Université Paris 13 , Sorbonne Paris Cité,
LIPN, UMR 7030 du CNRS
99 Avenue J-B. Clément - 93430 Villetaneuse - France

Scalable machine learning and distributed systems

Mustapha LEBBAH
LIPN - Univ. Paris 13
Machine Learning and Applications (A3)

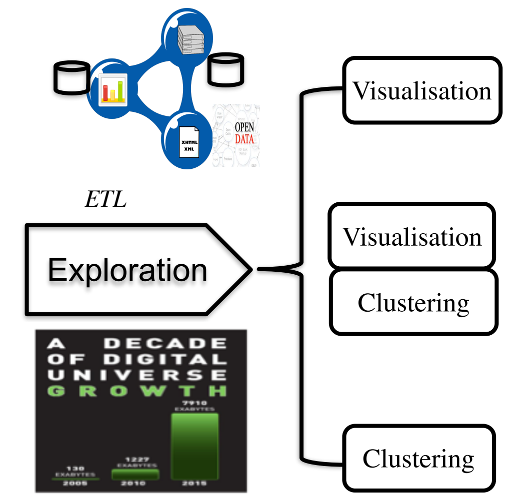
CAEN, SBDS 2017 9 June 2017

H. Azzag, D. Bouthinon, T. Sarazin, M. Ghesmoune, N. Doan, T. Duong, A. Chaibi,

Outline

- Context and Issues
- Clustering and new paradigm
  - K-means
  - Topological model (SOM)
  - Mean-shift
- Conclusion

Context



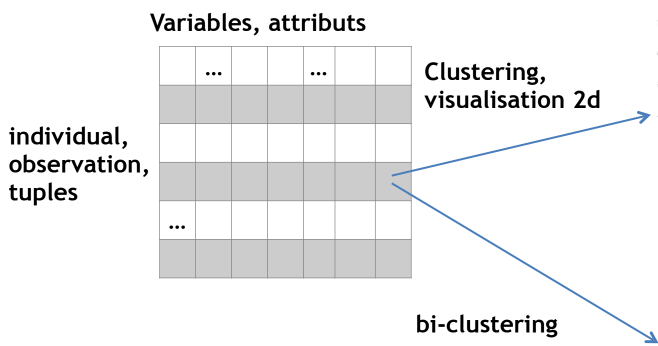
Difficulties

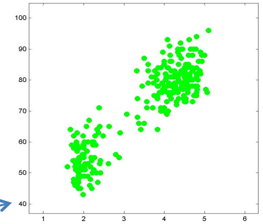
- Existence of a structure
- Similarity measure ?
- Number of groups (Combinatory)
- Validation (no labels)
- Nature of the variables

