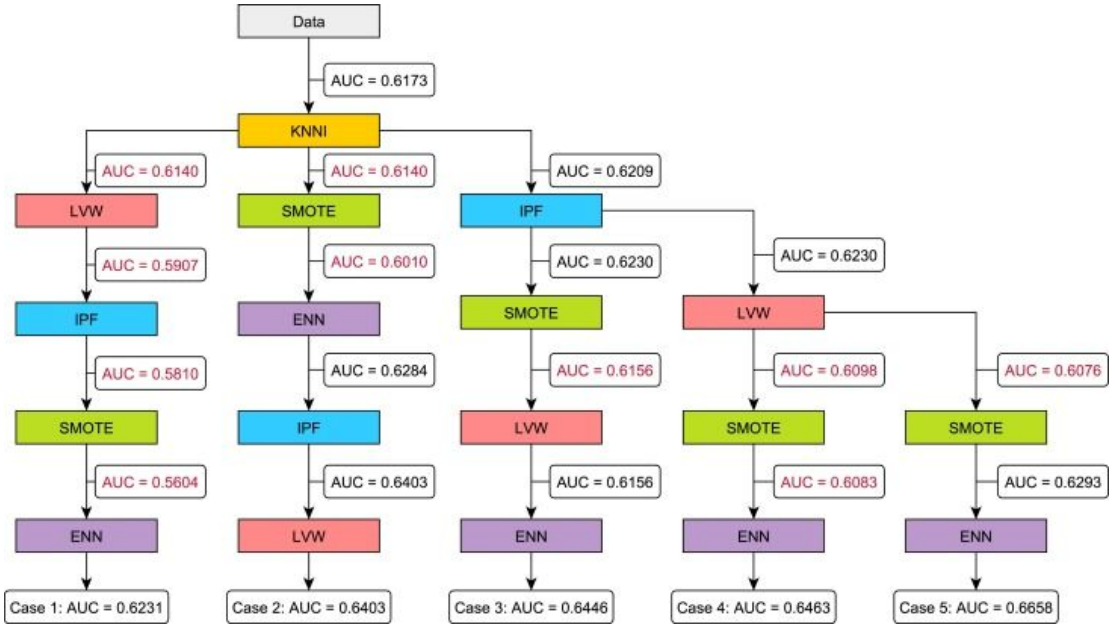
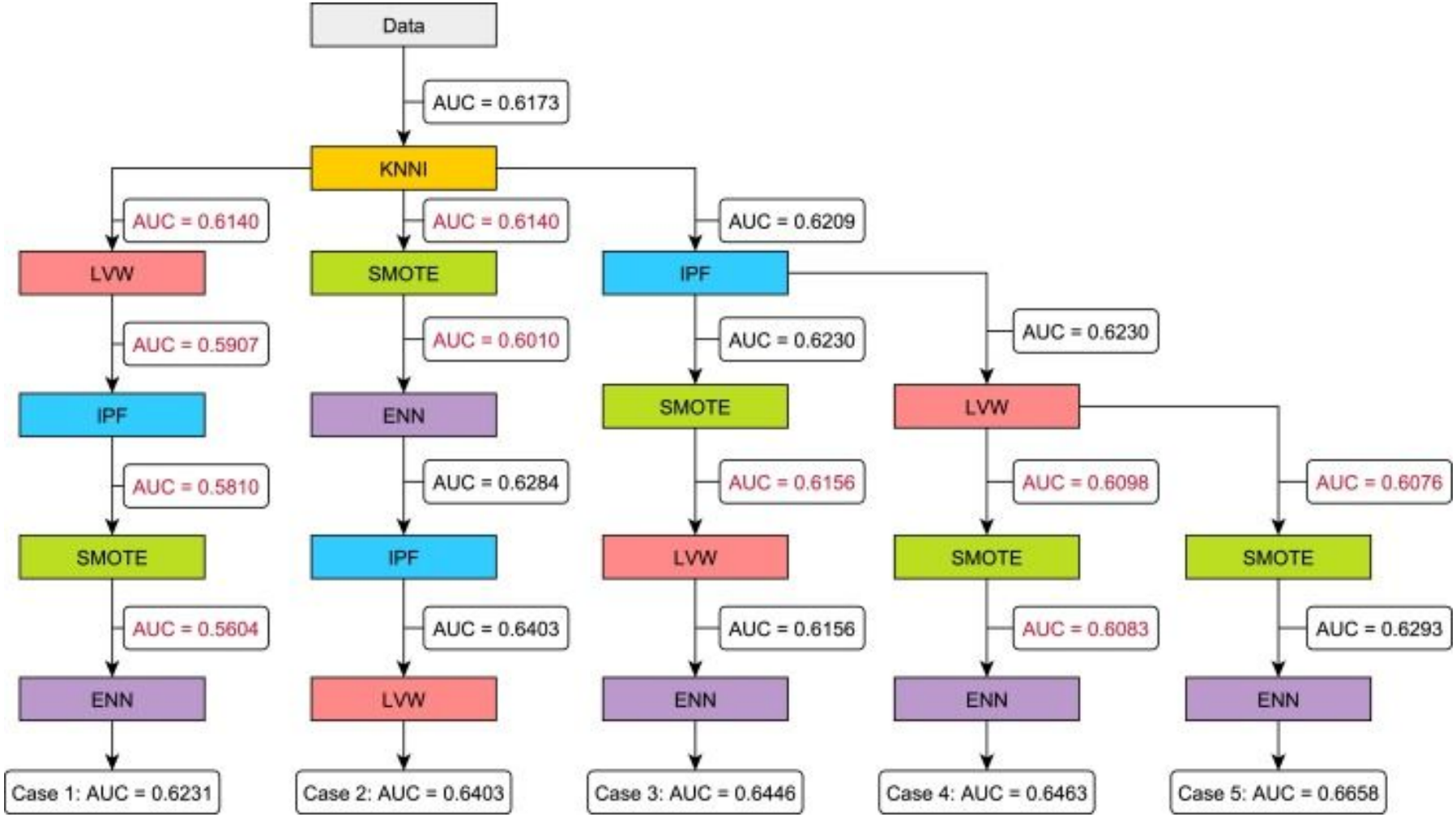


Fig. 29. Depiction of the 5 Cases, hierarchically distributed by the preprocessing application order, showing the AUC values obtained in each step

Snapshot on Data Preprocessing

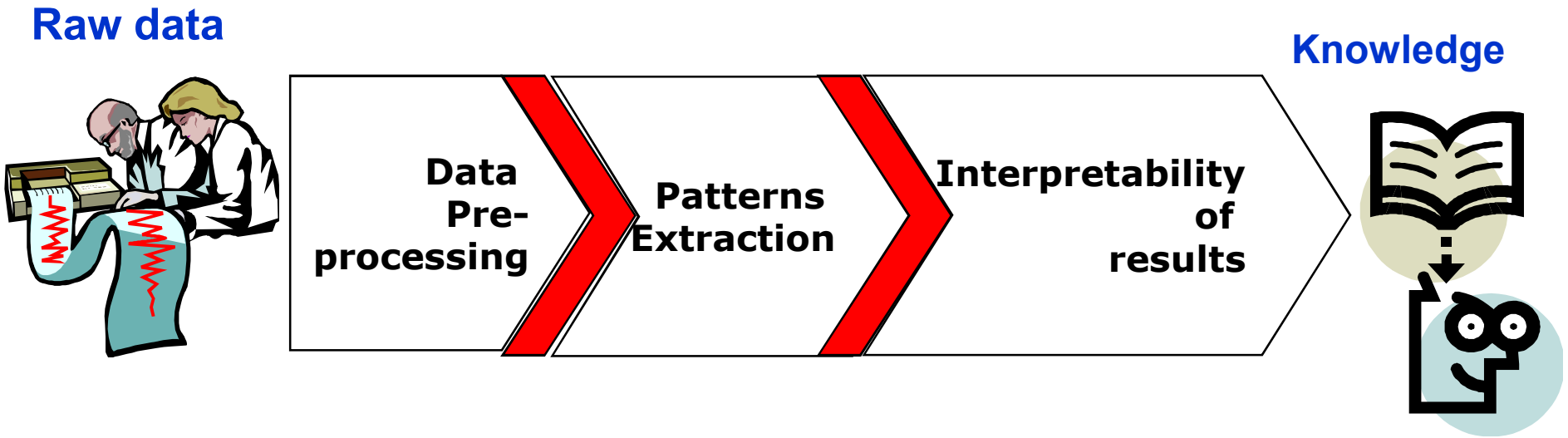


Snapshot on Data Preprocessing

In a nutshell

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

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

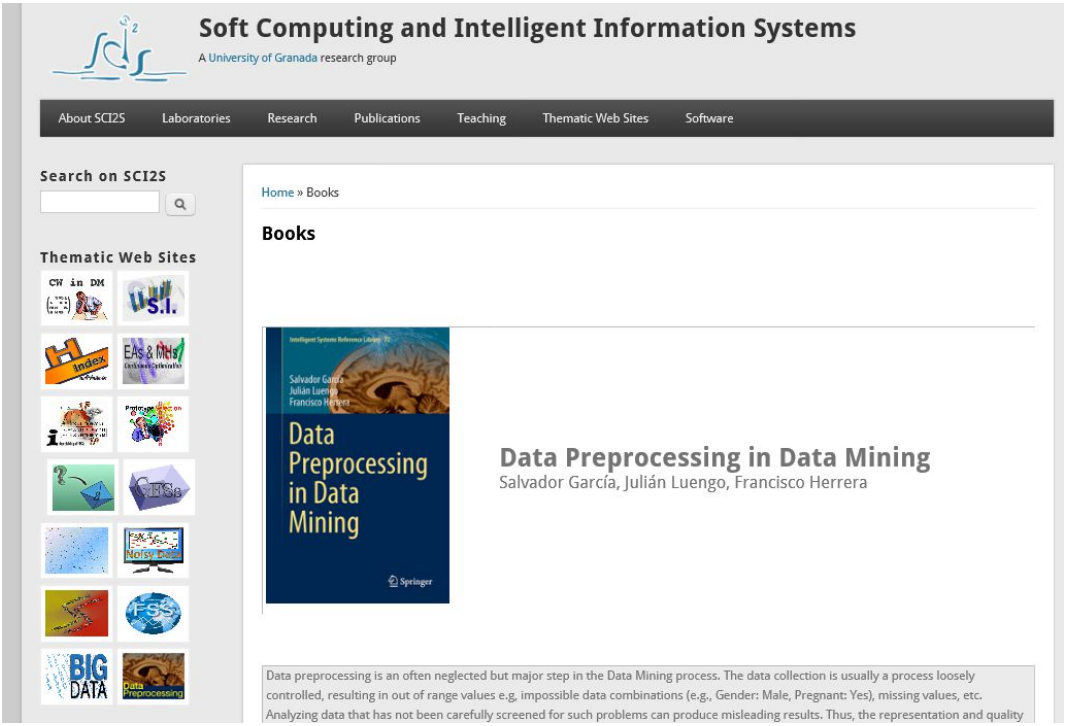


Snapshot on Data Preprocessing

In a nutshell

Website including slides, material, links …
(under preparation)

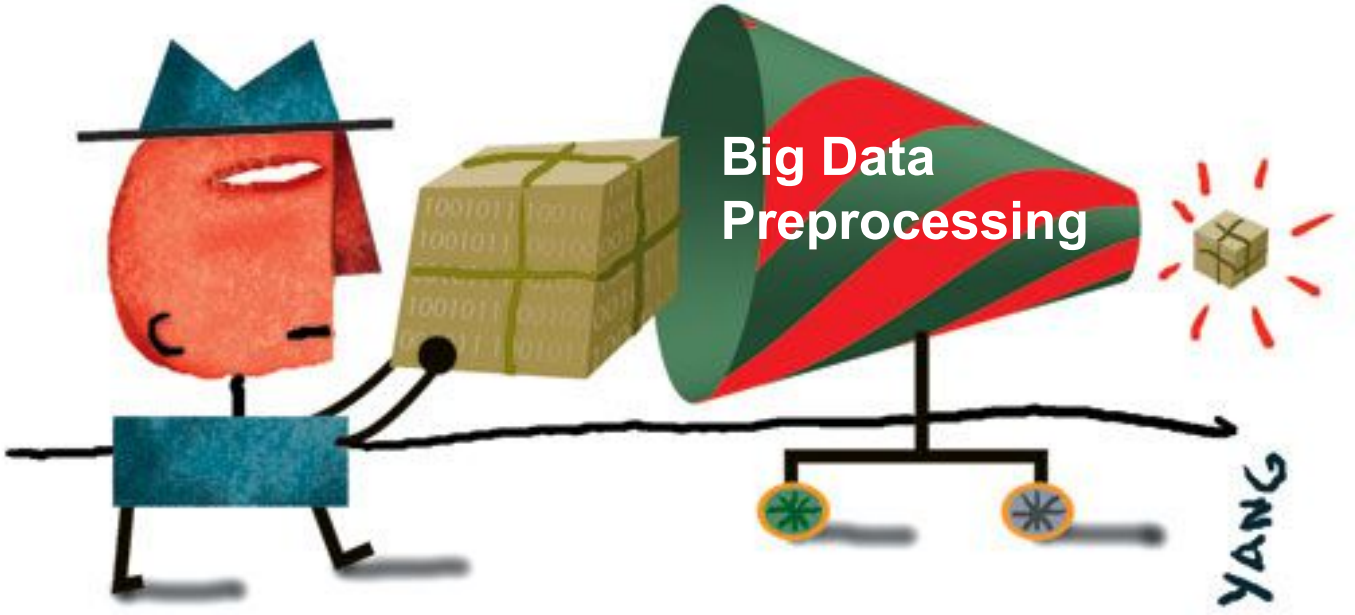
http://sci2s.ugr.es/books/data-preprocessing



From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. **A First Look on Big Data Preprocessing**
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. Discretization
9. Final Comments

Big Data Preprocessing → Quality Data (Smart Data)



Big Data → Smart Data

Quality models based on quality data!

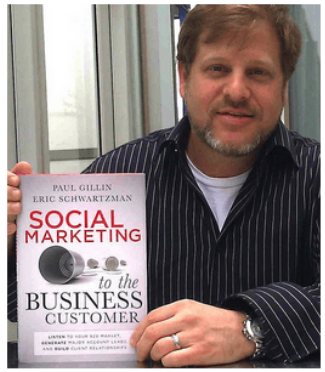
Big data preprocessing also must spends a very important part of the total time in a big data analytic process.

Credit: http://www.ibmsystemsmag.com/aix/storage/servers/The-Data-Squeeze/

Big Data Preprocessing: Two comments

Without Analytics, Big Data is just Noise

Guest post by Eric Schwartzman, founder and CEO of Comply Socially



http://www.briansolis.com/2013/04/without-analytics-big-data-is-just-noise/

Big Data Preprocessing: Two comments

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	

Lack of big data preprocessing algorithms

MLlib

spark.mllib: data types, algorithms, and utilities

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

https://spark.apache.org/mllib/
Version 1.6.0

Big Data Preprocessing

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

Extracting, transforming and selecting features - spark.ml

This section covers algorithms for working with features, roughly divided into these groups:

- Extraction: Extracting features from "raw" data
- Transformation: Scaling, converting, or modifying features
- Selection: Selecting a subset from a larger set of features

Table of Contents

- Feature Extractors
  - TF-IDF (HashingTF and IDF)
  - Word2Vec
  - CountVectorizer
- Feature Transformers
  - Tokenizer
  - StopWordsRemover
  - n-gram
  - Binarizer
  - PCA
  - PolynomialExpansion
  - Discrete Cosine Transform (DCT)
  - StringIndexer
  - IndexToString
  - OneHotEncode
  - VectorIndexer
  - Normalizer
  - StandardScaler
  - MinMaxScaler
  - Bucketizer
  - ElementwiseProduct
  - SQLTransformer
  - VectorAssembler
  - QuantileDiscretizer
- Feature Selectors
  - VectorSlicer
  - RFormula
  - ChiSqSelector

Dimensionality Reduction - spark.mllib

- 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

References

García *et al. Big Data Analytics* (2016) 1:9
DOI 10.1186/s41044-016-0014-0

Big Data Analytics

REVIEW Open Access

CrossMark

Big data preprocessing: methods and prospects

Salvador García*, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez and Francisco Herrera

Complex Intell. Syst.
DOI 10.1007/s40747-017-0037-9

CrossMark

REVIEW ARTICLE

An insight into imbalanced Big Data classification: outcomes and challenges

Alberto Fernández[1] · Sara del Río[1] · Nitesh V. Chawla[2,3] · Francisco Herrera[1]

Big Data Preprocessing

Table. Analyzed methods and the maximum data size managed by each one. The methods are grouped by pre-processing task, and ordered by maximum data size. Those methods with no information about number of features or instances have been set to zero size.

Methods	Category	# Features	# Instances	Size (GB)	Framework
[60]	FS	630	65,003,913	305.1196	Hadoop MapReduce
[61]	FS	1,156	5,670,000	48.8350	MPI
[59]	FS	29,890,095	19,264,097	4.1623	C++/MATLAB
[58]	FS	100,000	10,000,000	1.4901	MapReduce
[62]	FS	100	1,600,000	1.1921	Apache Spark
[63]	FS	54,675	2,096	0.8538	Hadoop MapReduce
[64]	FS	54	581,012	0.2338	Hadoop MapReduce
[65]	FS	20	1,000,000	0.1490	MapReduce
[66]	FS	-	-	0.0976	Hadoop MapReduce
[67]	FS	256	38,232	0.0729	Hadoop MapReduce
[68]	FS	52	5,253	0.0020	Hadoop MapReduce
[69]	FS	-	-	0.0000	Hadoop MapReduce
[70]	FS	-	-	0.0000	Hadoop MapReduce
[71]	FS	-	-	0.0000	Hadoop MapReduce
[72]	Imbalanced	630	32,000,000	150.2037	Hadoop MapReduce
[73]	Imbalanced	41	4,856,151	1.4834	Hadoop MapReduce
[74]	Imbalanced	6	29,887,416	1.3361	Hadoop MapReduce
[75]	Imbalanced	14	1,432,941	0.1495	Hadoop MapReduce
[76]	Imbalanced	18	492,796	0.0661	Hadoop MapReduce
[77]	Imbalanced	36	95,048	0.0255	Hadoop MapReduce
[78]	Incomplete	625	4,096,000	19.0735	MapReduce (Twister)
[79]	Incomplete	481	191,779	0.6873	Hadoop MapReduce
[80]	discretization	630	65,003,913	305.1196	Apache Spark
[81]	discretization	-	-	4.0000	Hadoop MapReduce
[82]	IR	41	4,856,151	1.4834	Hadoop MapReduce

RESEARCH

Big data preprocessing: Methods and Prospects

S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez and F. Herrera

Big Data Preprocessing

Bird's eye view SCI²S website

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:

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

Big Data Preprocessing

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

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Our approaches:

Isaac Triguero

https://github.com/triguero

Sergio Ramírez
sramirez

https://github.com/sramirez

- **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...

- **EUS-BigData**
  Evolutionary Undersampling for Extremely Im...
- **MR-EFS**
  This project includes the implementation of e...
- **MRPR**
  This repository includes the MapReduce impl...
- **ROSEFW-RF**
  This project contains the code used in the RO...

Sara Del Río

https://github.com/saradelrio

Imb-sampling-ROS_and_RUS

Spark implementations of two data sampling methods (random oversampling and random undersampling) for imbalanced classification datasets

Jesús Maillo Hidalgo
JMailloH

https://github.com/JMailloH

kNN_IS

Scala ★ 18 9

Diego García
djgarcia

PhD Researcher in Data Science

https://github.com/djgarcia

RandomNoise

Scala

SmartFiltering

Java

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Discretization

Distributed Minimum Description Length Discretizer for Spark

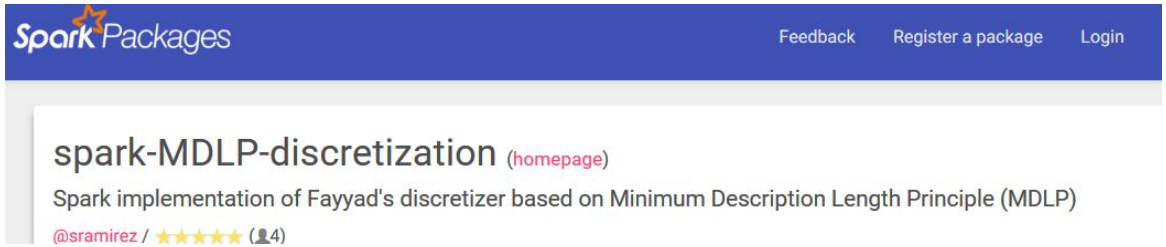
GitHub sramirez / spark-MDLP-discretization

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. Trustcom/BigDataSE/ISPA, 2015 IEEE, Vol, 33-40, DOI 10.1109/Trustcom.2015.559

S Ramírez-Gallego, S García, H Mouriño-Talín, D Martínez-Rego, V. Bolón, A. Alonso-Betanzos, J.M. Benitez, F. Herrera.
Data discretization: taxonomy and big data challenge
Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (1) (2016) 5-21



Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

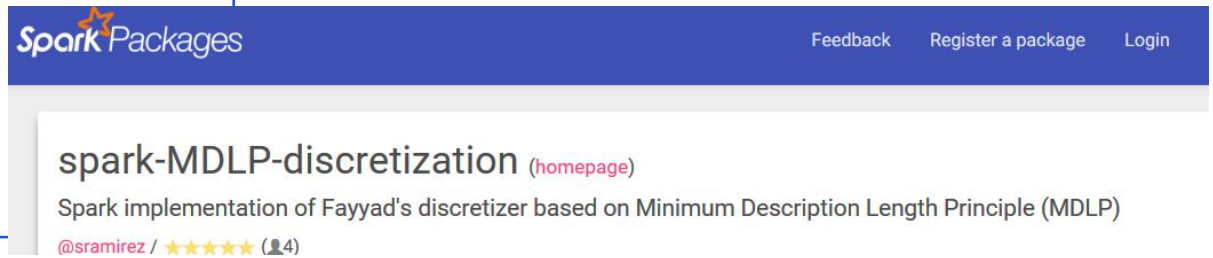
Discretization

Distributed Evolutionary Multivariate Discretizer (DEMD) for Spark

https://spark-packages.org/package/sramirez/spark-DEMD-discretizer

spark-DEMD-discretizer (homepage)
A Distributed Evolutionary Multivariate Discretizer (DEMD) @sramirez /
Here, a Distributed Evolutionary Multivariate Discretizer (DEMD) for data reduction on Spark is presented. This evolutionary-based discretizer uses binary chromosome representation and a wrapper fitness function. The algorithm is aimed at optimizing the cut points selection problem by trading-off two factors: simplicity of solutions and its classification accuracy. In order to alleviate the complexity derived from the evolutionary process, the complete evaluation phase has been fully parallelized. For this purpose, both the set of chromosomes and instances are split into different partitions and a random cross-evaluation process between them is performed.