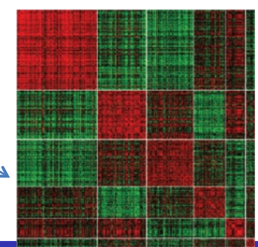
Tutoriel, IEEE IEEE BigData 2014

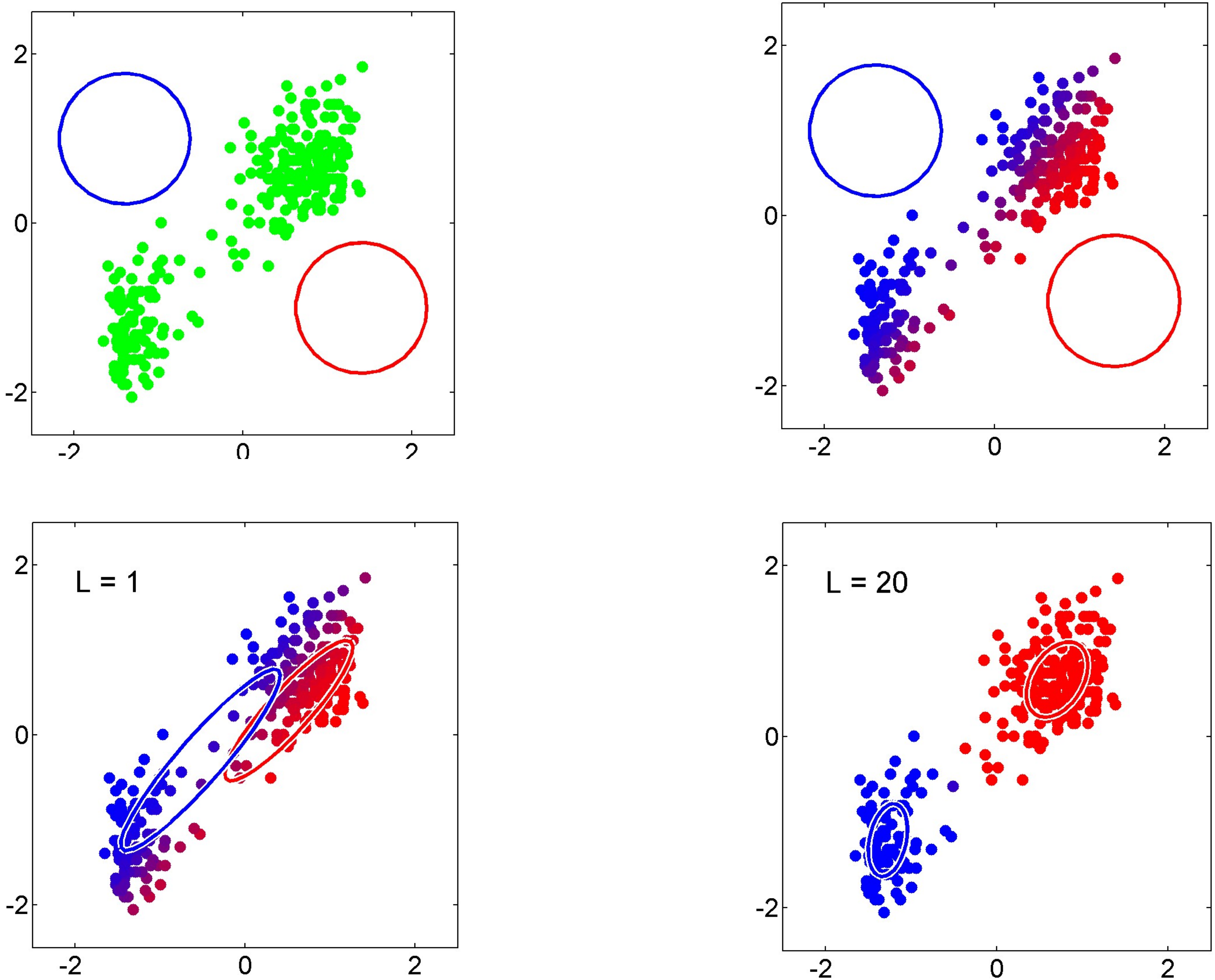
Unsupervised learning

- Hierarchical approaches
- Partitionning approches
- Self-organizing approches
- Probabilistic approches





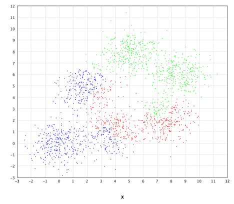



L = 1
L = 20

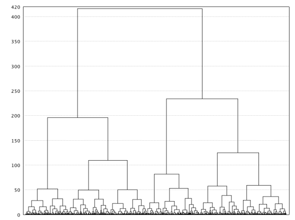
Challenge - scaling up

Parallelism, distributed system, intermediate, collaborative (adaptable)

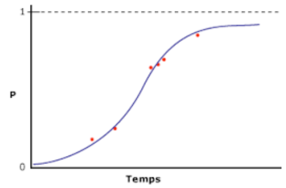
Challenge - scaling



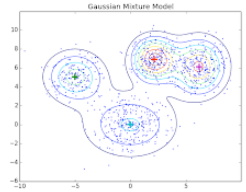
K-means



Hierarchical clustering

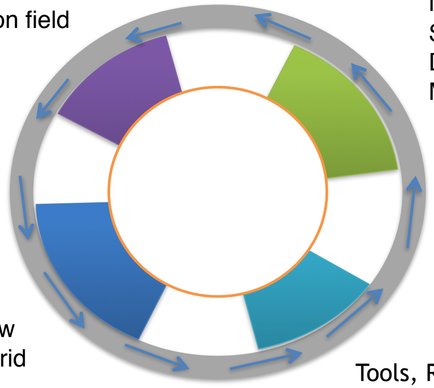


Regression



- Collaborative methods
- Distributed methods
- Ensemble methods

Challenge - Bringing scientific communities together



Challenge - New Systems

- Why do we need new management and analysis systems?