S Ramírez-Gallego, S García, JM Benítez, F Herrera
A Distributed Evolutionary Multivariate Discretizer for Big Data processing on Apache Spark
Swarm and Evolutionary Computation
38, 240-250, 2018

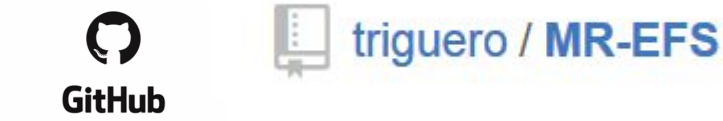


Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Feature Selection

MapReduce based Evolutionary Feature Selection



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.

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

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Feature Weighting

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

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

Evolutionary Feature Weighting
RandomForest
Random Oversampling

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.

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, 87 (2015) 69-79. doi: 10.1016/j.knosys.2015.05.027

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Feature Selection

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

https://github.com/sramirez/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.

Fast-mRMR: Fast Minimum Redundancy Maximum Relevance Algorithm for High-Dimensional Big Data
S Ramírez-Gallego, I Lastra, D Martínez-Rego, V Bolón-Canedo, José Manuel Benítez
Francisco Herrera, Amparo Alonso-Betanzos.
International Journal of Intelligent Systems 32 (2), 134–152

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Feature Selection

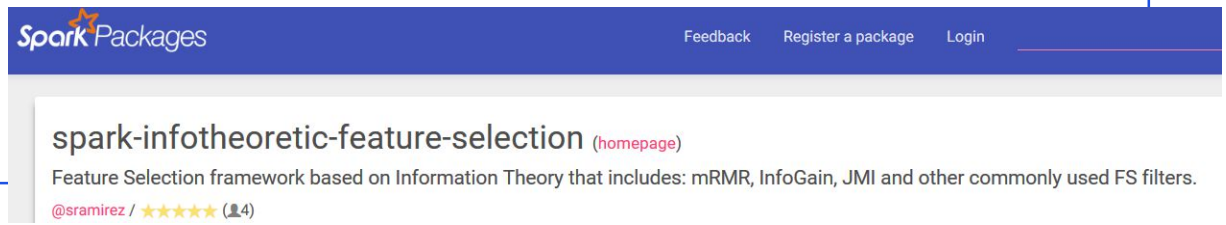
An Information Theoretic Feature Selection Framework for Apache Spark

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

An Information Theory-Based Feature Selection Framework for Big Data under Apache Spark
S Ramırez-Gallego, H Mourino-Talın, D Martınez-Rego, V Bolón-Canedo, José Manuel Benıtez, Amparo Alonso-Betanzos, Francisco Herrera
IEEE Transactions on Systems, Man, and Cybernetics: Systems, In press



Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Feature Selection

Distributed version of RELIEF-F algorithm for Apache Spark.

https://spark-packages.org/package/sramirez/spark-RELIEFFC-fselection

spark-RELIEFFC-fselection (homepage) @sramirez /

The present algorithm (called BELIEF) implements Feature Weighting (FW) on Spark for its application on Big Data problems. This repository contains an improved implementation of RELIEF-F algorithm [1], which has been extended with a cheap but effective feature redundancy elimination technique. BELIEF leverages distance computations computed in prior steps to estimate inter-feature redundancy relationships at virtually no cost. BELIEF is also highly scalable to different sample sizes, from hundreds of samples to thousands.

Big Data Preprocessing

Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Prototype Reduction

MapReduce based Prototype Reduction

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

Big Data Preprocessing

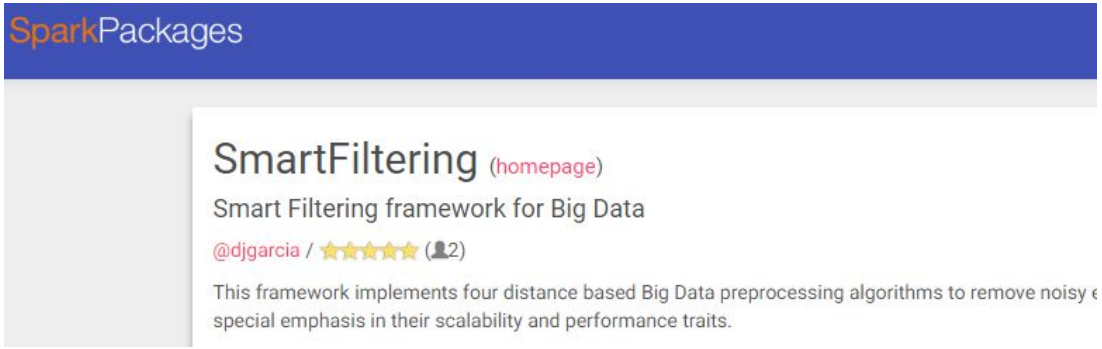
Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Instance Filtering

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

SmartFiltering (homepage)
Smart Filtering framework for Big Data @djgarcia /

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



Big Data Preprocessing

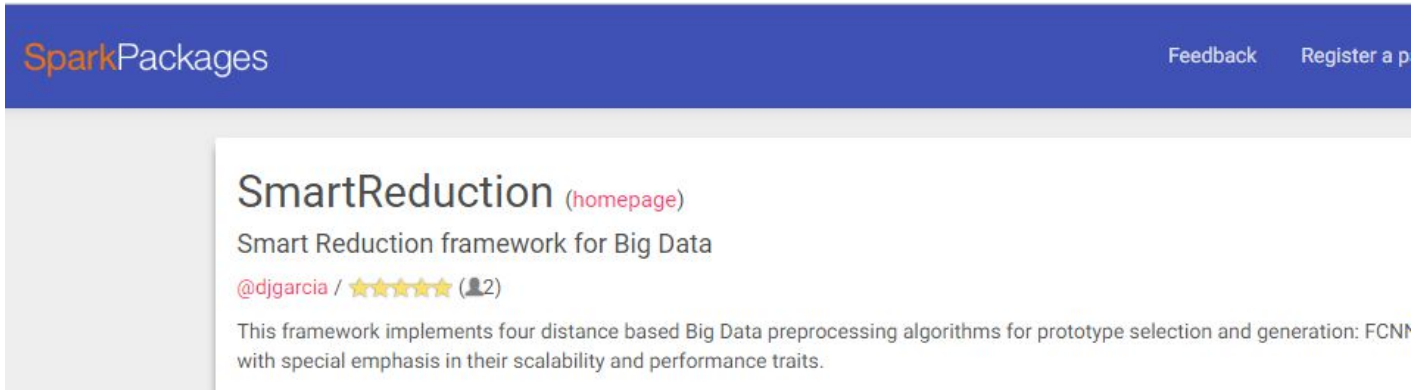
Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Instance Reduction

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

SmartReduction (homepage)
Smart Reduction framework for Big Data @djgarcia /

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



Big Data Preprocessing

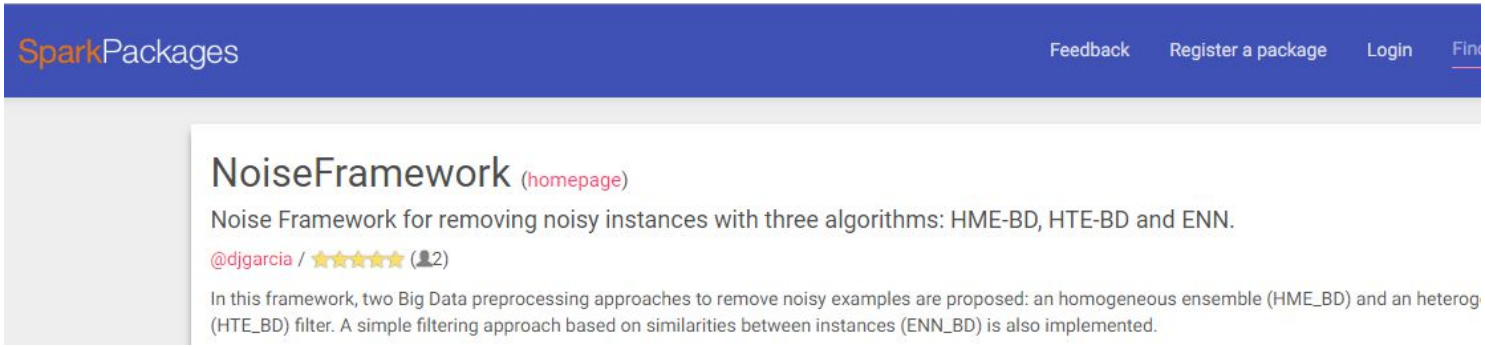
Bird's eye view SCI²S website http://sci2s.ugr.es/BigData

Noise Filtering

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

NoiseFramework (homepage)
Noise Framework for removing noisy instances with three algorithms: HME-BD, HTE-BD and ENN. @djgarcia /

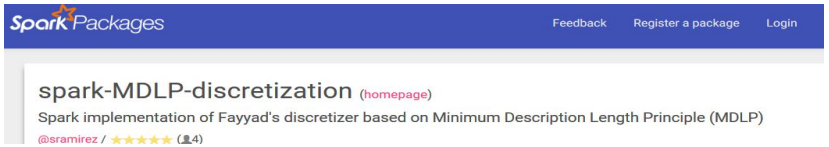
In this framework, two Big Data preprocessing approaches to remove noisy examples are proposed: an homogeneous ensemble (HME_BD) and an heterogeneous ensemble (HTE_BD) filter. A simple filtering approach based on similarities between instances (ENN_BD) is also implemented.



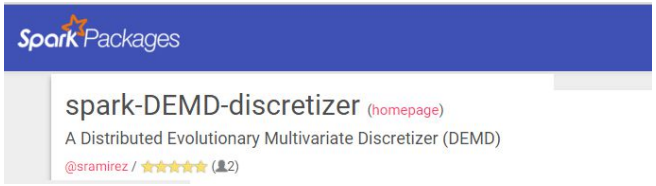
Big Data Preprocessing

Bird's eye view SCI²S website

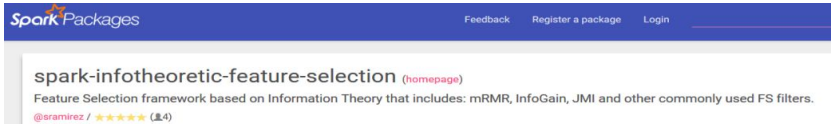
Discretization, feature selection, kNN, Imbalanced Data, ….



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

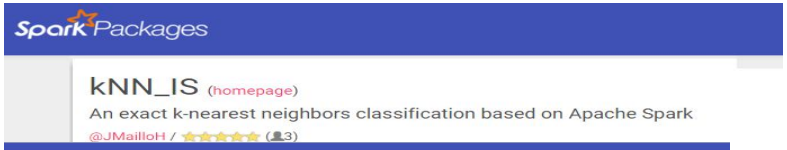


http://spark-packages.org/package/sramirez/spark-DEMD-discretizer



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

https://spark-packages.org/package/sramirez/spark-RELIEFFC-fselection



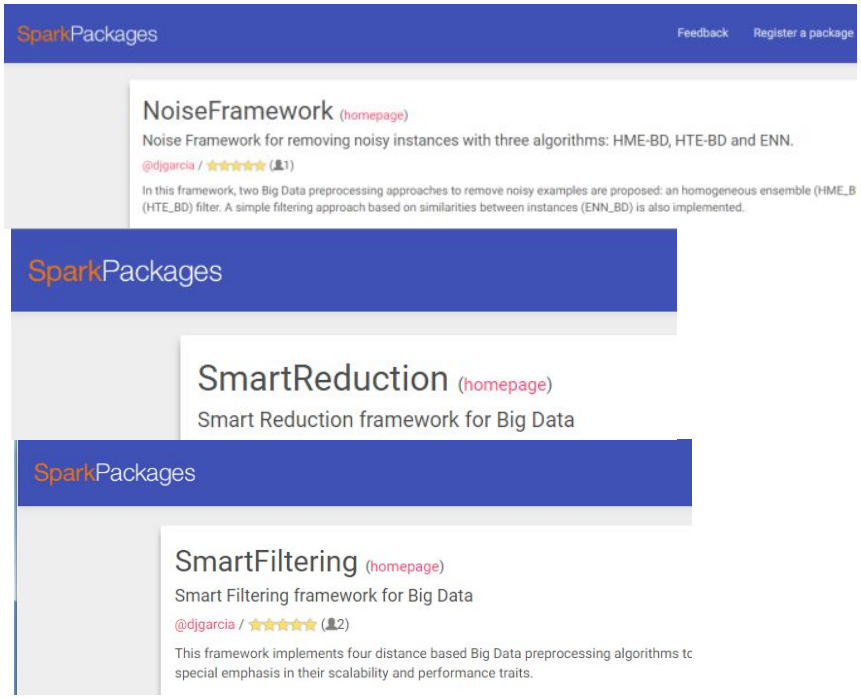
http://spark-packages.org/package/JMailloH/kNN_IS

https://spark-packages.org/package/saradelrio/Imb-sampling-ROS_and_RUS

Big Data Preprocessing

Bird's eye view SCI²S website

Noise Filtering, Instance Reduction



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

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

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

Big Data Preprocessing

Describing some Approaches

- Noise Framework
- MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- Evolutionary Undersampling for Imbalanced Big Data
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-FS: 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

From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. **Imperfect Data**
6. Instance Reduction
7. Feature Selection
8. Discretization
9. Final Comments

Big Data Preprocessing

Describing some Approaches

- **Noise Framework**
- **MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **Evolutionary Undersampling for Imbalanced Big Data**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-FS: 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**

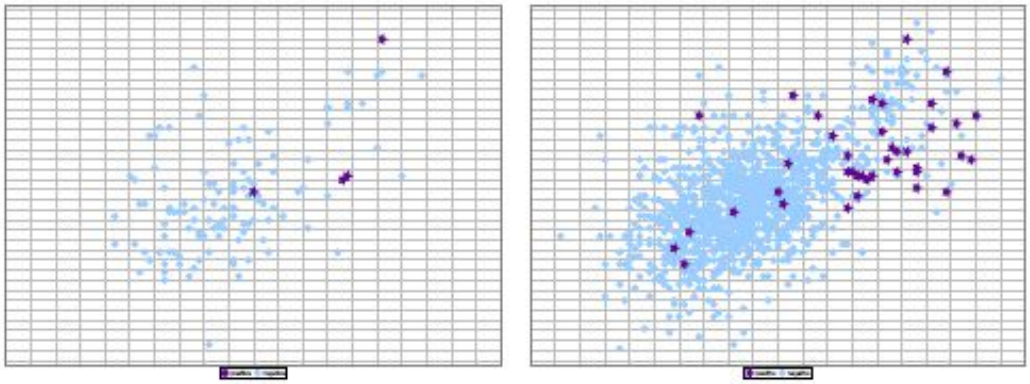
Noise Data and Big Data

Noise values are present in big data.

To design noise filters it is necessary to design methods to identify the incorrect labels and to take decisions on the noise and forders.

The lack of data is also a problem to analyse.

Lack of Data. It may introduce small disjuncts



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

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

Noise Data and Big Data

EnsembleFilter-BigData

Building the noise filter

- Create *N* folds from the *training* set
- Each of the *N* folds is used to train a classifier:
  - We have considered to use RandomForest-BigData –RF-BigData-
- In order to build a RF-BigData model, we distribute the instances of each fold in *m* mappers:
  - Each mapper will contain a disjoint subset of instances belonging to the same folds
  - Each mapper will generate a RandomForest made of *k* trees
- All the RandomForest from the mappers belonging to the same fold are joined, creating the final RF-BigData model → we obtain *N* RF-BigData models

Noise Data and Big Data

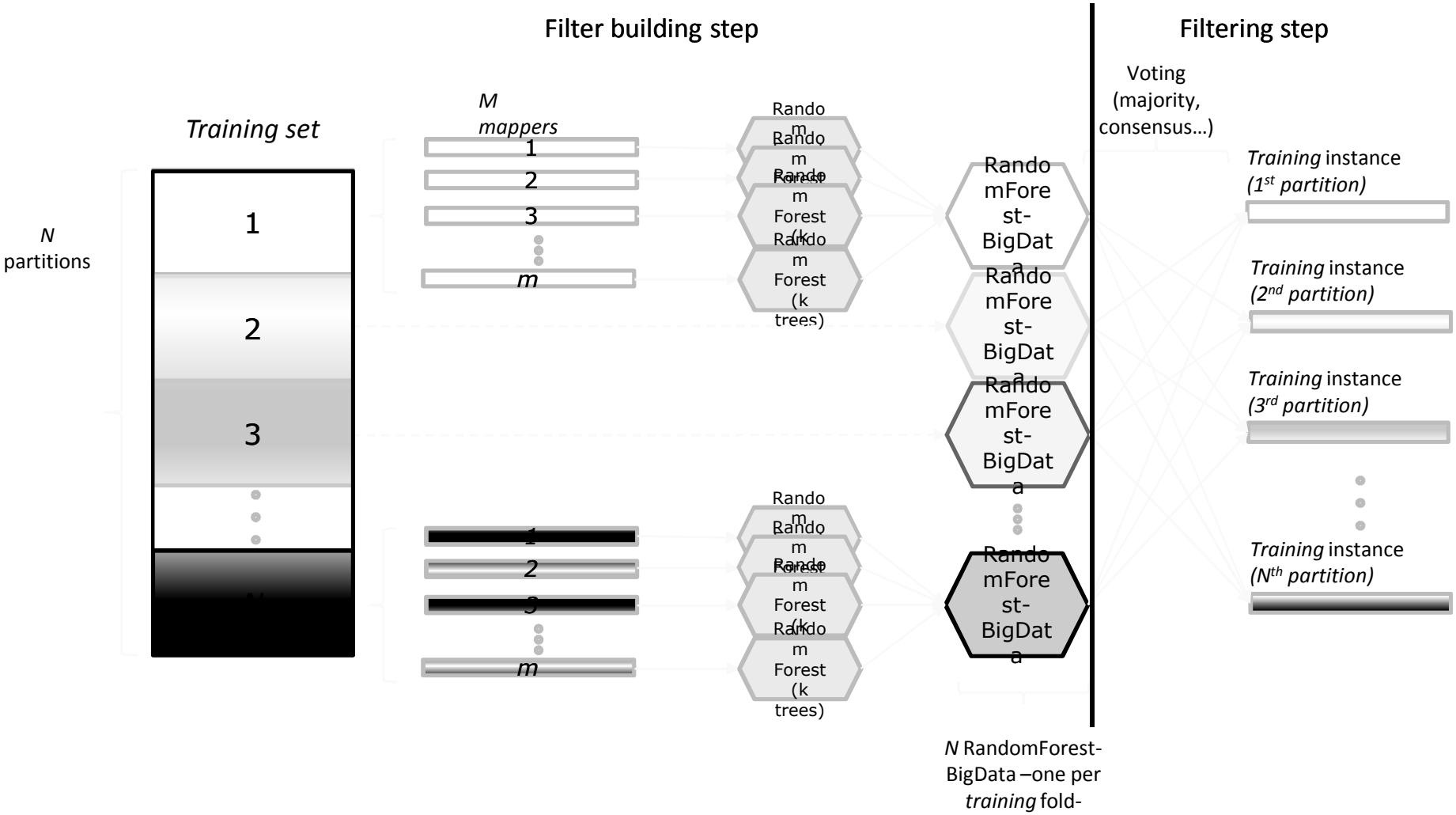
EnsembleFilter-BigData

Filtering the instances

- Once we have the *N* RF-BigData models, each *training* instance is then evaluated:

1. It is classified by the *N-1* RF-BigData that belong to other folds different than the instance's.
2. Depending of the voting scheme chosen, if the instance is voted as noisy by the *N-1* predictions then it is removed from the *filtered training* set

Noise Data and Big Data



Noise Data and Big Data

Filtered instances percentages

- Let us consider N=5, m=4 and k=5, with consensus voting*
- In the following table we show the percentages of removed instances in the training partition made by a preliminary version of the EnsembleFilter-BigData filter
- Four scenarios of noise are taken into account: 0%, 5%, 10% and 20% of artificially added class noise