From an analyst point of view
- Fewer iterations to converge
- Data are always available

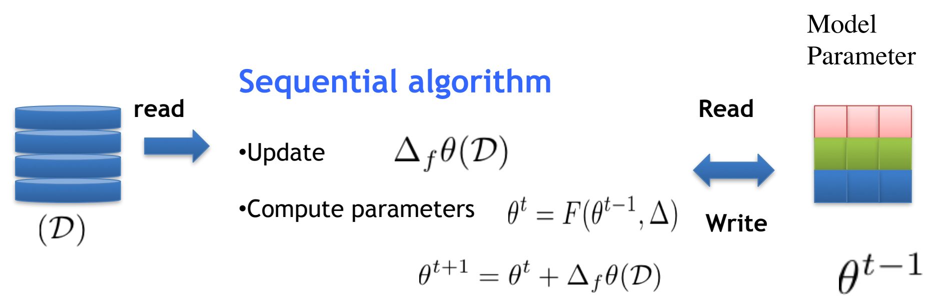
From a system point of view
- More iterations per second
- The algorithm is seen as a black box
- Complex data management (different types of data)

From a researcher's point of view
- Suppose an ideal system

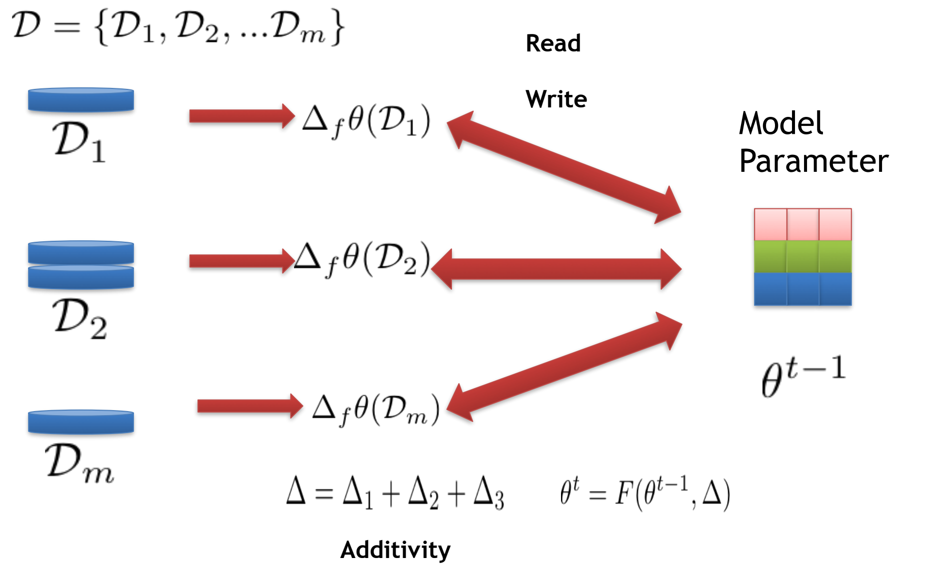
Representation of an iterative algorithm

New model = old model + update (gradient, Data)