POKER		0% noise	5% noise	10% noise	20%noise
	#TotalSamples	757439	757439	757439	757439
	#FilteredSamples	757439	757439	757439	757439
	%Filtered	0.0000	0.0000	0.0000	0.0000
SUSY					
	#TotalSamples	4000000	4000000	4000000	4000000
	#FilteredSamples	3998691	3999375	3999979	4000000
	%Filtered	0.0327	0.0156	0.0005	0.0000
HIGGS					
	#TotalSamples	8800000	8800000	8800000	8800000
	#FilteredSamples	8799927	8799921	8799922	8799924
	%Filtered	0.0008	0.0009	0.0009	0.0009
KDDCUP_NORMAL_VS_DOS					
	#TotalSamples	3884921	3884921	3884921	3884921
	#FilteredSamples	3884915	3714183	3558603	3273466
	%Filtered	0.0002	4.3949	8.3996	15.7392

* If we use majority voting, too many instances are filtered, resulting in subsampled training sets → **overfitting**

- We have considered 4 data sets:
  - Poker
  - Susy
  - Higgs
  - KDDcup (class *normal* vs. *dos*)
- The original size of the data set is shown, as well as the size **after** the filtering
- The percentage of removed instances is also shown

Noise Data and Big Data

Performance results for *kNN*

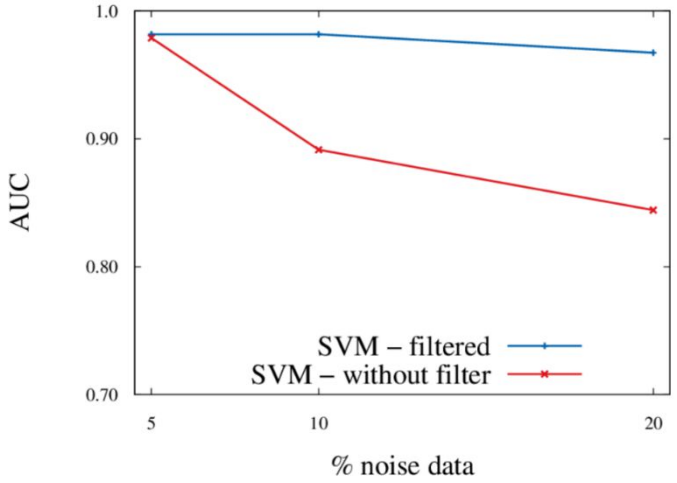
- We focus on the KDDcup problem, as it is the only one being filtered by the EnsembleFilter-BigData
- In the case of SVM, the filtering helps to improve the performance if compared to the original data set (for %5 onwards noise scenarios)
- We show the results of test Accuracy and AUC measures
  - In the 0% case, only 6 instances were removed, thus no differences can be remarked

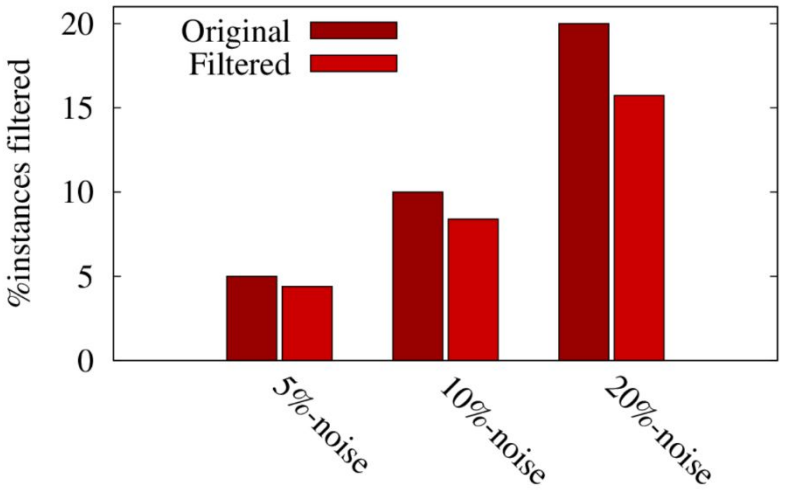
%NOISE	Time		Accuracy		AUC	
	No Filter	Filtered	No Filter	Filtered	No Filter	Filtered
5	221.2760	231.9147	0.9813	0.9858	0.9788	0.9817
10	219.7052	221.2946	0.9081	0.9827	0.8914	0.9817
20	222.6837	219.0413	0.7970	0.9670	0.8442	0.9673

Noise Data and Big Data

Performance results for *SVM*

- SVM is able to benefit from the removed noise
- The difference in performance increases with the amount of noise introduced in the data set





Noise Data and Big Data

Performance results for *kNN*

- If we consider *kNN* with the KDDcup data set, the results are not that appealing
- Since the filter tends to remove the minority class examples, *kNN* performance is diminished
- Please also note that *kNN* also performs poorly at 0% level in AUC, showing that KDDcup is a difficult imbalanced problem

%NOISE	Time		Accuracy		AUC	
	No Filter	Filtered	No Filter	Filtered	No Filter	Filtered
5	4032.6836	4790.1359	0.7625	0.6870	0.5003	0.4995
10	4553.6011	4616.8112	0.5749	0.4625	0.5000	0.5000
20	4458.4321	4254.0720	0.5376	0.5376	0.5002	0.5002

From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. **Instance Reduction**
7. Feature Selection
8. Discretization
9. Final Comments

Big Data Preprocessing

Describing some Approaches

- **Noise Framework**
- **MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **Evolutionary Undersampling for Imbalanced Big Data**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-FS: 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**

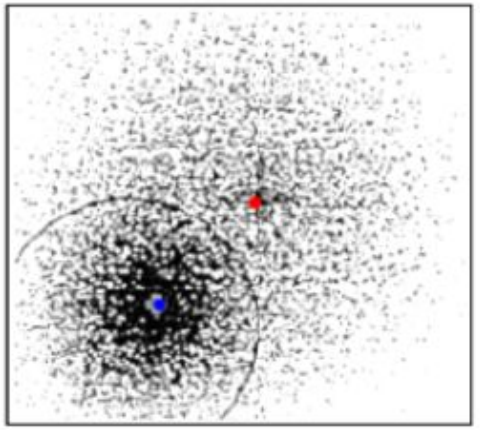
Big Data Preprocessing:MRPR

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

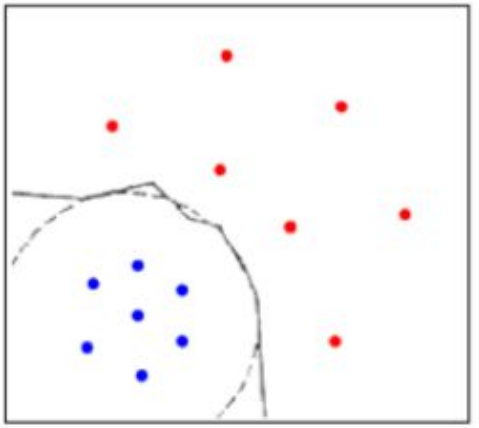
I. Triguero, D. Peralta, J. Bacardit, S.García, F. Herrera. A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation.
Evolutionary Computation (CEC), 2014 IEEE Congress on, 3036-3043

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.

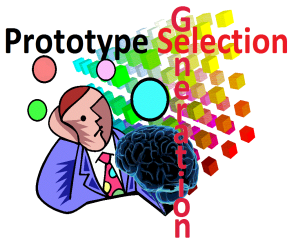


Big Data Preprocessing: MRPR

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 uses Windowing: Incremental Learning with Alternating Strata (ILAS)
- As output, it returns a Generated Set of prototypes.

Reduce phase:

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

Big Data Preprocessing: MRPR

Parallelizing PG with MapReduce

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

Two alternative reducers:

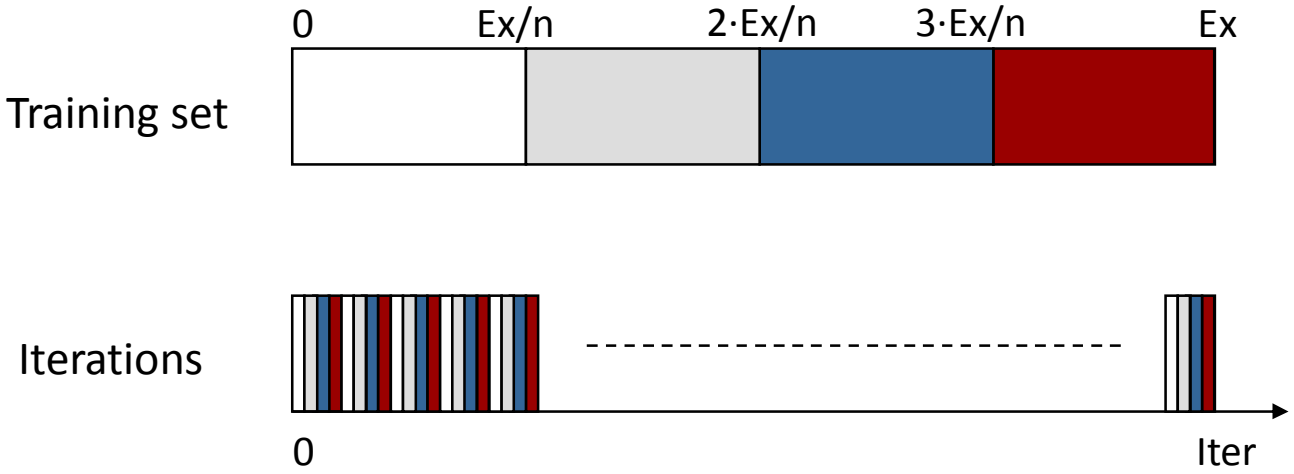
- Join: Concatenates all the resulting generated sets.
  - This process does not guarantee that the final generated set does not contain irrelevant or even harmful instances
- Fusion: This variant eliminates redundant prototypes by fusion of prototypes. Centroid-based PG methods: ICPL2 (Lam et al).

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

Big Data Preprocessing: MRPR

Windowing: Incremental Learning with Alternating Strata (ILAS)

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



Main properties:

✓Avoids a (potentially biased) static prototype selection

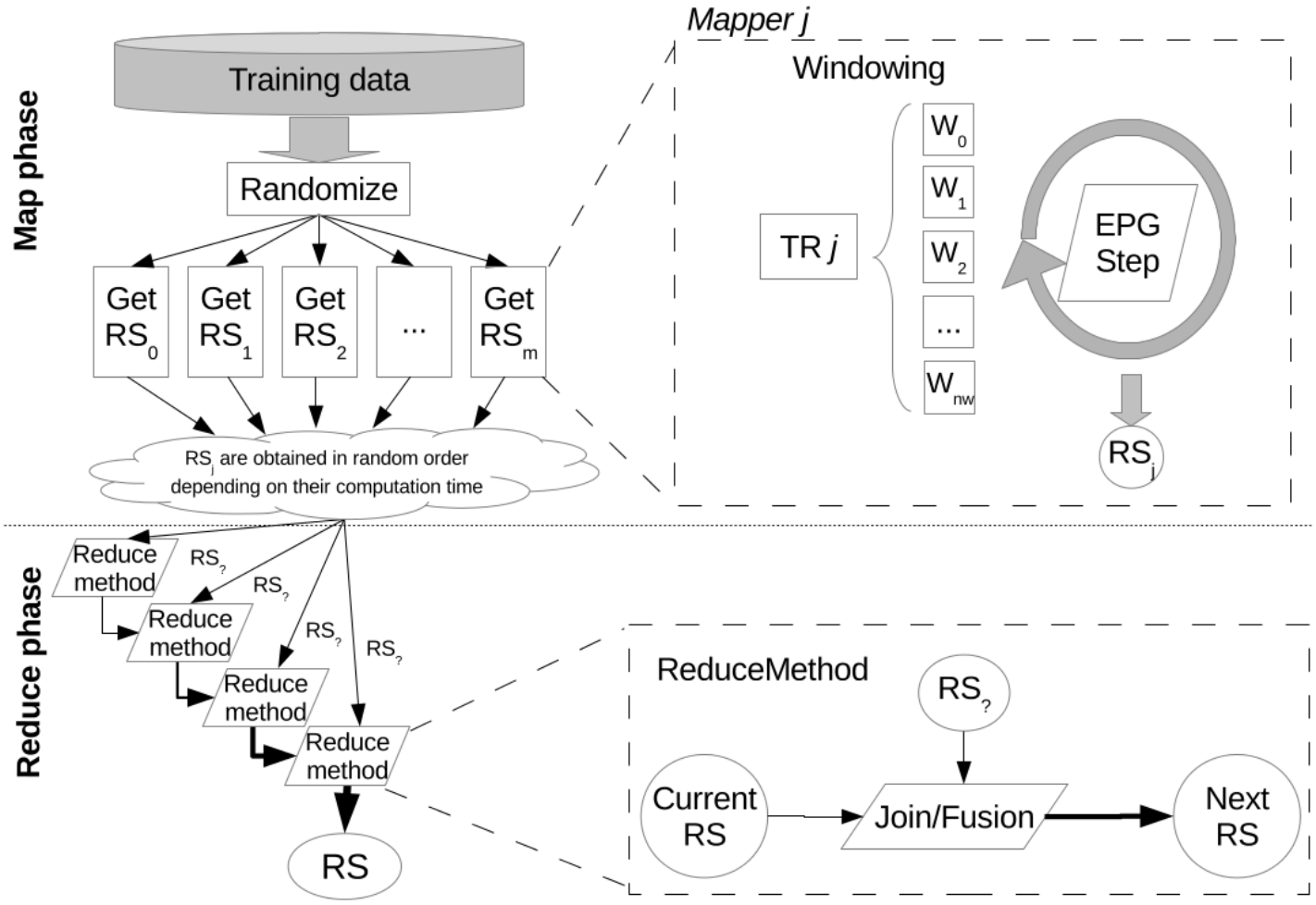
✓This mechanism also introduces some generalization pressure

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

Big Data Preprocessing: MRPR

The MRW-EPG scheme

Windowing: Incremental Learning with Alternating Strata (ILAS)



Big Data Preprocessing: MRPR

Experimental Study

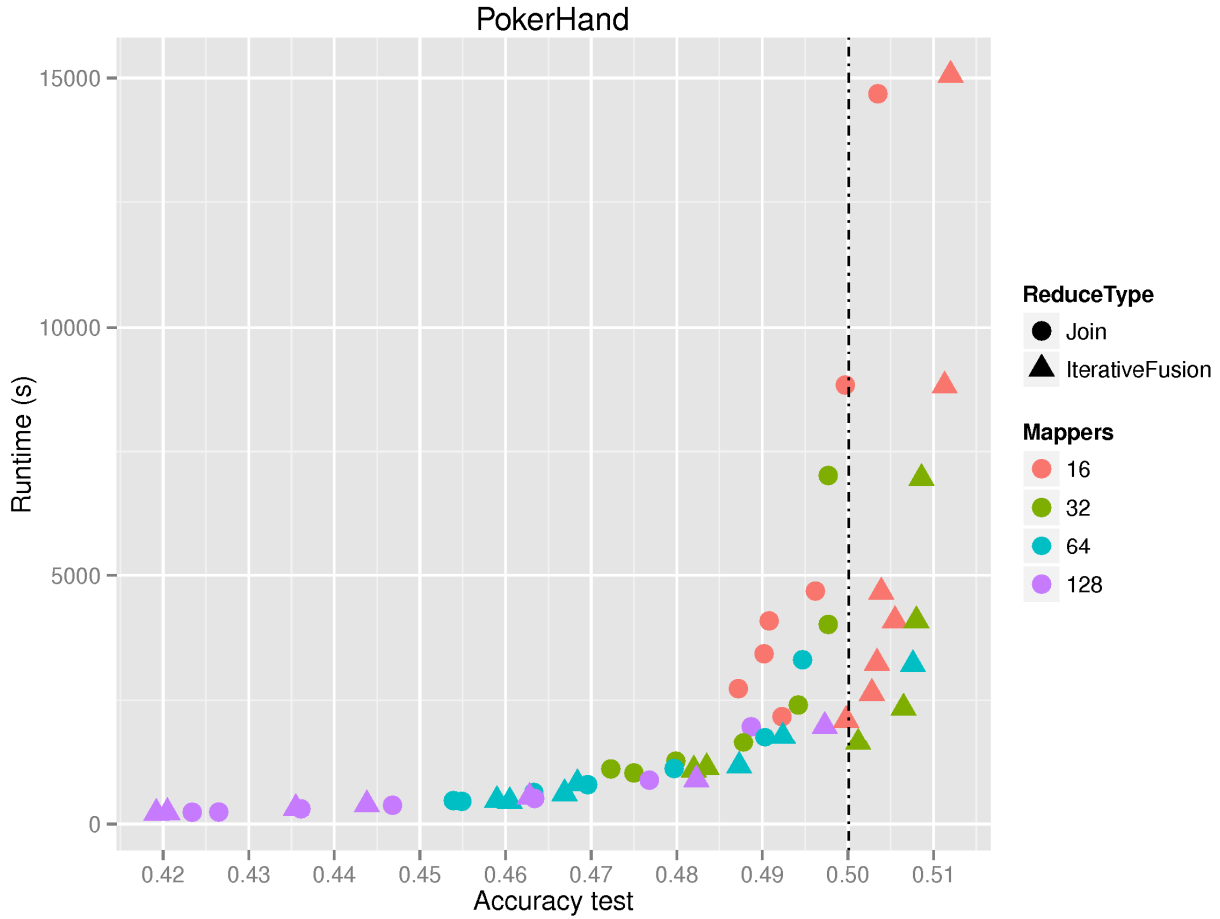
- PokerHand data set. 1 million of instances, 3x5 fcv.
- Performance measures: Accuracy, reduction rate, runtime, test classification time and speed up.
- PG technique tested: IPADECS.

TABLE I: Parameter specification for all the methods

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

Big Data Preprocessing: MRPR

Results



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

Big Data Preprocessing: MRPR

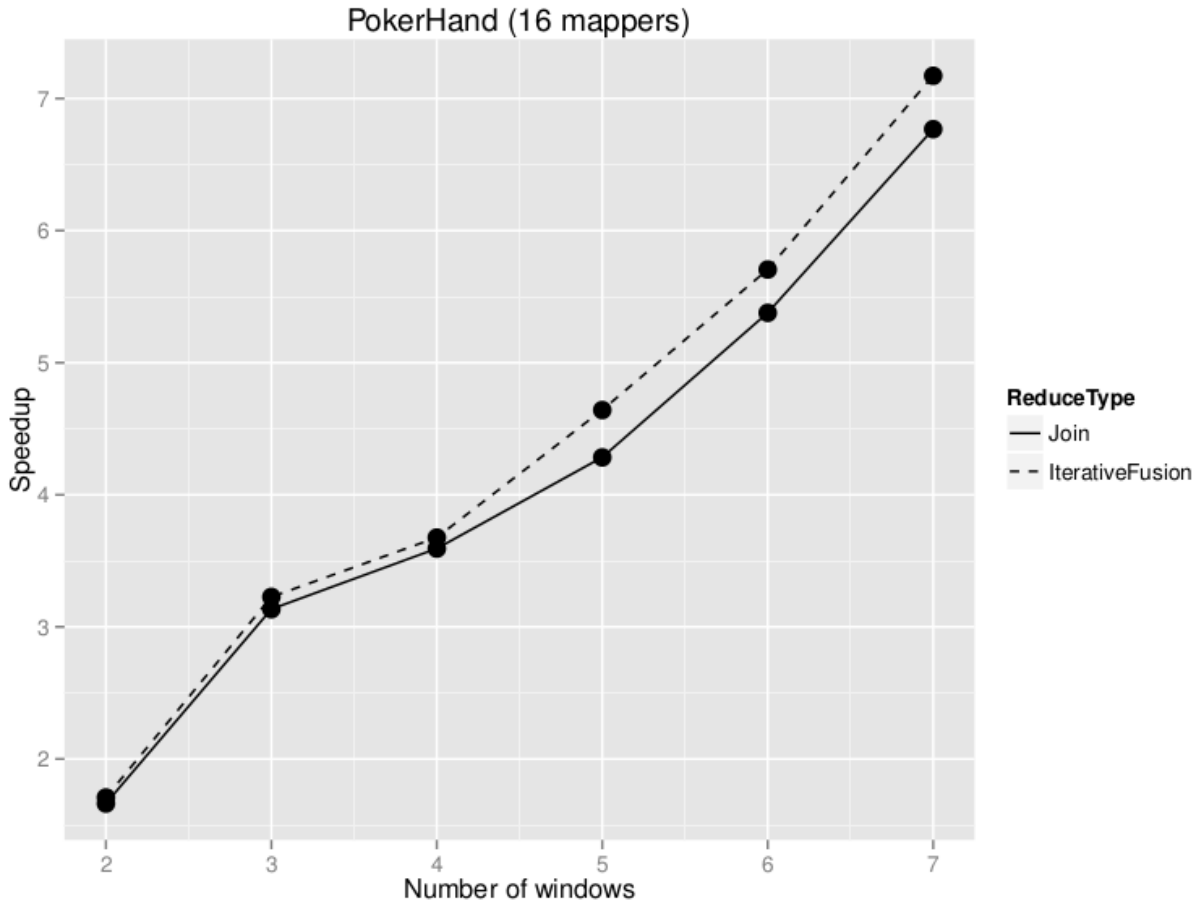
Results

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

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

Big Data Preprocessing: MRPR

Results: Speed-up



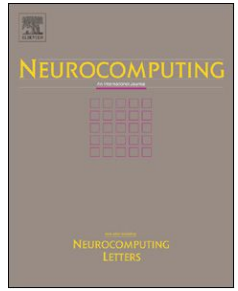
Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments