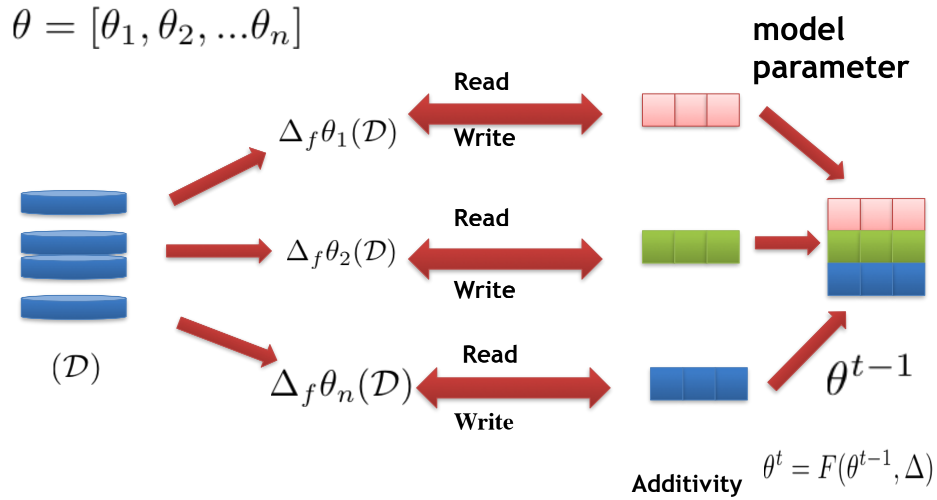
$$\theta^{t+1} = \theta^t + \Delta_f \theta(\mathcal{D})$$



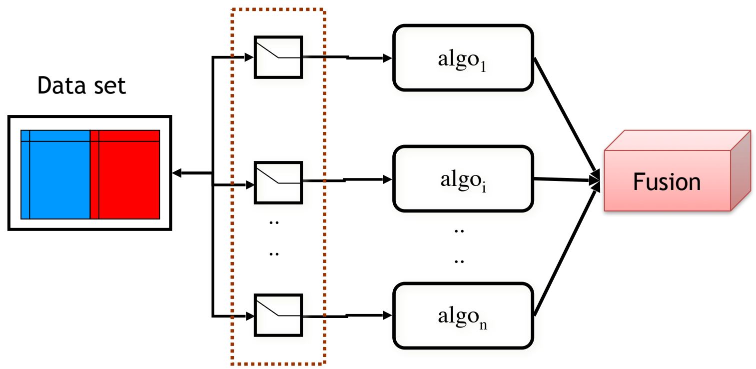
Distributed Data (ensemble clustering)



Distributed Model



Fusion (Clustering Ensembles)

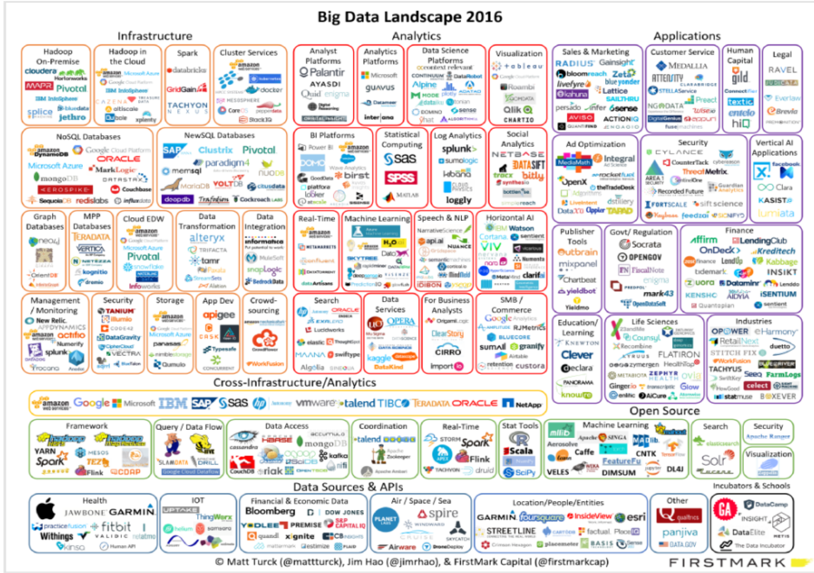


Question

- The distribution of data is often possible when the data is IID (independent and identically distributed (iid))
- The ideal is to work on an adaptable system (distributed, parallel, collaborative)

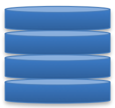
- What about tools / languages to manage data, models, parallelize / distribute, **easily, quickly**?