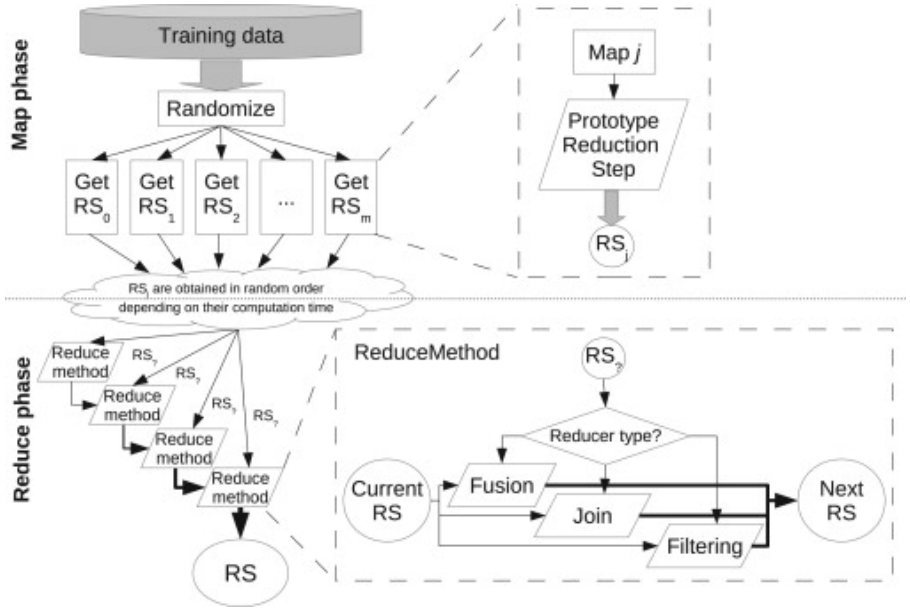
- There is a good synergy between the windowing and MapReduce approaches. They complement themselves in the proposed two-level scheme.
- Without windowing, evolutionary prototype generation could not be applied to data sets larger than approximately ten thousands instances
- The application of this model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

Big Data Preprocessing: MRPR

EPG for Big Data: Final Comments



- **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

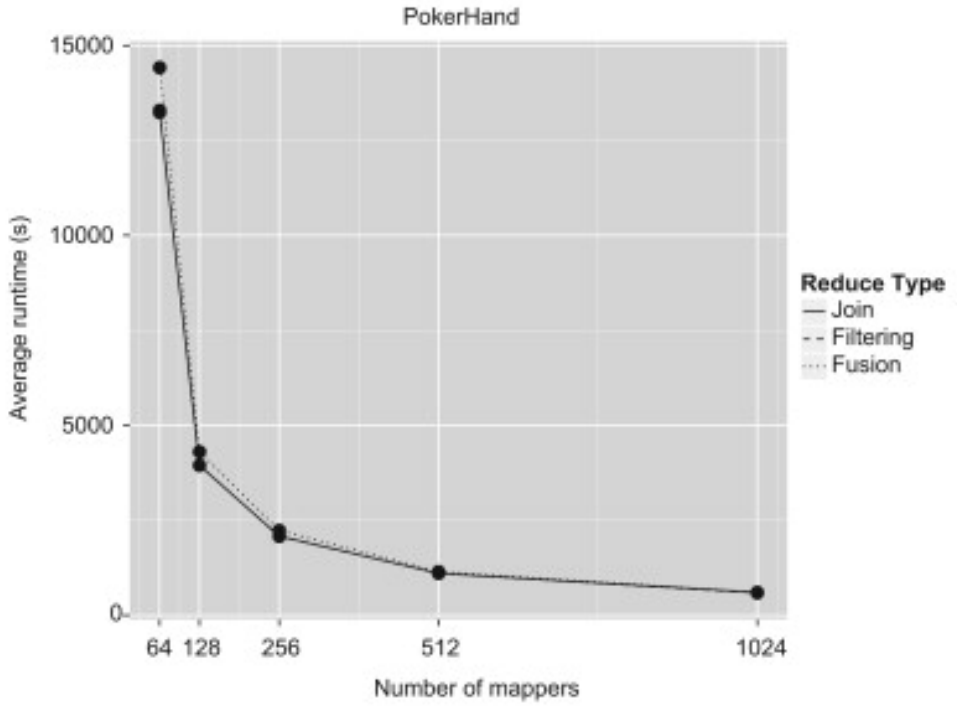


Fig. 6 Average runtime obtained by MRPR. (a) PokerHand

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

- **Noise Framework**
- **MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation**
- **Evolutionary Undersampling for Imbalanced Big Data**
- **MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach**
- **Spark-FS: 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**

Imbalanced Big Data

Evolutionary Undersampling for Imbalanced Big Data

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

Evolutionary Undersampling

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

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

Evolutionary Undersampling in the big data context

- The application of **EUS in big data** problems is interesting because it **does not generate more data**, as opposed to oversampling methods.

- However, the increasing number of instances would lead to obtain an **excessive chromosome size** that can limit their application.

- The required **runtime** increases not only with the **number of examples** but also with the **imbalance ratio** (IR).

Evolutionary undersampling

- **Representation of the solution**

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

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

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

- **Fitness Function**

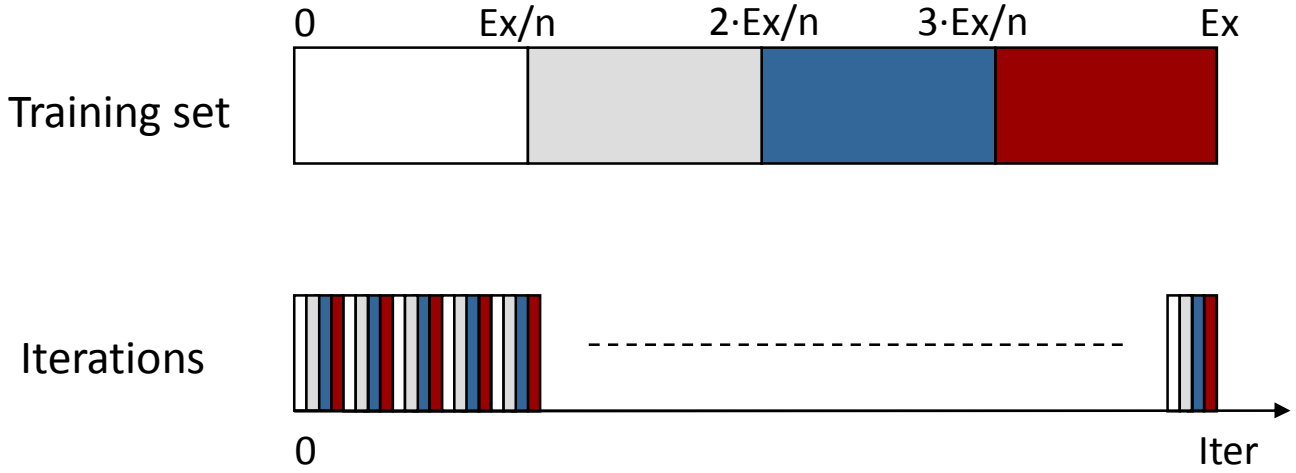
$$\text{fitness}_{\text{EUS}} = \begin{cases} \text{g-mean} - \left|1 - \frac{n^+}{N^-} \cdot P\right| & \text{if } N^- > 0 \\ \text{g-mean} - P & \text{if } N^- = 0, \end{cases}$$

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

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

Windowing: Incremental Learning with Alternating Strata (ILAS)

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



Main properties:

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

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

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

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

    Goal: Memory limitations and runtime.

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

    Goal: Runtime.

Parallelizing EUS with MapReduce

Map phase:

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

Reduce phase:

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

Parallelizing EUS with windowing: Motivation

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

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

Windowing for Class Imbalance

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

Parallelizing EUS-BD with windowing: Properties

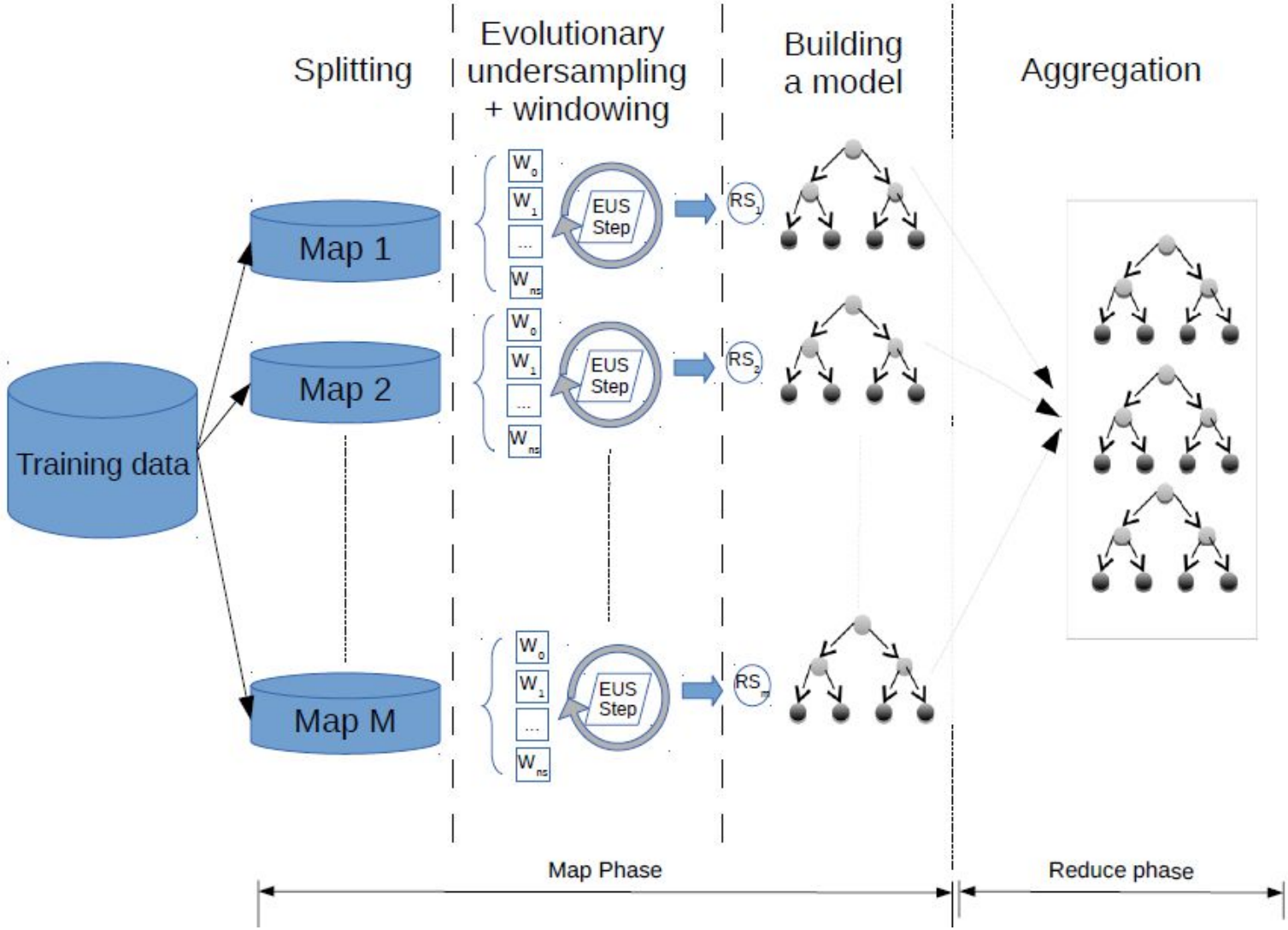
Properties:

- Within this scheme, the algorithm disposes of the whole information although it is accessed in successive iterations.

- This model itself aims to improve the **runtime requirements** of EUS. But it does **not** deal with **the memory consumption** problem.

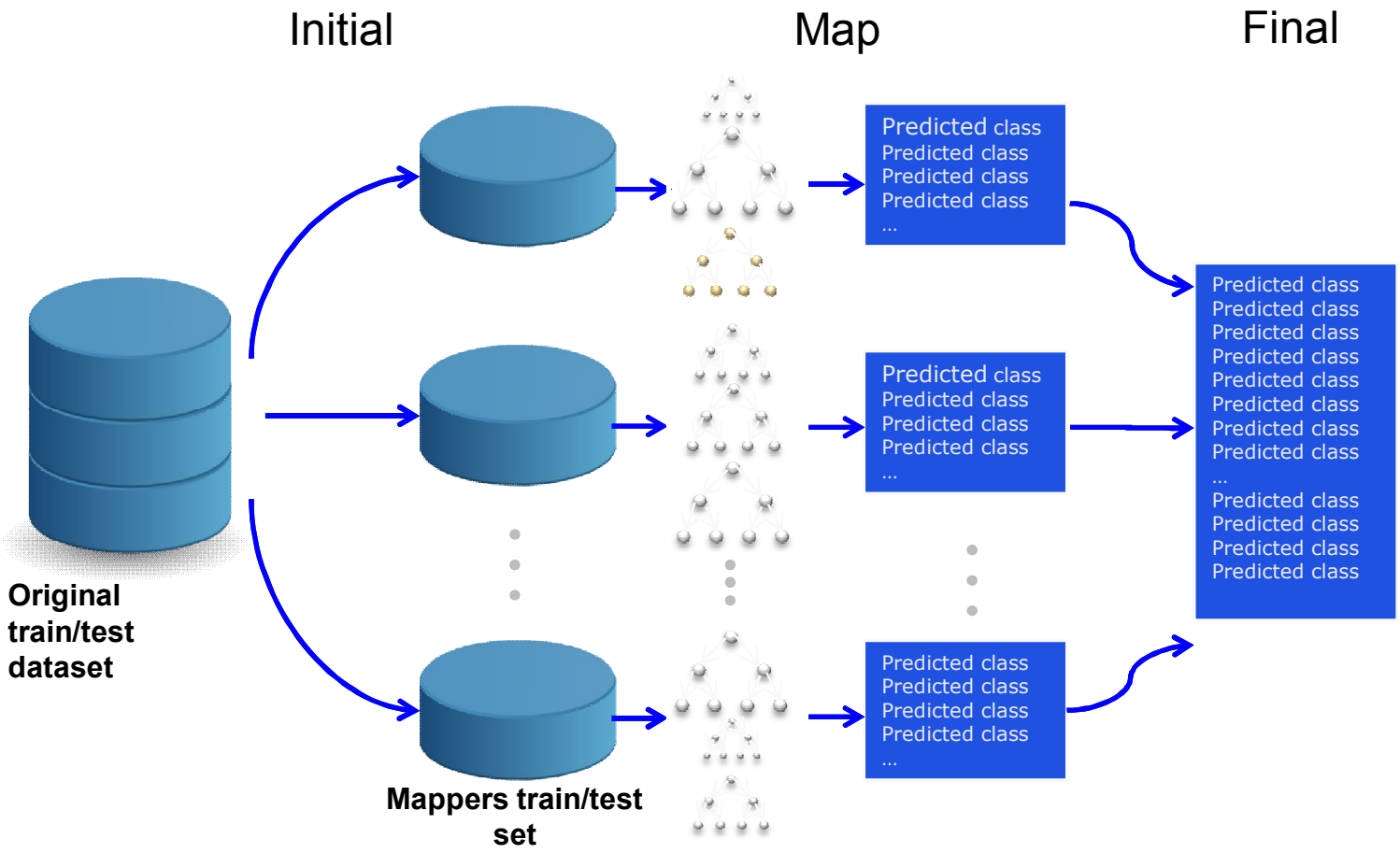
- This is why we use this strategy as a second level parallelization scheme after a previous distribution of the processing in a cluster of computing elements.

The EUS-BD scheme: learning phase



The EUS-BD scheme: testing phase

Estimating the class for a Big Dataset



Experimental Framework

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

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

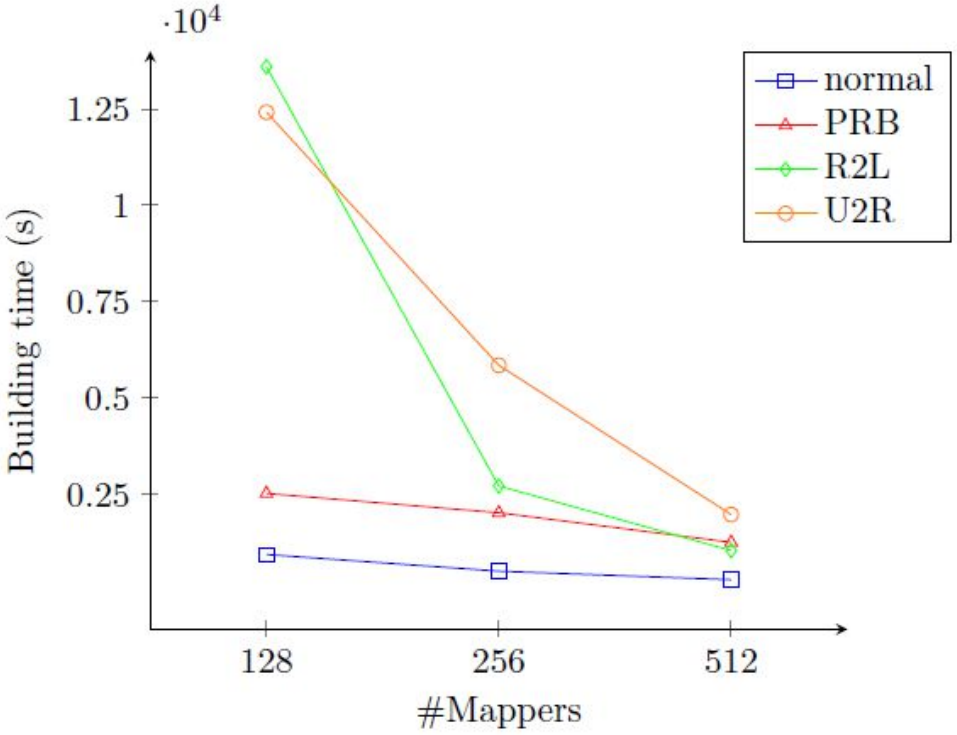
Results obtained without using the windowing mechanism

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

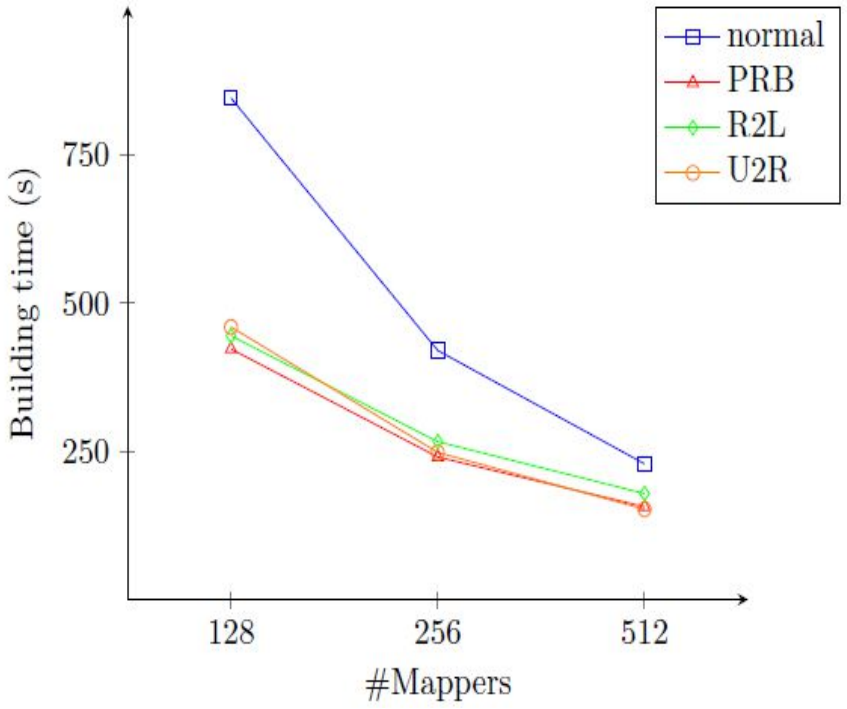
Results obtained using the windowing mechanism

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

Results: Building time comparison



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



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

Final comments

- **Good synergy** between the **windowing** and **MapReduce** approaches.
- It enables **EUS** to be applied on **data sets** of almost **arbitrary size**.
- Replacing EUS by other preprocessing mechanisms such as **hybrid oversampling/undersampling approaches**.
- Beyond MapReduce: **Spark** operators.

From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. **Feature Selection**
8. Discretization
9. Final Comments

Big Data Preprocessing

Describing some Approaches

- Noise Framework
- MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- Evolutionary Undersampling for Imbalanced Big Data
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-FS: 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, Vol. 2015, Article ID 246139, 11 pages, 2015,
doi: 10.1155/2015/246139

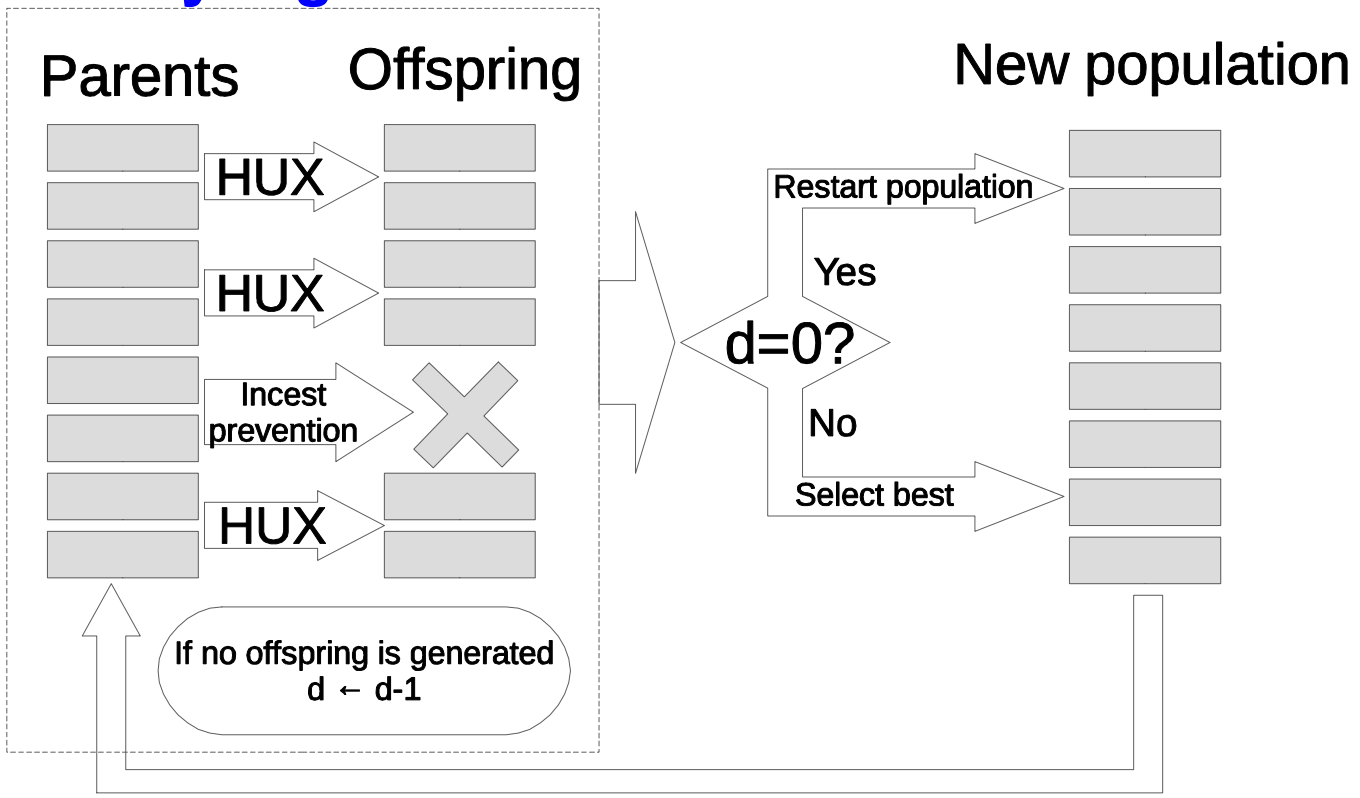
Big Data Preprocessing: MR-EFS

Evolutionary Feature Selection (EFS)

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

Big Data Preprocessing: MR-EFS

Evolutionary Algorithm: CHC



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

Big Data Preprocessing: MR-EFS

Parallelizing FS with MapReduce

Map phase

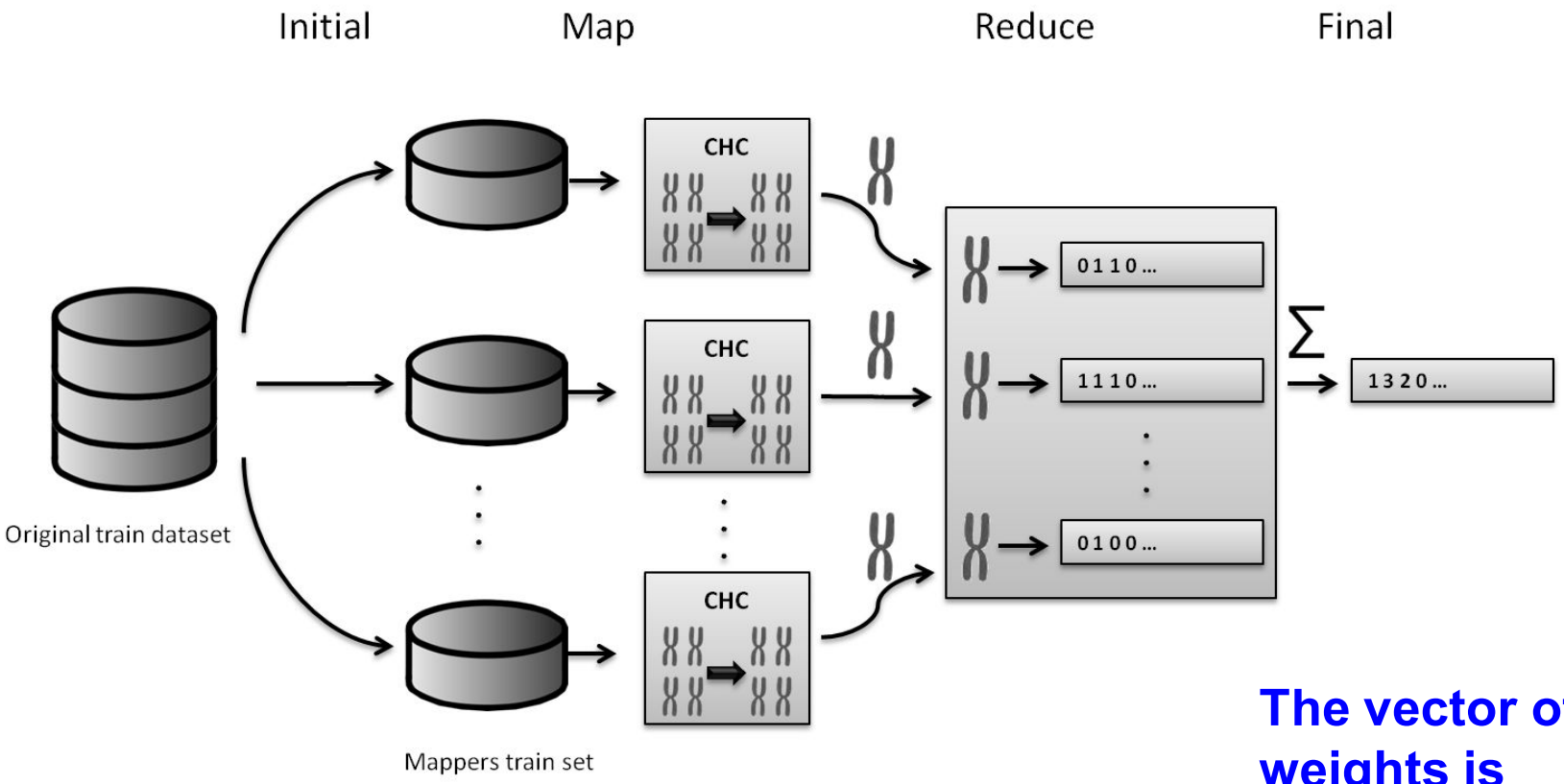
- Each map task uses a subset of the training data.
- It applies an EFS algorithm (CHC) over the subset.
- A k-NN classifier is used for the evaluation of the population.
- Output (best individual):
  - Binary vector, indicating which features are selected.

Reduce phase

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

Big Data Preprocessing: MR-EFS

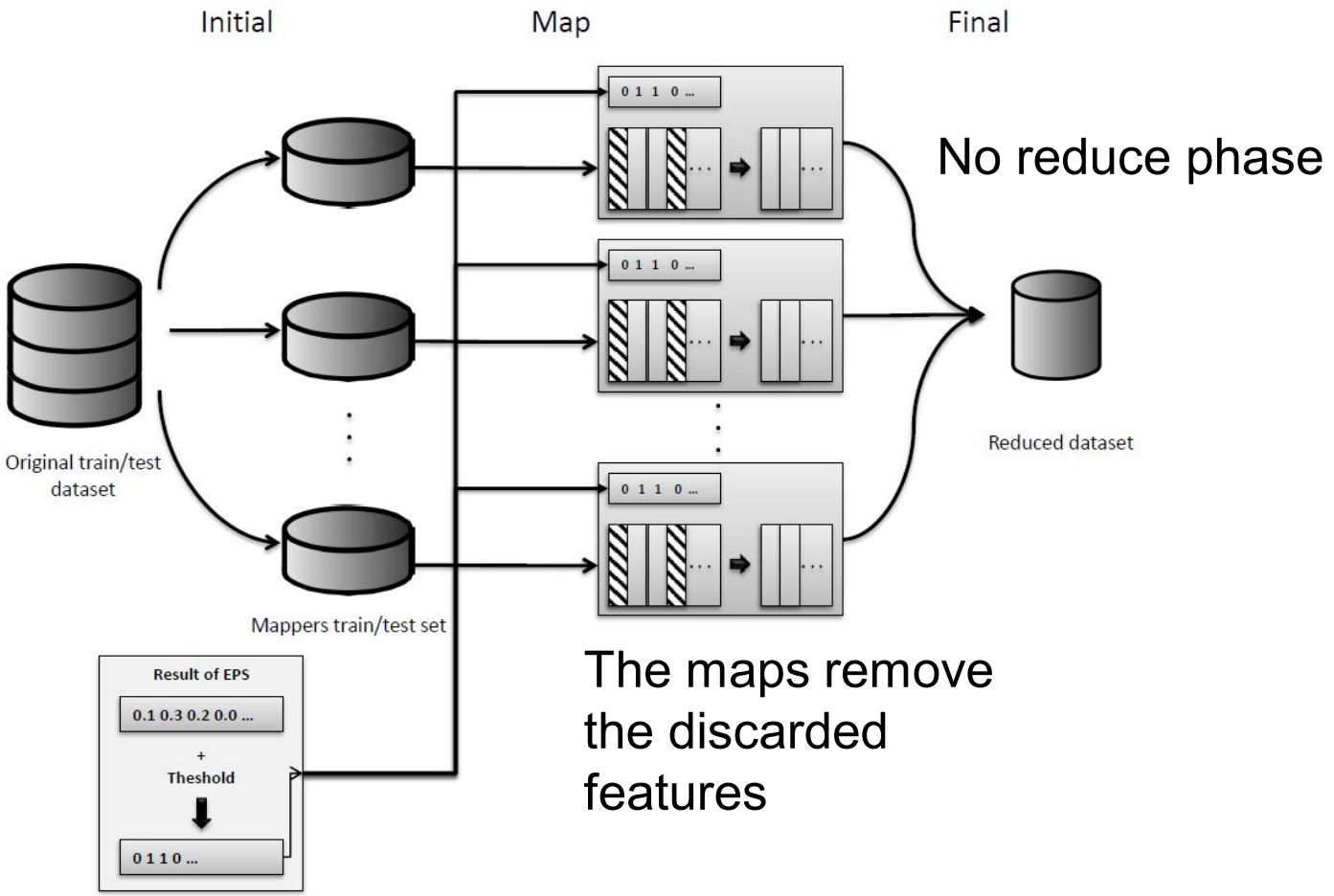
MapReduce EFS process



The vector of weights is binarized with a threshold

Big Data Preprocessing: MR-EFS

Dataset reduction

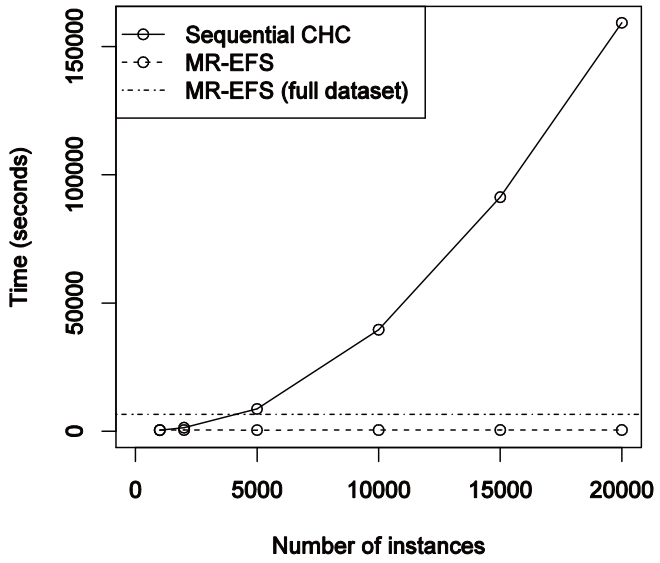


Big Data Preprocessing: MR-EFS

Experimental Study: EFS scalability in MapReduce

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

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



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

Big Data Preprocessing: MR-EFS

Experimental Study: Classification

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

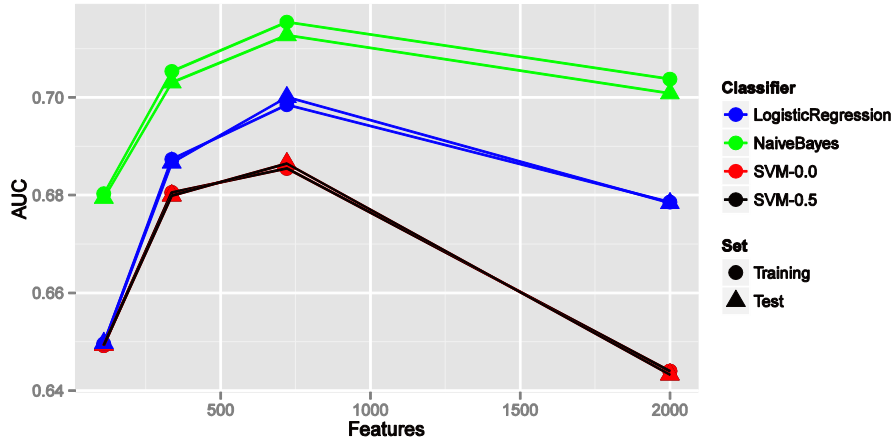
Dataset	Training instances	Test instances	Features	Splits	Instances per split
epsilon	400 000	100 000	2000	512	~780
ECBDL14	31 992 921	2 897 917	631	–	–
ECBDL14-ROS	65 003 913	2 897 917	631	32 768	~1984

Big Data Preprocessing: MR-EFS

Experimental Study: results

Table 4: AUC results for the Spark classifiers using epsilon

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



Big Data Preprocessing: MR-EFS

Experimental Study: Feature selection scalability

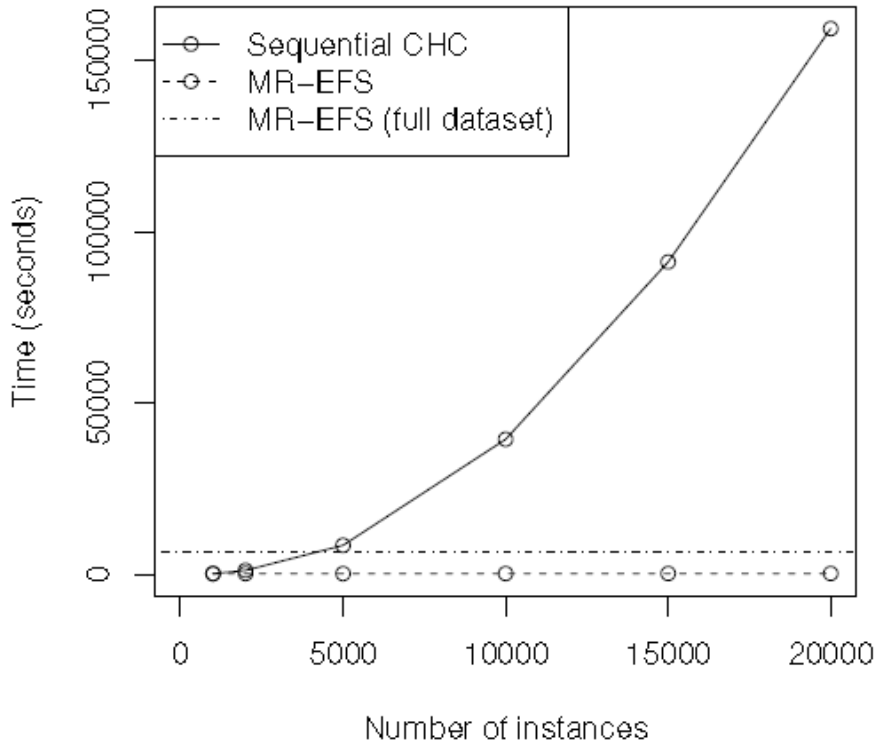


Figure 5: Execution times of the sequential CHC and MR-EFS.

Big Data Preprocessing: MR-EFS

EFS for Big Data: Final Comments

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

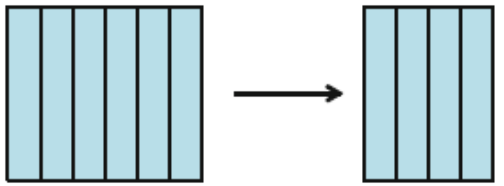
https://github.com/triguero/MR-EFS

Big Data Preprocessing

Describing some Approaches

- Noise Framework
- MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- Evolutionary Undersampling for Imbalanced Big Data
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-FS: 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

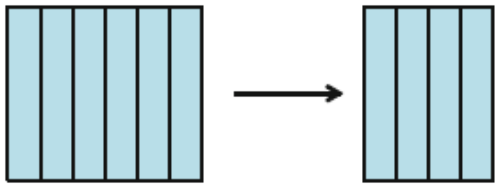
A Feature Selection Framework for Big Data



The curse of Big Dimensionality

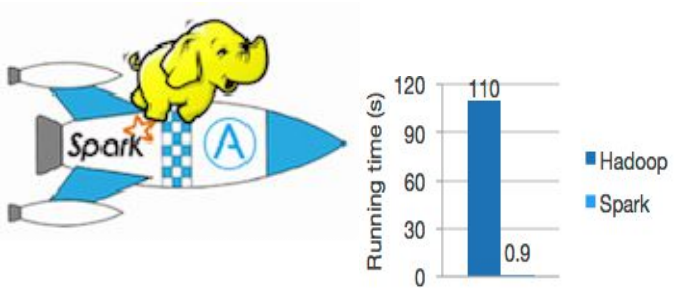
- **"Curse of Big Dimensionality":** A phenomenon influenced by the explosion of features and the combinatorial effects from new large incoming data, where thousand or even millions of features are present.
- **Problem**: Dimensionality reduction techniques (like Feature Selection (FS), or Feature Extraction) can reduce the dimensionality. However, those methods are not expected to scale well with Big Data.
- **Main objective**: to demonstrate that standard FS methods can be parallelized in Big Data platforms like Apache Spark, boosting both performance and accuracying their complexity. Not an **easy** thing!
  - We have also wanted to contribute the emerging Mllib (from Apache Spark), where no complex FS algorithm has been included.

A Feature Selection Framework for Big Data

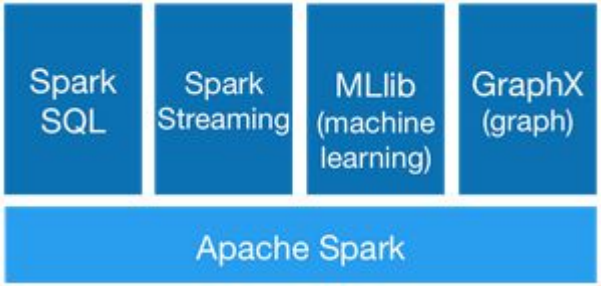


- We propose a distributed implementation of a generic feature selection framework for **Apache Spark** which includes a wide group of well-known Information Theoretic methods.

InMemory
HDFS Hadoop + SPARK



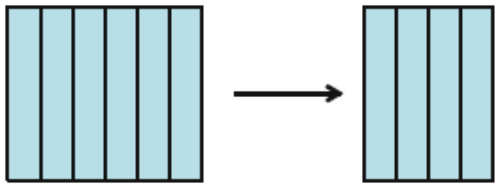
Ecosystem
Apache Spark



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.

H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark: Lightning-Fast Big Data Analytics. O'Reilly Media, Incorporated, 2015.

A Feature Selection Framework for Big Data

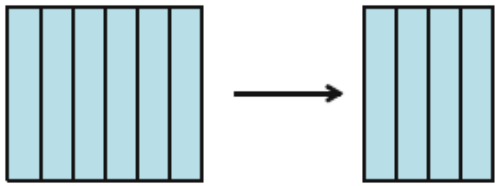


A Feature Selection Framework for Big Data

- **Information measures:** tell us how much information has been acquired by the receiver when he/she gets a message (e.g. Mutual Information and Conditional Mutual Information).
- **Filter methods:** are FS techniques that uses quantitative criterion or index like separability measures or statistical dependences. Independent from the learning phase.
- Some of them are based on information measures like:
  - **Relevance (self-interaction):** usefulness of a feature with respect to the class.
  - **Redundancy (multi-interaction):** carry similar information.

G. Brown, A. Pocock, M.-J. Zhao, and M. Luj´an, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection," J. Mach. Learn. Res., vol. 13, pp. 27–66, Jan. 2012.

A Feature Selection Framework for Big Data



A Feature Selection Framework for Big Data

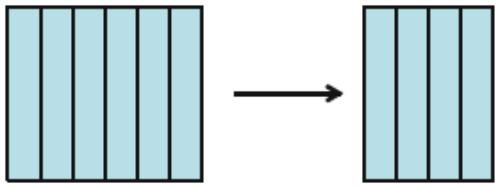
- There are a **wide range of methods** in the literature built on these information measures.
- **Brown et al. Proposal:** generic expression that allows to ensemble multiple information theoretic criteria into a unique FS framework. The criterion are transformed as linear combination of Shannon entropy terms (MI and CMI).

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

relevancy **redundancy** **conditional redundancy**

G. Brown, A. Pocock, M.-J. Zhao, and M. Luj´an, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection," J. Mach. Learn. Res., vol. 13, pp. 27–66, Jan. 2012.