Outils - Langages



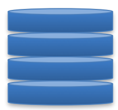
Challenge - the same workspace

In Machine Learning



Workspace
Matrix view

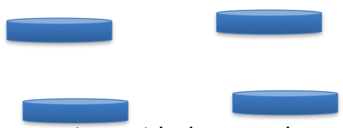
Database



Tables

Relational Database
(Key, values)

In distributed system



Unify the file management system with the analysis system taking into account the environment

We can not have two observations with the same key

The same workspace

- **Manage Workspace as a single database**
- **Resilient Distributed Datasets (RDDs)**
  - Collection of objects stored in memory (or disk) through a cluster
  - RDDs are distributed across workers
  - Parallel processing (map, filter, ...)
  - Rebuild automatically in case of failure

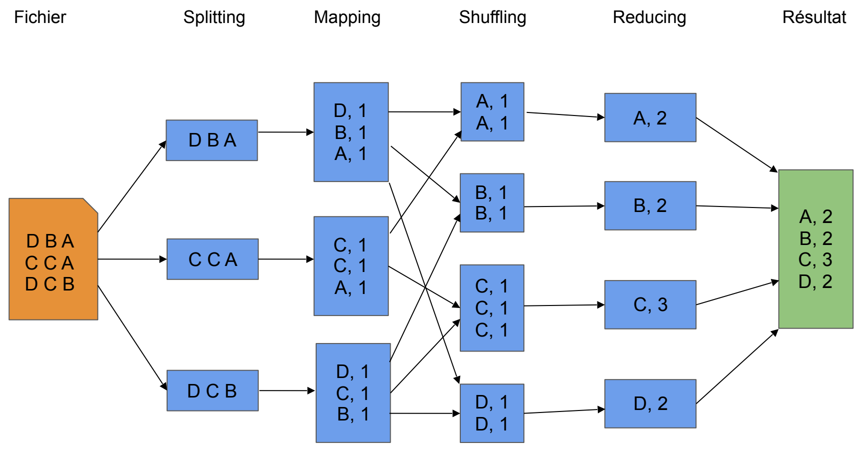
(Matei Zaharia)

Spark/Hadoop = HDFS + MapReduce

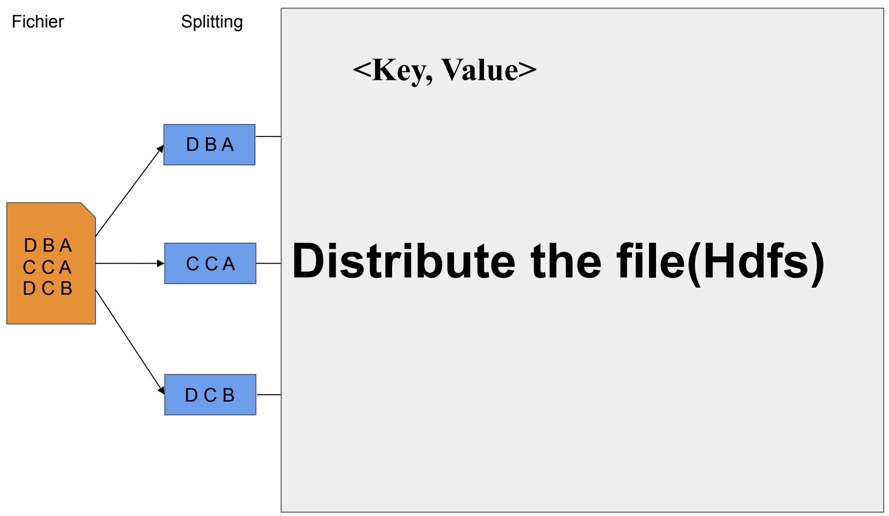
- HDFS : *Hadoop Distributed File System*
- MapReduce :is a programming model for processing a large data sets with a parallel, distributed algorithm on a cluster

- **Architecture Scale-Out:** Adding machines
- **Security**
- **Data locality:** Maximizes execution to the nearest data
- **Scheduling:** Automatic optimization of job scheduling
- **Flexibility: Support for many languages and programming logic (Java, Scala, R,Python)**
- **Resiliency & High Availability:** Unsuccessful jobs and tasks are restarted automatically without loss of the processes already performed