A Feature Selection Framework for Big Data

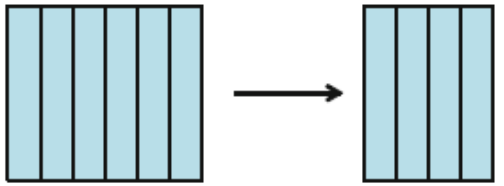


A Feature Selection Framework for Big Data

Criterion name	
Original proposal	**Brown's reformulation**
Mutual Information Maximisation (MIM) [27]	
$J_{mim}(X_i) = I(X_i;Y)$	$J_{mim} = I(X_i;Y) - 0\sum_{X_j \in S} I(X_j;X_i) + 0\sum_{X_j \in S} I(X_j;X_i\|Y)$
Mutual Information FS (MIFS) [26]	
$J_{mifs}(X_i) = I(X_i;Y) - \beta\sum_{X_j \in S} I(X_i;X_j)$	$J_{mifs} = I(X_i;Y) - \beta\sum_{X_j \in S} I(X_j;X_i) + 0\sum_{X_j \in S} I(X_j;X_i\|Y)$
Joint Mutual Information (JMI) [28]	
$J_{jmi}(X_i) = \sum_{X_j \in S} I(X_iX_j;Y)$	$J_{jmi} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{X_j \in S} I(X_j;X_i) + \frac{1}{\|S\|}\sum_{X_j \in S} I(X_j;X_i\|Y)$
Conditional Mutual Information (CMI)	
$J_{cmi} = I(X_i;Y\|S)$	$J_{cmi} = I(X_i;Y) - \sum_{X_j \in S} I(X_j;X_i) + \sum_{X_j \in S} I(X_j;X_i\|Y)$
Minimum-Redundancy Maximum-Relevance (mRMR) [29]	
$J_{mrmr} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{X_j \in S} I(X_j;X_i)$	$J_{mrmr} = I(X_i;Y) - \frac{1}{\|S\|}\sum_{X_j \in S} I(X_j;X_i) + 0\sum_{X_j \in S} I(X_j;X_i\|Y)$
Conditional Mutual Information Maximization (CMIM) [30]	
$J_{cmim} = \min_{X_j \in S}[I(X_i;Y\|X_j)]$	$J_{cmim} = I(X_i;Y) - \max_{X_j \in S}[I(X_j;X_i) - I(X_j;X_i\|Y)]$
Informative Fragments (IF) [31] (equivalent to CMIM)	
$J_{if} = \min_{X_j \in S}[I(X_iX_j;Y) - I(X_j;Y)]$	$J_{if} = J_{cmim} = I(X_i;Y) - \max_{X_j \in S}[I(X_j;X_i) - I(X_j;X_i\|Y)]$
Interaction Capping (ICAP) [32]	
$J_{icap} = I(X_i;Y) - \sum_{X_j \in S} \max[0, I(X_i;X_j) - I(X_i;X_j\|Y)]$	$J_{icap} = I(X_i;Y) - \sum_{X_j \in S} \max[0, I(X_i;X_j) - I(X_i;X_j\|Y)]$

TABLE I: Implemented Information Theoretic criteria: originals and adaptations.

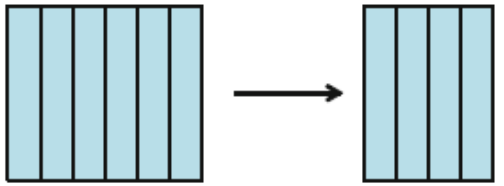
A Feature Selection Framework for Big Data



A Feature Selection Framework for Big Data

- The complexity of the framework is determined by **the computations of relevance and redundancy (two types).**
- **Proposal: complete re-design of Brown's framework**
  - **Columnar transformation:** The access pattern presented by most FS methods is feature-wise. The partitioning scheme of data is quite influential in **Apache Spark**.
  - **Caching variables:** relevance is computed and cached once at the start. The marginal and joint proportions derived from these operation are also cached. This info. is replicated.
  - **Greedy approach:** only one feature is selected in each iteration. The quadratic complexity is transformed to a complexity determined by the number of features selected.

A Feature Selection Framework for Big Data



Main algorithm: Sequential selection

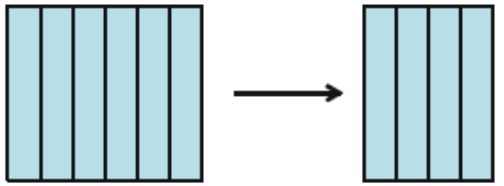
1) **Columnar transformation:** transpose data (line 1, MapPartitions and sorting)
2) **Compute relevances:** compute the initial relevances for all the features and initialize the criteria. These info is cached. (line 3-4, MapReduce process)
3) **Compute redundancies:** update the accumulated redundancies (simple and conditional) of the criteria. The best candidate feature in each iteration is selected. (7-12, MapReduce process)

Algorithm 1 Main FS Algorithm

Input: D Data set
Input: $|S_\theta|$ Number of features to select.
Input: $npart$ Number of partitions to set.
Input: $cindex$ Index of the output feature.
Output: S_θ Index list of selected features

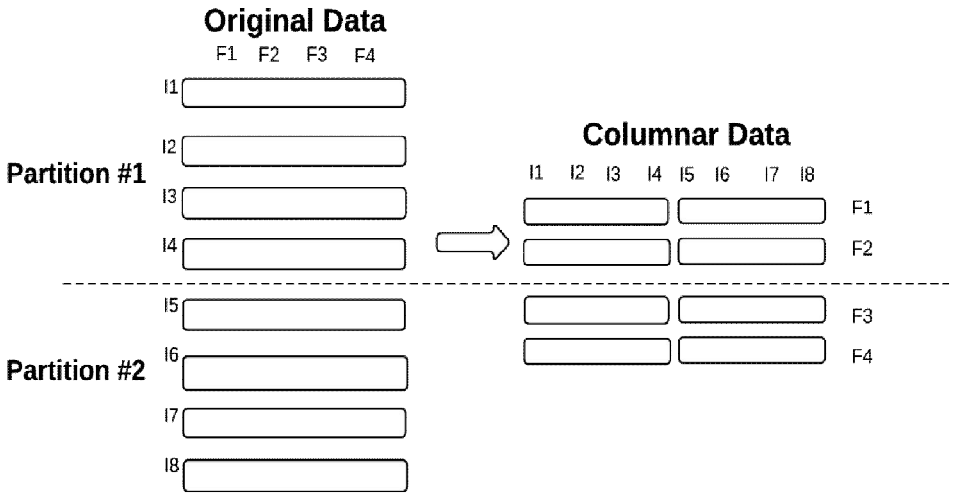
$D_c \leftarrow columnarTransformation(D, |S_\theta|, npart)$
$ni \leftarrow D.nrows; nf \leftarrow D.ncols$
$relvs \leftarrow computeRelevances(D_c, cindex, ni)$
$criteria \leftarrow initCriteria(relvs)$
$p_{best} \leftarrow max(criteria)$
$S \leftarrow Set(p_{best})$
while $|S| < |S_\theta|$ **do**
  $reds \leftarrow computeRedundancies(D_c, p_{best}.index)$
  $criteria \leftarrow updateCriteria(criteria, reds)$
  $p_{best} \leftarrow max(criteria)$
  $S \leftarrow addTo(p_{best}, S)$
end while
$return(S)$

A Feature Selection Framework for Big Data

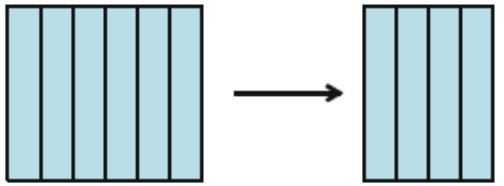


Columnar Transformation

- The idea behind this transformation is to **transpose the local data matrix** provided by each partition. **Take advantage of data locality, reduce shuffle.**
- The result of this operation is a **new matrix with one row per feature**.
- After the transformation, the partial rows **are sorted by key (feature)** in order to put together those values for the same feature that were allocated in different partitions.
- Afterwards, the columnar data is **cached and re-used** in the next steps.



A Feature Selection Framework for Big Data



Compute Relevances

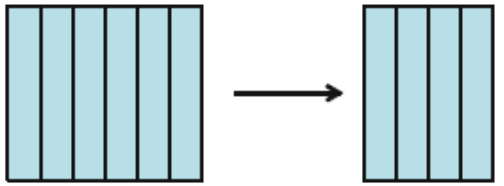
- **Compute the relevance (MI) between all the input features and the class.**
- Both relevance and redundancy is computed for each feature independently. This is done by **replicating the class variable** (line 1-2, broadcast).
- This computes **3-dimensional histograms** between all non-selected features and one variable (line 4, compute histograms).
- **Joint and marginal proportions** are generated from the histograms by computing proportions for joint and by aggregating proportions by row for marginal (line 5-6).

Algorithm 3 Compute mutual information between the set of features X and Y. *(computeRelevances)*

Input: D_c Dataset, an RDD of (index, (block, vector)).
Input: $yind$ Index of Y.
Input: ni Number of instances.
Output: MI values for all input features.

1: $ycol \leftarrow data.lookup(yind)$
2: $bycol \leftarrow broadcast(ycol)$
3: $counter \leftarrow broadcast(D_c.getMaxByFeature())$
4: $histograms \leftarrow getHistograms(D_c, yind, bycol, null, null)$
5: $joint \leftarrow histograms.getProportions(ni)$
6: $marginal \leftarrow joint.aggregateByRow().getProportions(ni)$
7: $return(computeMutualInfo(D_c, yind, null))$

A Feature Selection Framework for Big Data



Compute Redundancies

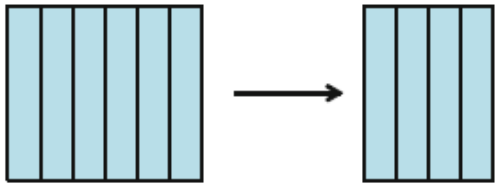
- **Compute the simple and conditional between the best feature, each candidate one and the class (conditional).**
- The **last selected feature (pbest)** is collected and replicated across all the nodes. (line 1-2, broadcast).
- This also computes **3-dimensional histograms** between between all non-selected features and two secondary variables (line 3, compute histograms). The class is already broadcasted.

Algorithm 4 Compute CMI and MI between p_{best}, the set of candidate features, and Y. *(computeRedundancies)*

Input: D_c Dataset, an RDD of (index, (block, vector)).
Input: $jind$ Index of p_{best}.
Output: CMI values for all input features.

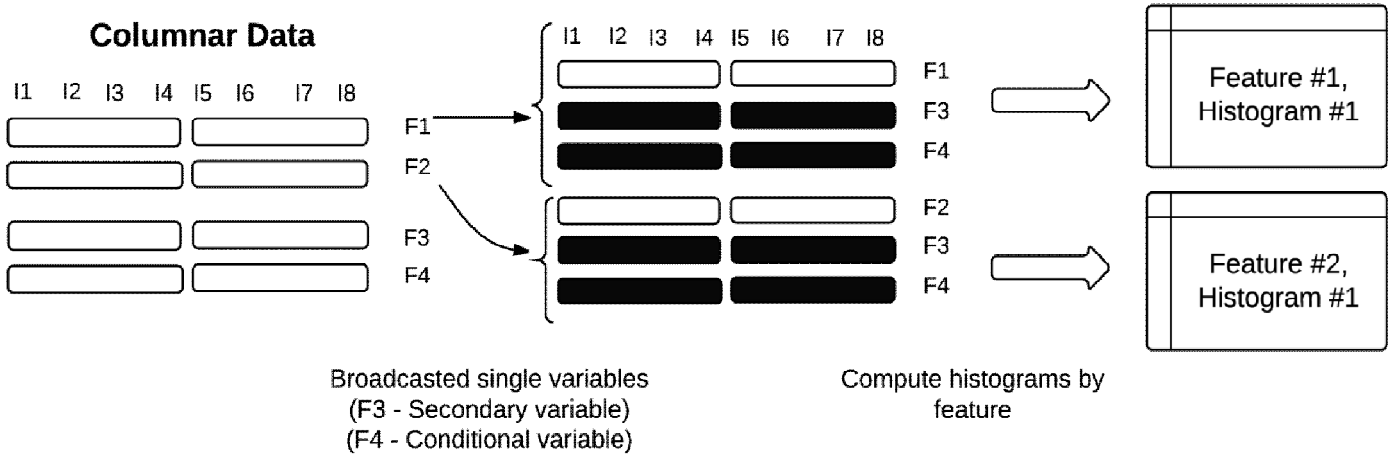
1: $jcol \leftarrow data.lookup(jind)$
2: $bjcol \leftarrow broadcast(jcol)$
3: $histograms \leftarrow getHistograms(D_c, jind, bjcol, yind, bycol)$
4: $return(computeMutualInfo(D_c, jind, yind))$

A Feature Selection Framework for Big Data

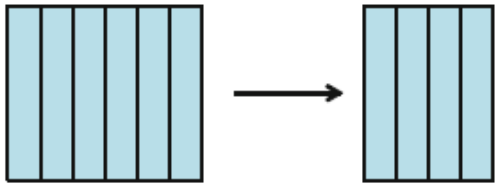


Histograms Creation

- This computes **3-dimensional histograms for the set of candidate features with respect to two secondary variables**, which later will be used to compute MI and CMI (MapPartition and ReduceByKey).
- If it the third variable is not provided, **2-dimensional histograms** are generated.
- Each partition generates a histogram by feature. Then, the histograms are aggregated by key (feature index) to yield the total histograms.

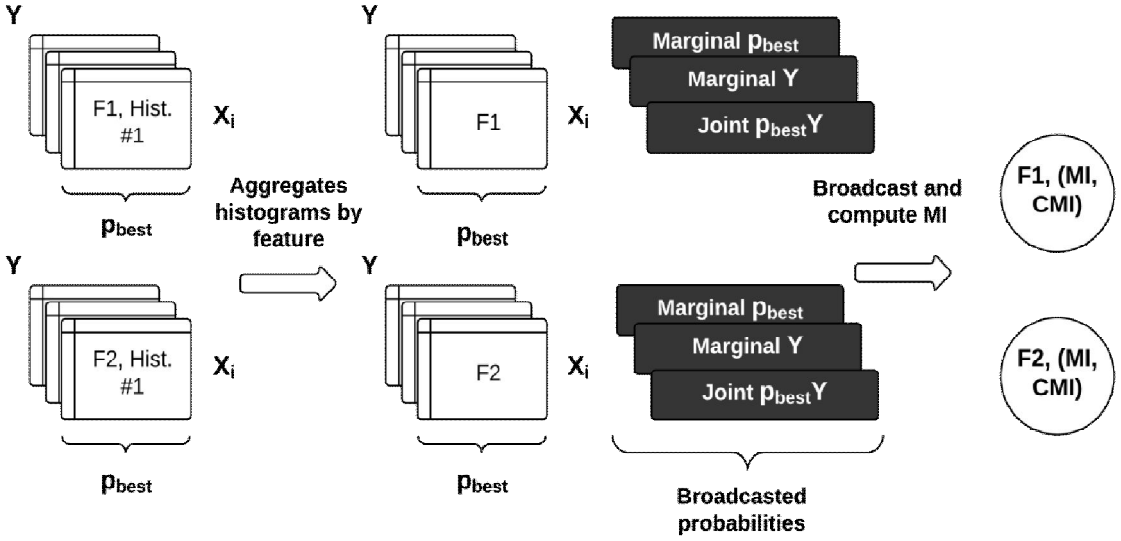


A Feature Selection Framework for Big Data

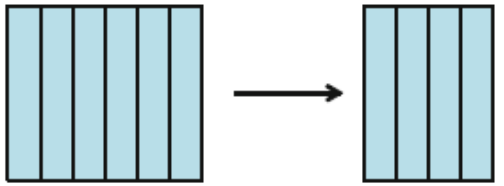


Computing Simple and Conditional Mutual Information

- The algorithm takes as input **the indices of the secondary variables (the class and/or the last selected feature),** and all previously computed **histograms** (MapReduce process).
- As a preliminary step, the algorithm broadcasts the marginal and joint matrices that correspond to these variables, and computes the rest ones.
- A **map phase** is then started on each histogram/feature to compute the final MI and CMI value per feature.



Experimental Analysis



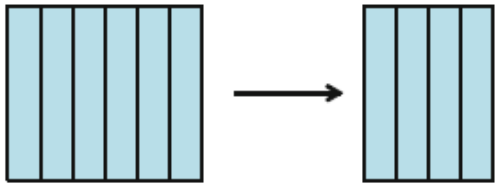
Experimental Framework

- **Datasets:** Five datasets, described below:

Data Set	#Train Ex.	#Test Ex.	#Atts.	#Cl.	Sparse
epsilon	400 000	100 000	2000	2	No
dna	79 739 293	10 000 000	200	2	No
ECBDL14	65 003 913	2 897 917	630	2	No
url	1 916 904	479 226	3 231 961	2	Yes
kddb	19 264 097	748 401	29 890 095	2	Yes

- **Parameters:** FS algorithm = mRMR, level of parallelism = 864 partitions.
- **Classifier:** Naive Bayes and SVM (default parameters), from MLLIB.
- **Measures:** selection time, classification time and classification accuracy (AU-ROC).
- **Hardware:** 18 nodes (12 cores per node), 64 GB RAM.
- **Software**: Hadoop 2.5 and Apache Spark 1.2.0.

Experimental Analysis



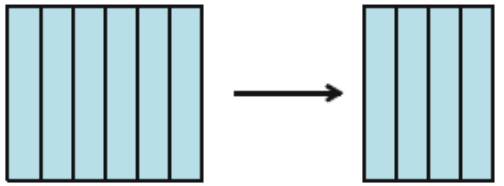
Performance results: Selection time

- Our algorithm yields **competitive results** in all cases regardless of the number of iterations employed (represented by the threshold value)
- For all cases, **100 features selected in less than one hour** (even for ECBDL14 and URL).

TABLE IV: Selection time by dataset and threshold (in seconds)

# Features	kddb	url	dna	ECBDL14	epsilon
10	283.61	94.06	97.83	332.90	111.42
25	774.43	186.22	148.78	596.31	173.39
50	1365.82	333.70	411.84	1084.58	292.07
100	2789.55	660.48	828.35	2420.94	542.05

Experimental Analysis



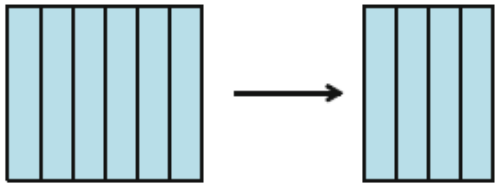
Performance results: Selection time (sequential vs. distributed)

- Comparison between **our method and the sequential version** developed by Brown's lab.
- Data: Samples from ***dna*** have been generated with different ratios of instances. Results in italics were estimated because of memory lack.
- **Maximum level of parallelism** = 200 (the number of features).

TABLE IV: Selection time by dataset and threshold (in seconds)

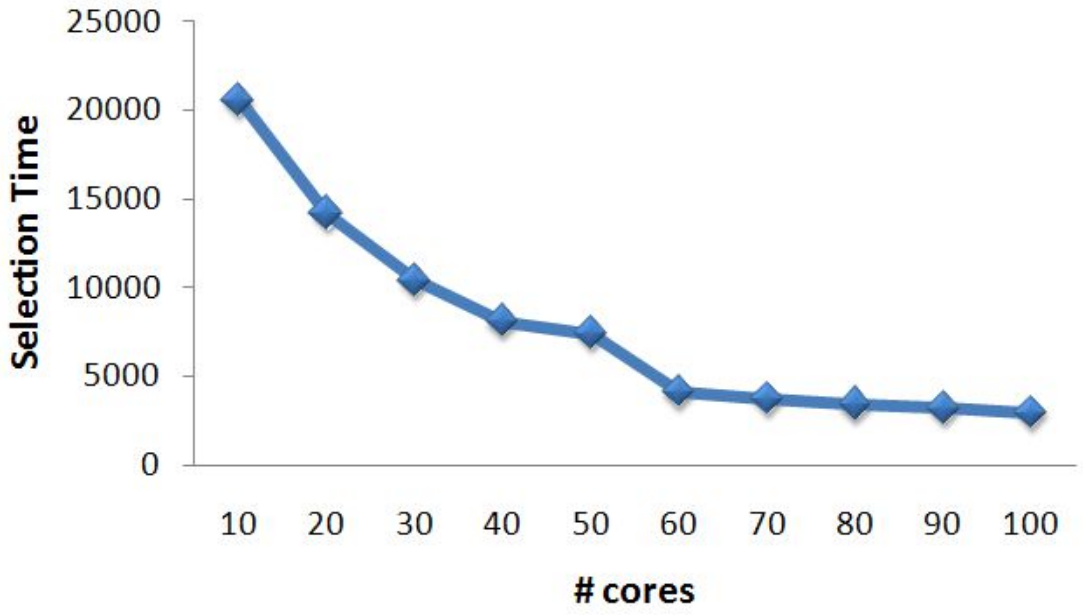
# Features	kddb	url	dna	ECBDL14	epsilon
10	283.61	94.06	97.83	332.90	111.42
25	774.43	186.22	148.78	596.31	173.39
50	1365.82	333.70	411.84	1084.58	292.07
100	2789.55	660.48	828.35	2420.94	542.05

Experimental Analysis

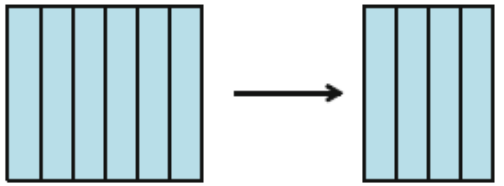


Performance results: Selection time (scalability study)

- Additional **study of scalability** varying the number of cores used (from 10 to 100).
- For this study, **ECBDL14 (the largest dense dataset)** was used as a reference.

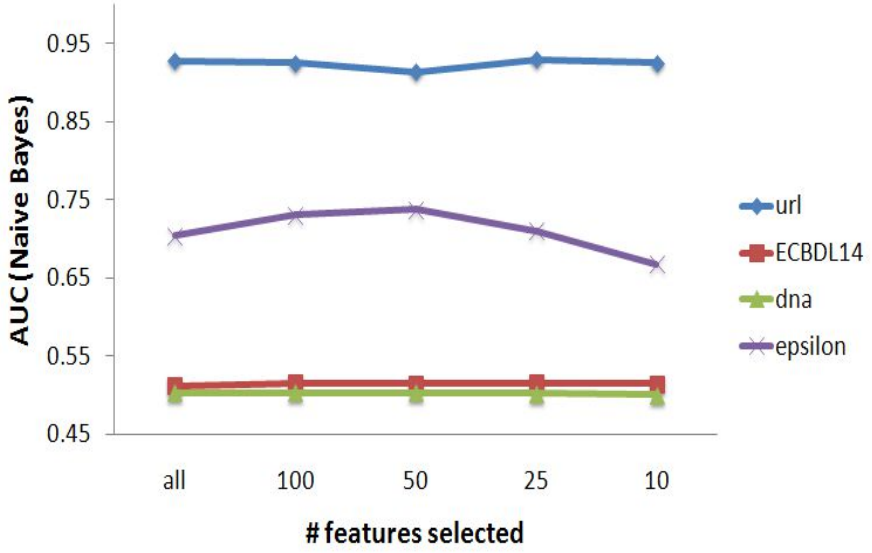


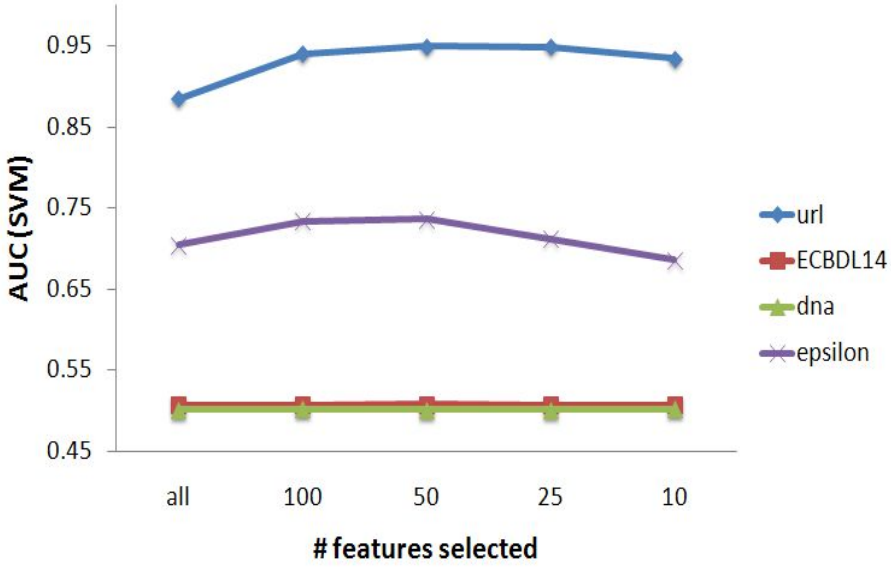
Experimental Analysis



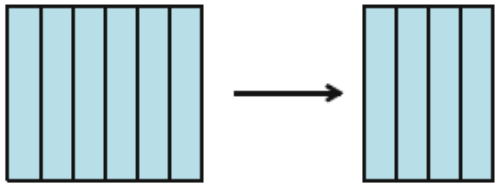
Performance results: Classification accuracy

- We performed a study about the **usefulness of our FS solution** on large-scale classification. *kddb* was omitted because the classifiers do not work well with such big dimensionality.
- **Important improvement** over *url* and *epsilon,* but not for *dna* or *ECBDL14* (low number of features).



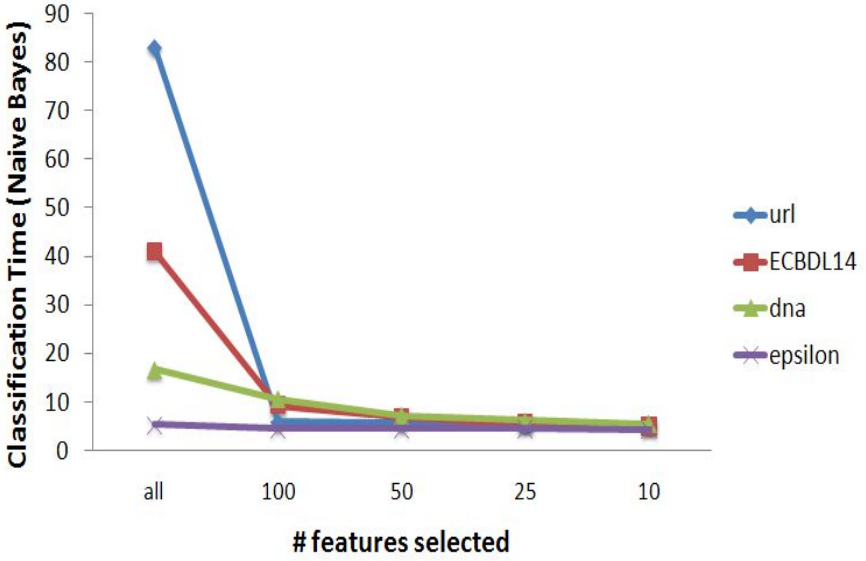


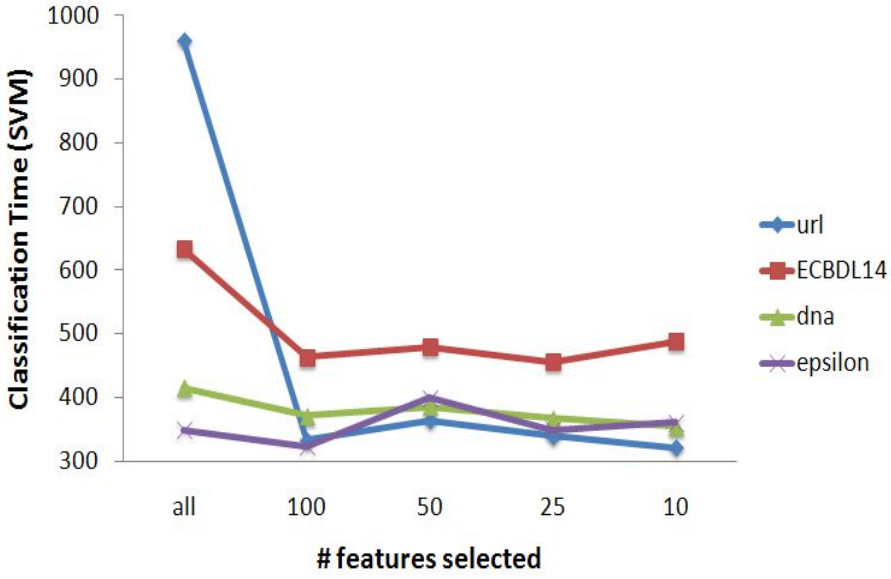
Experimental Analysis



Performance results: Classification time

- This study show the **classification time** employed in the **training phase** for different datasets and thresholds.
- The results demonstrate that **the simplicity and the performance** of the generated models is normally improved after applying FS, in spite of using a **small percentage of the original set of features**.





Final Comments

- We have proposed a **generic FS framework for Big Data** based on Information Theory, adapting a previous one proposed by Brown et al.
- This has implied a **deep redesign** of the original proposal.
- Adapting FS methods to Big Data is **not a trivial task.**
- Our solution has revealed to perform well with **two dimensions of Big Data** (samples and features), obtaining competitive performance results when dealing with **ultra-high dimensional datasets** as well as those with a huge number of samples.
- It is available at Spark Packages http://spark-packages.org/

Final Comments

https://github.com/sramirez


Sergio Ramírez

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

Big Data Preprocessing

Describing some Approaches

- Noise Framework
- MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- Evolutionary Undersampling for Imbalanced Big Data
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-FS: 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.

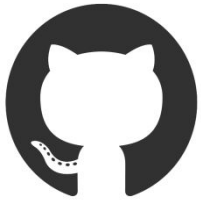
$$\mathrm{mRMR} = \max_{S} \left[\frac{1}{|S|} \sum_{f_i \in S} I(f_i; c) - \frac{1}{|S|^2} \sum_{f_i, f_j \in S} I(f_i; f_j) \right] .$$

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

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.

Final Comments

BIG DATA Preprocessing

Discretization and feature selection are two important approaches for big data preprocessing.

There are a few approaches, and they will increase in the near future.

Methods	Category	# Features	# Instances	Size (GB)	Framework
[60]	FS	630	65,003,913	305.1196	Hadoop MapReduce
[61]	FS	1,156	5,670,000	48.8350	MPI
[59]	FS	29,890,095	19,264,097	4.1623	C++/MATLAB
[58]	FS	100,000	10,000,000	1.4901	MapReduce
[62]	FS	100	1,600,000	1.1921	Apache Spark
[63]	FS	54,675	2,096	0.8538	Hadoop MapReduce
[64]	FS	54	581,012	0.2338	Hadoop MapReduce
[65]	FS	20	1,000,000	0.1490	MapReduce
[66]	FS	-	-	0.0976	Hadoop MapReduce
[67]	FS	256	38,232	0.0729	Hadoop MapReduce
[68]	FS	52	5,253	0.0020	Hadoop MapReduce
[69]	FS	-	-	0.0000	Hadoop MapReduce
[70]	FS	-	-	0.0000	Hadoop MapReduce
[71]	FS	-	-	0.0000	Hadoop MapReduce
[80]	discretization	630	65,003,913	305.1196	Apache Spark
[81]	discretization	-	-	4.0000	Hadoop MapReduce

RESEARCH

Big data preprocessing: Methods and Prospects

S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez and F. Herrera

Under Submission

From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. **Discretization**
9. Final Comments

Big Data Preprocessing

Describing some Approaches

- Noise Framework
- MRPR: A Combined MapReduce-Windowing Two-Level Parallel Scheme for Evolutionary Prototype Generation
- Evolutionary Undersampling for Imbalanced Big Data
- MR-EFS: Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach
- Spark-FS: 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

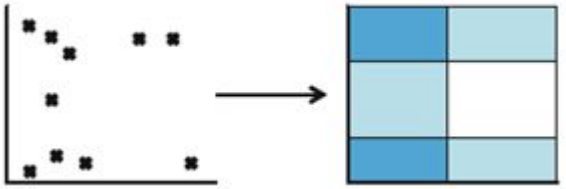
Big Data Preprocessing: Spark-MDLP

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 Conference, Helsinki, August 2015, 33-40.

Sergio Ramírez-Gallego, Salvador García, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, Francisco Herrera
Data discretization: Taxonomy and big data challenge
Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 6 (1), 2016, 5-21

Distributed Entropy Minimization Discretizer

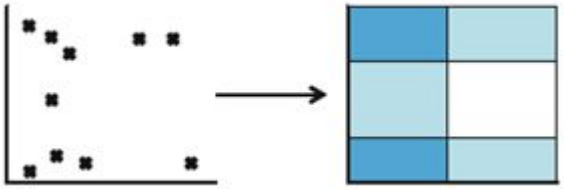


Discretization: An Entropy Minimization Approach

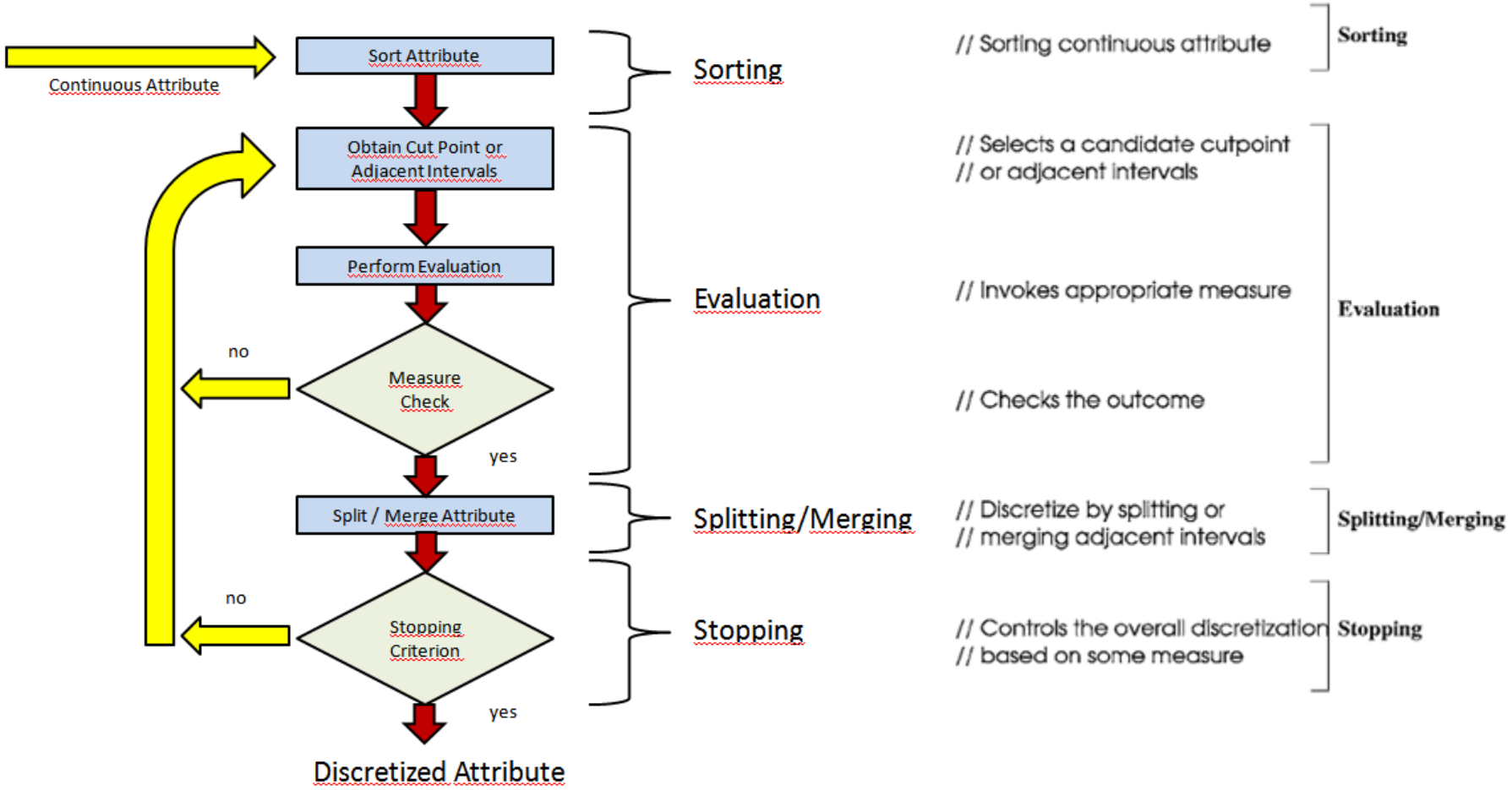
- **Discretization**: transforms numerical attributes into discrete or nominal attributes with a finite number of intervals.
- **Main objective**: find the best set of points/intervals according to a quality measure (e.g.: inconsistency or **entropy**).
- Optimal discretization is NP-complete, determined 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).

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.

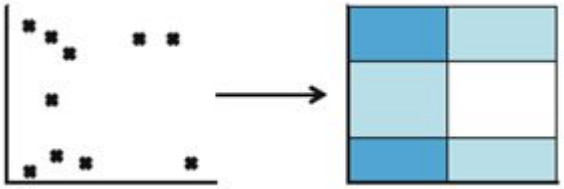
Distributed Entropy Minimization Discretizer



Stages in the discretization process

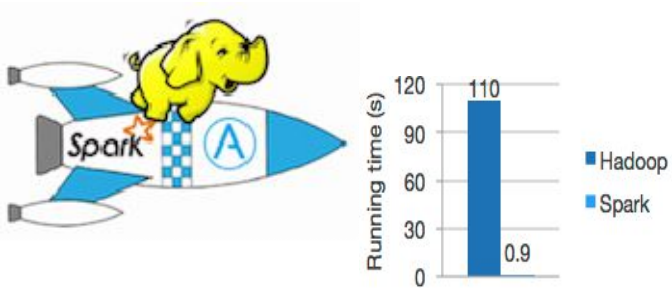


Distributed Entropy Minimization Discretizer

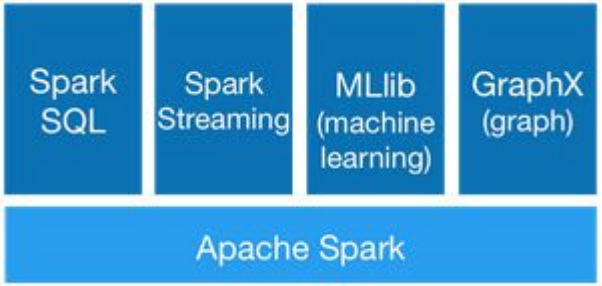


- We propose a distributed implementation of the entropy minimization discretizer proposed by Fayyad and Irani using **Apache Spark** platform.

InMemory
HDFS Hadoop + SPARK



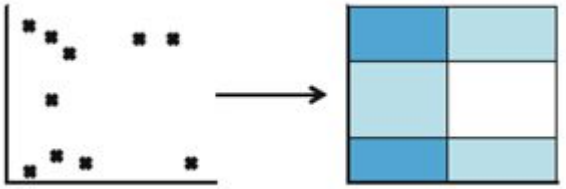
Ecosystem
Apache Spark



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.

Distributed Entropy Minimization Discretizer



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 (attribute A, new boundary point b_α, partition S, minimize EP)

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

- **Stop criterion:** MDLP criterion

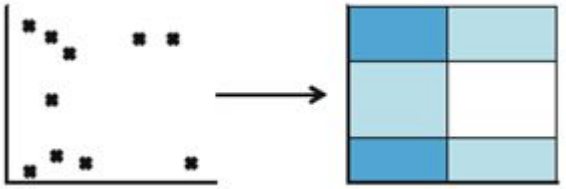
$$G(A, b_\alpha, S) > \frac{\log_2(N-1)}{N} + \frac{\Delta(A, b_\alpha, S)}{N},$$

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

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

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.

Distributed Entropy Minimization Discretizer

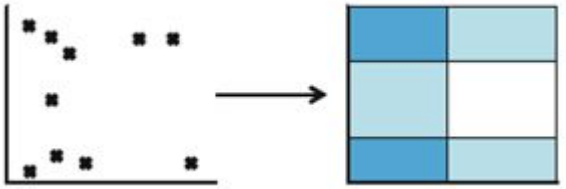


Distributed MDLP Discretization

- **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: MDLP complete re-design**
  - **Sort all points in the dataset using a single distributed operation using a SPARK primitive.**
  - **Evaluates boundary points (per feature) in an parallel way.**

Distributed Entropy Minimization Discretizer

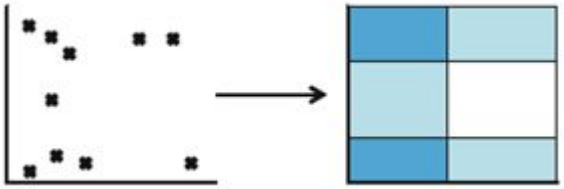


Distributed MDLP Discretization

- **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: MDLP complete re-design**
  - Sort all points in the dataset using a single distributed operation using a SPARK primitive.
  - 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 (to evaluate the boundary points, …)

Distributed Entropy Minimization Discretizer



Distributed MDLP Discretization:

main procedure (6 main steps)

1) **Distinct points:** it creates tuples where key is formed by the feature index and the value (line 2-9, MapReduce process)

   Value: frequency vector for classes.