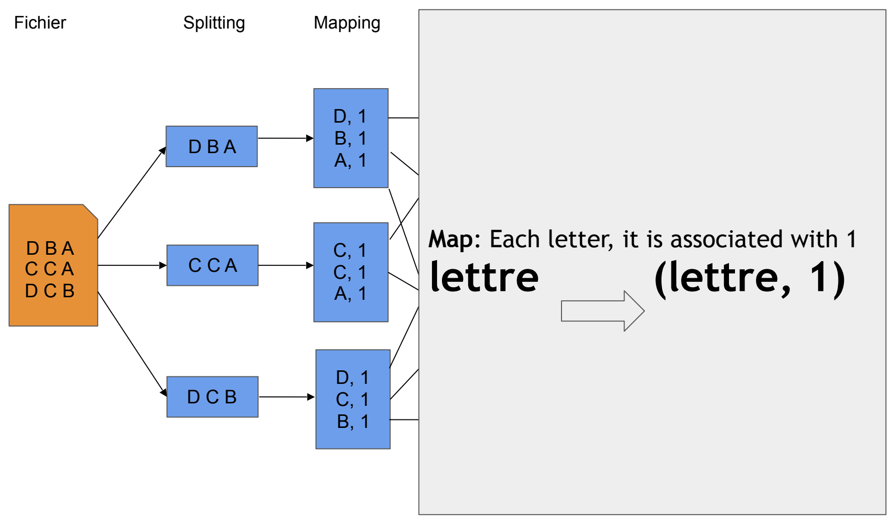
MapReduce (wordcount)