2) **Sorting:** distinct points are sorted by key (feature, value) (line 10, sort_by_key, Spark primitive)

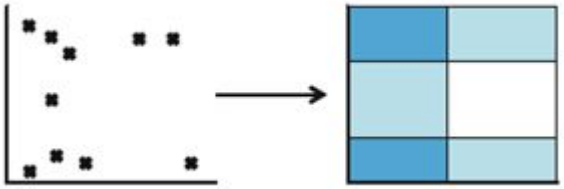
3) **Boundary points:** boundary points are calculated evaluating consecutives points (11-12, MapPartition, Spark primitive)

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

1: $comb \leftarrow$
2: **map** $s \in S$
3: $v \leftarrow zeros(k)$
4: $v(c) \leftarrow 1$
5: **for all** $A \in M$ **do**
6: $EMIT < (A, A(s)), v >$
7: **end for**
8: **end map**
9: $distinct \leftarrow reduce(comb,\ sum\_vectors)$
10: $sorted \leftarrow sort\_by\_key(distinct)$
11: $first \leftarrow first\_by\_part(sorted)$
12: $boundaries \leftarrow get\_boundary\_points(sorted, first)$
13: $boundaries \leftarrow$
14: **map** $b \in boundaries$
15: $< (att, point,)q > \leftarrow b$
16: $EMIT < (att, (point, q)) >$
17: **end map**
18: $(small, big) \leftarrow divide\_attributes(boundaries, maxcand)$
19: $sthresholds \leftarrow select\_thresholds(small, maxbins, maxcand)$
20: **for all** $att \in keys(big)$ **do**
21: $bthresholds \leftarrow bthresholds + select\_thresholds(big(att), maxbins, maxcand)$
22: **end for**
23: $return(union(bthresholds, sthresholds))$

Distributed Entropy Minimization Discretizer



Distributed MDLP Discretization:

main procedure (6 main steps)

4) **Points grouped by feature:** grouped and new key: only the attribute index (lines 14-17 Map)

5) **Attributes divided by size:** depends on the number of boundary points (> 10,000 points, new process MapReduce per attribute) (line 18, Count_by_key Spark primitive)

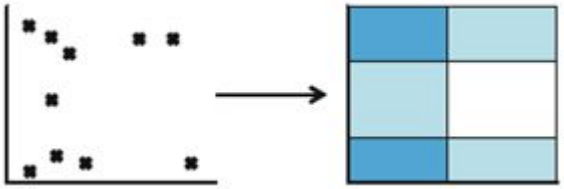
6) **Points selection and MDLP evaluation** (line 19-22, 19 MapPartition with only one Partition per attribute ("small" attributes), 20-22 MapPartition with several partitions per attribute ("big" attributes))

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

1: $comb \leftarrow$
2: **map** $s \in S$
3: $v \leftarrow zeros(k)$
4: $v(c) \leftarrow 1$
5: **for all** $A \in M$ **do**
6: $EMIT < (A, A(s)), v >$
7: **end for**
8: **end map**
9: $distinct \leftarrow reduce(comb,\ sum\_vectors)$
10: $sorted \leftarrow sort\_by\_key(distinct)$
11: $first \leftarrow first\_by\_part(sorted)$
12: $boundaries \leftarrow get\_boundary\_points(sorted, first)$
13: $boundaries \leftarrow$
14: **map** $b \in boundaries$
15: $< (att, point,)q > \leftarrow b$
16: $EMIT < (att, (point, q)) >$
17: **end map**
18: $(small, big) \leftarrow divide\_attributes(boundaries, maxcand)$
19: $sthresholds \leftarrow select\_thresholds(small, maxbins, maxcand)$
20: **for all** $att \in keys(big)$ **do**
21: $bthresholds \leftarrow bthresholds + select\_thresholds(big(att), maxbins, maxcand)$
22: **end for**
23: $return(union(bthresholds, sthresholds))$

Distributed Entropy Minimization Discretizer



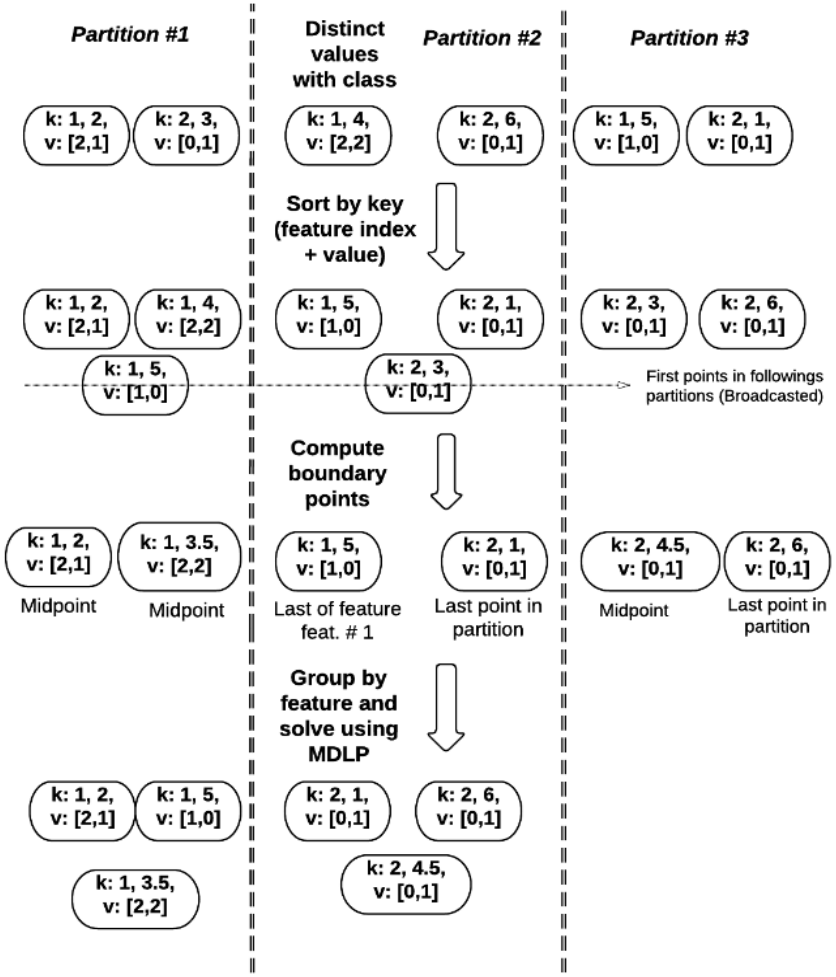
Distributed MDLP Discretization: main procedure

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

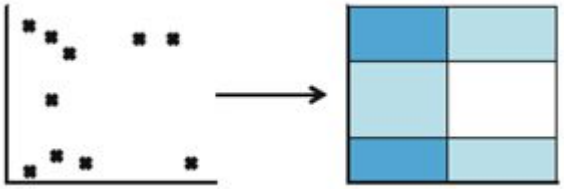
Sorting: distinct points are sorted by key. For the next step, the first points in each partition are sent to the following partition.

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.

Points grouped feature and MLDP selection and evaluation: parallelized by feature



Distributed Entropy Minimization Discretizer



Distributed MDLP Discretization: MDLP evaluation (small)

One partition per attribute

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:

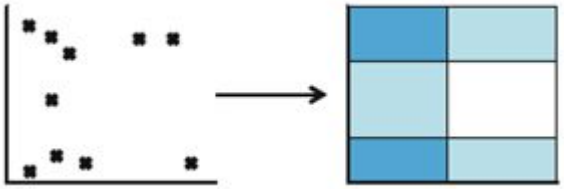
(point, frequency, left, right)

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

1: $total \leftarrow sum\_freqs(candidates)$
2: $leftacc \leftarrow ()$
3: **for** $< point, q > \in candidates$ **do**
4: $\quad leftacc \leftarrow leftacc + q$
5: $\quad freqs \leftarrow freqs + (point, q, leftacc, total - leftacc)$
6: **end for**
7: $return(select\_best\_cut(candidates, freqs))$

Distributed Entropy Minimization Discretizer



Distributed MDLP Discretization: MDLP evaluation (big)

Several partitions per attribute

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,

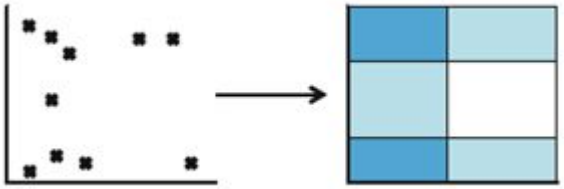
5) The point is evaluated as: (point, frequency, left, right).

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

```
1: npart ← round(|candidates|/maxcand)
2: candidates ← coalesce(candidates, npart)
3: totalpart ←
4: map partitions partition ∈ candidates
5:    return(sum(partition))
6: end map
7: total ← sum(totalpart)
8: freqs ←
9: map partitions partition ∈ candidates
10:   index ← get_index(partition)
11:   lefttotal ← ()
12:   freqs ← ()
13:   for i = 0 until index do
14:      lefttotal ← lefttotal + totalpart(i)
15:   end for
16:   for all < point, q >∈ partition do
17:      freqs ← freqs+(point, q, leftotal+q, total−leftotal)
18:   end for
19:   return(freqs)
20: end map
21: return(select_best_cut(candidates, freqs))
```

Experimental Analysis



Experimental Framework

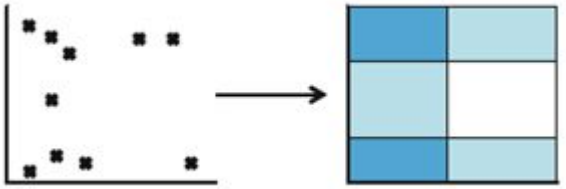
- **Datasets:** Two huge datasets (*ECBDL14 and epsilon*)

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.

Experimental Analysis



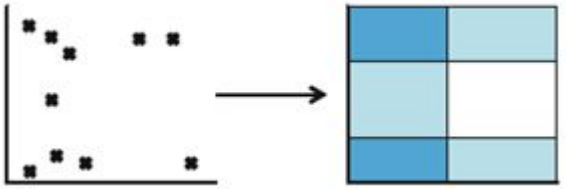
Performance results: Classification accuracy

- **NB** use a probability per value (without discretization)
- **NB-disc** – Our model
- **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**

Experimental Analysis



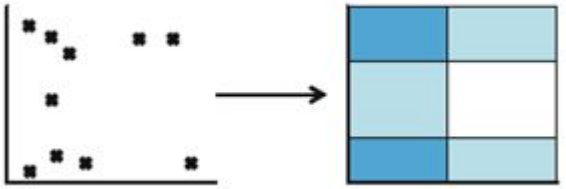
Performance results: Discretization time

- **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**

Experimental Analysis



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**

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**

Final Comments

- We have proposed a **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**

- The experimental results has demonstrated **the improvement in accuracy and time** (both classification and discretization) with respect to the sequential proposal.

- It is available at Spark Packages http://spark-packages.org/

Final Comments

Discretization

https://github.com/sramirez

Distributed Minimum Description Length Discretizer for Spark

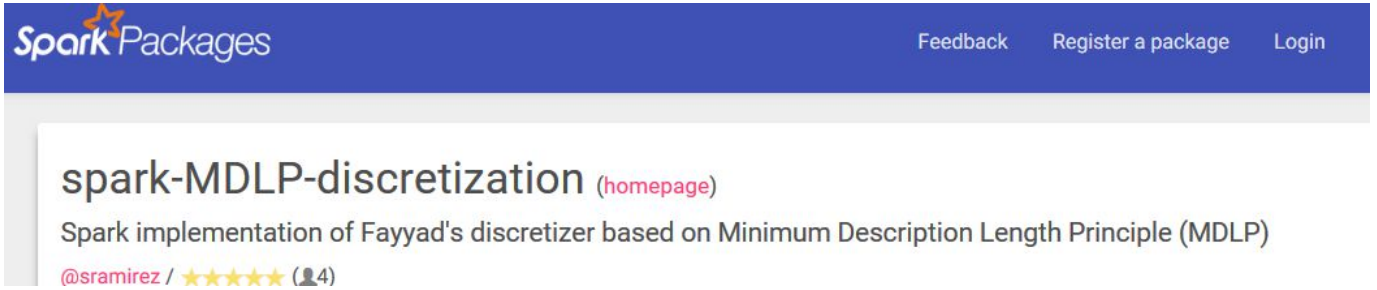
GitHub sramirez / spark-MDLP-discretization

Sergio Ramírez

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.



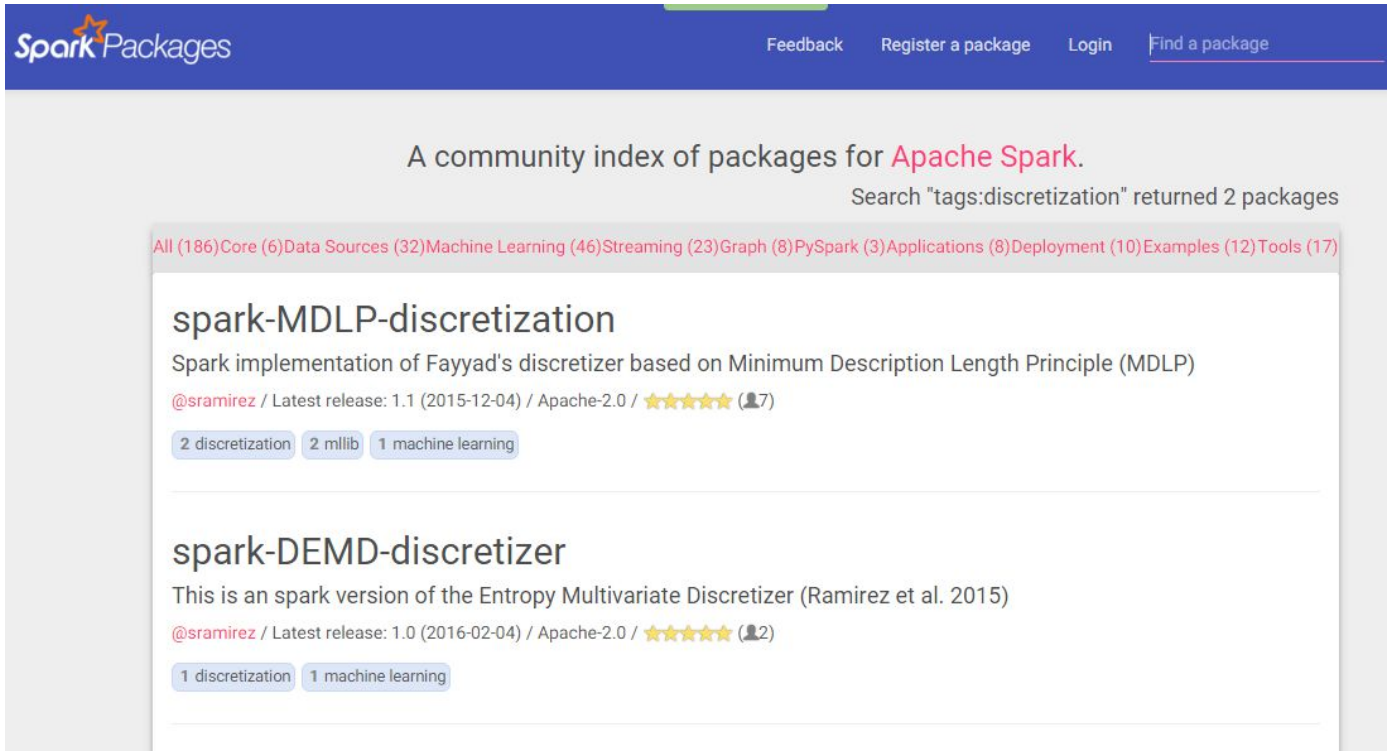
Final Comments

Discretization (2 algorithms)

A Distributed Evolutionary Multivariate Discretizer (DEMD)

https://github.com/sramirez/spark-DEMD-discretizer

Sergio Ramírez

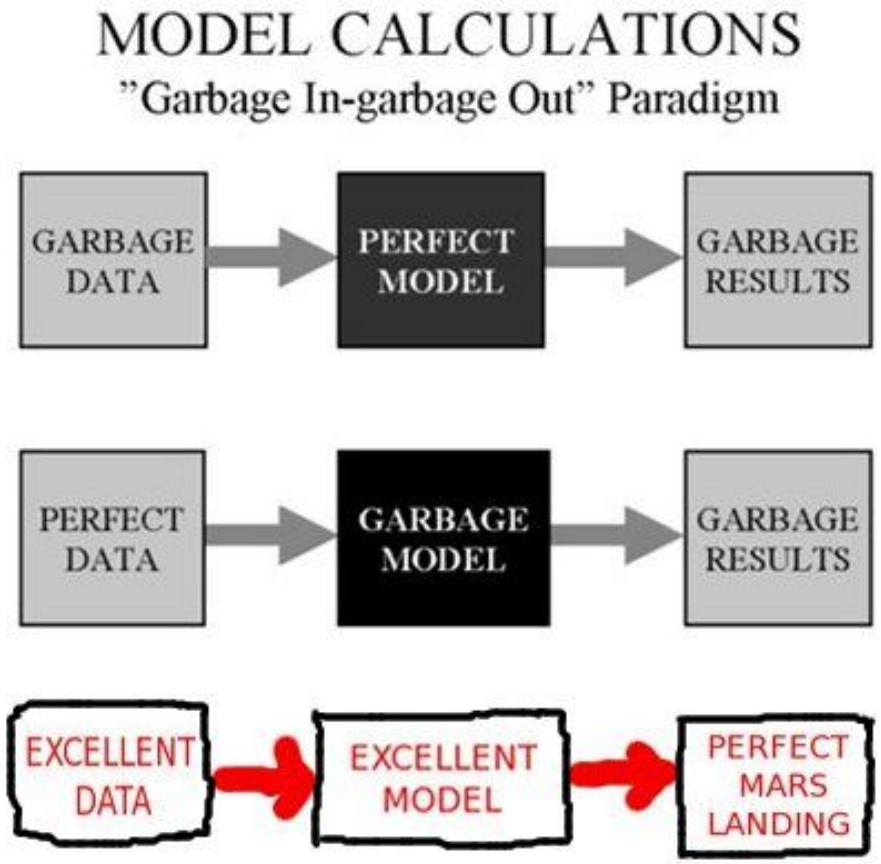


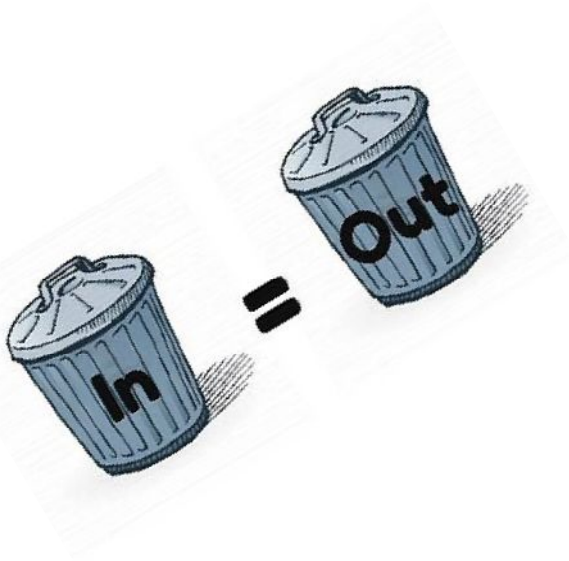
From Big Data to Smart Data: Big Data Preprocessing

1. Smart Data: Towards Quality Data
2. Data Preprocessing. Importance
3. Data Preparation and Data Reduction
4. A First Look on Big Data Preprocessing
5. Imperfect Data
6. Instance Reduction
7. Feature Selection
8. Discretization
9. **Final Comments**

Final Comments

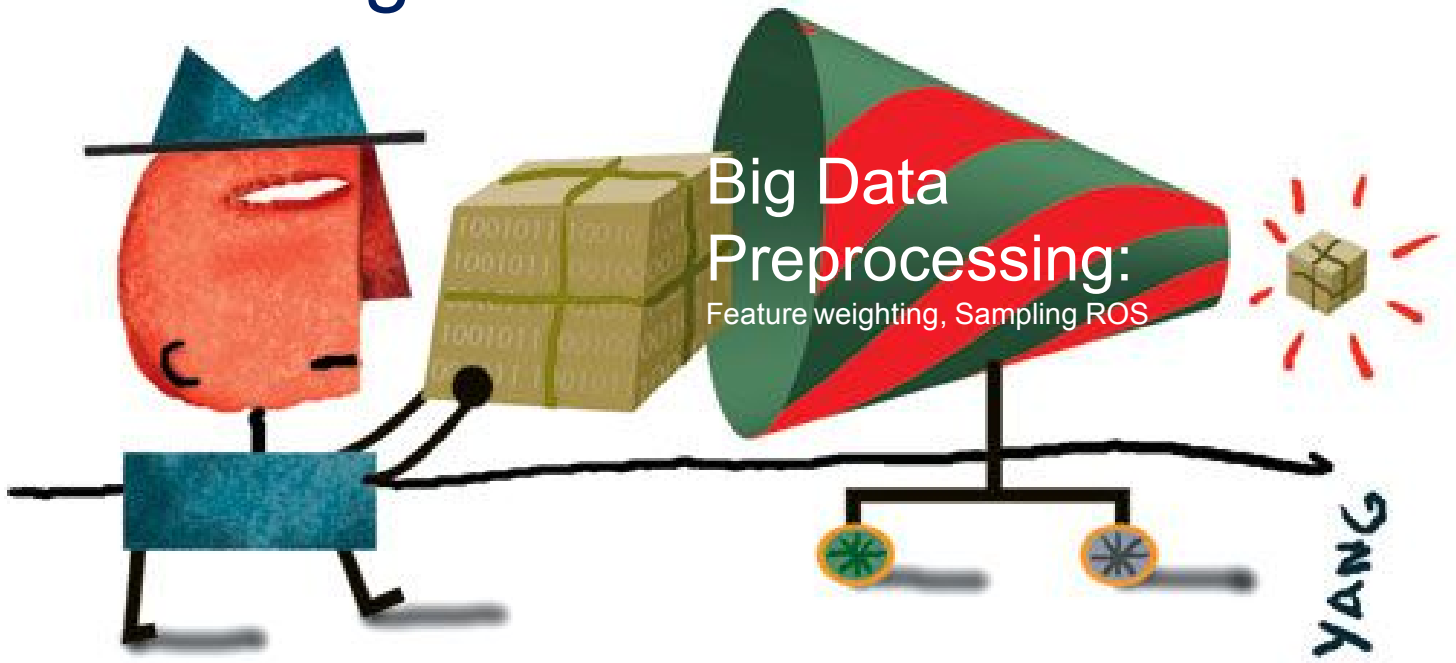
Towards Quality Data





Final Comments

Big Data Preprocessing – Bridge between Big Data and Smart Data



Big Data → Smart Data

Eje. 56.7 GB → FS 8.08 GB, ROS 170% 21.83 GB

ECBDL'14 Big Data Competition

Quality models based on quality data!

Final Comments

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

Big Data Analytics

Big Data: A small subset of algorithms

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

Final Comments

Challenges of Big Data analysis

Jianqing Fan[1,*], Fang Han[2] and Han Liu[1]

National Science Review
1: 293–314, 2014
doi: 10.1093/nsr/nwt032
Advance access publication 6 February 2014

What are the challenges of analyzing Big Data?

Big data are characterized by high dimensionality and large sample size. These two features raise three unique challenges: (i) high dimensionality brings noise accumulation, spurious correlations and incidental ho

Noise accumulation

Spurious correlation

Large sample brings

Duplicity

Redundancy

Final Comments

Big data and analytics: a large challenge offering great opportunities

- A small subset of algorithms.
  It is necessary to redesign new algorithms
- Computing Model
  - Accuracy and Approximation
  - Efficiency requirements for Algorithm

- Big Data Preprocessing
- Noise in data distorts
  Need automatic methods for "cleaning" the data
- Missing values management
- Big Data Reduction

Quality data for quality models in big data analytics!

BIG DATA

Big Data