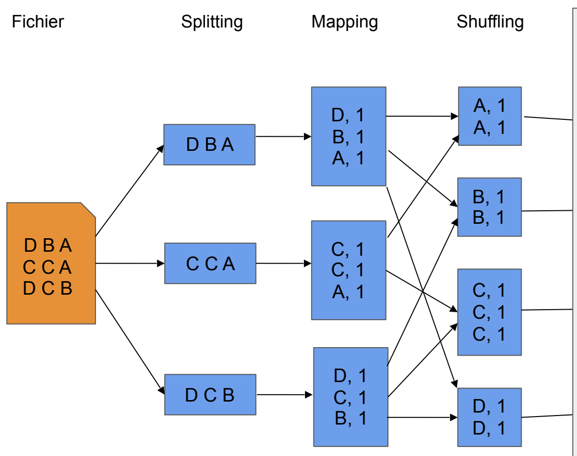
MapReduce



MapReduce



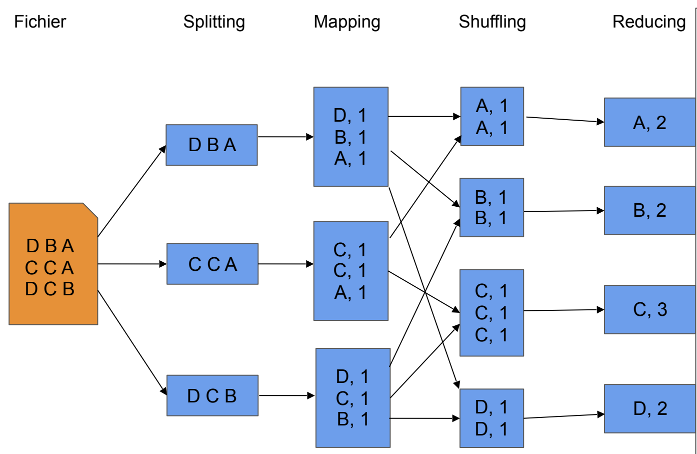
MapReduce



Map: **Map:** Each letter, it is associated with 1
lettre ⇨ **(lettre, 1)**

~group by key

MapReduce

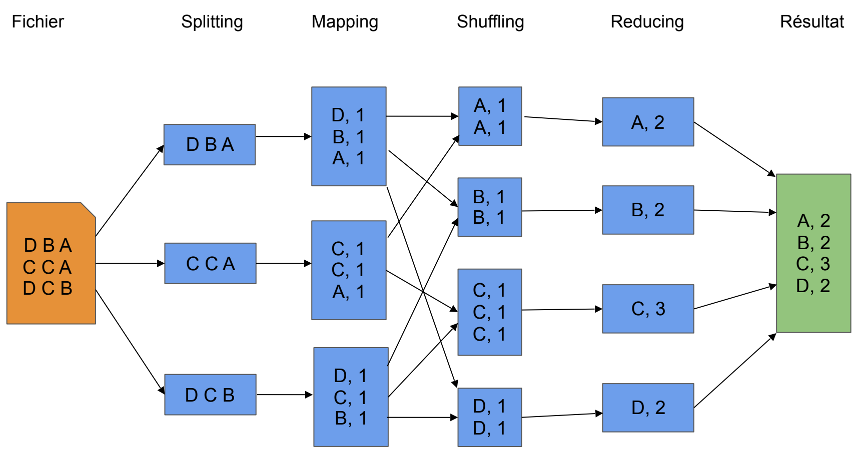


Map: Each letter, it is associated with a 1
lettre —> (lettre, 1)

~group by Key

Addition

MapReduce



WordCount : Hadoop MR vs Spark

hadoop MapReduce

```
public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
} } }
 public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
}}
```

```
file = spark.textFile("hdfs://...")
file.flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)
```

It's very close to the algorithm idea

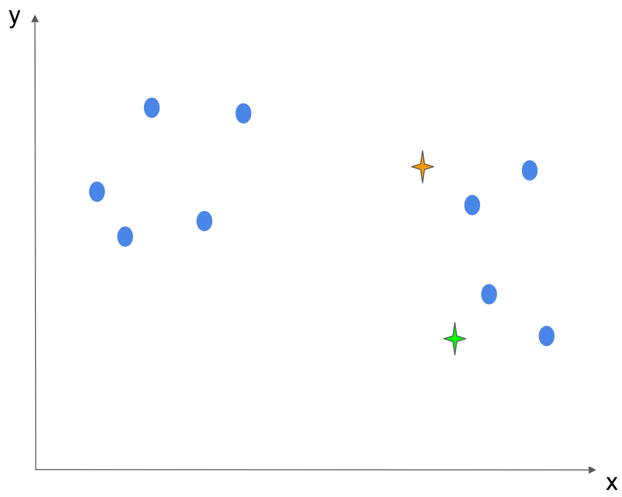
We discover a new language

Idea ➡ Model ➡ design ➡ production

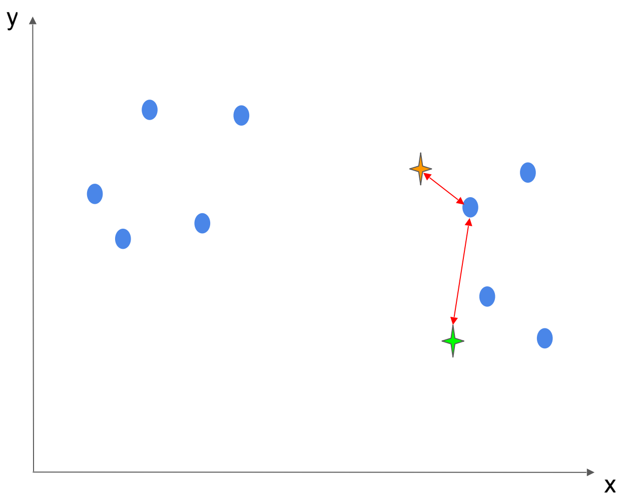
K-means

Understand the K-means-MR from MLIB

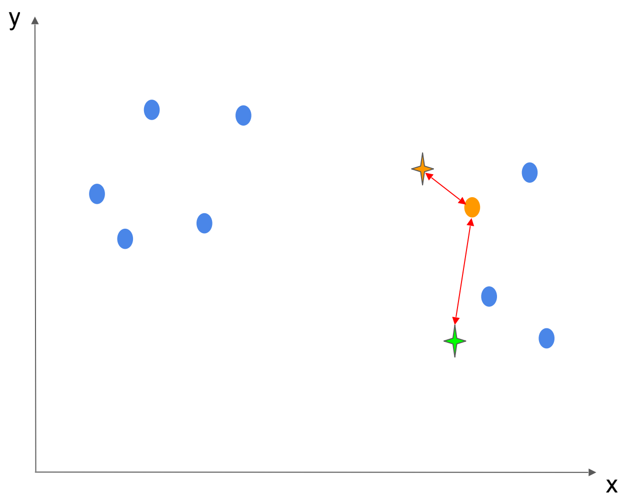
K-means - initialisation



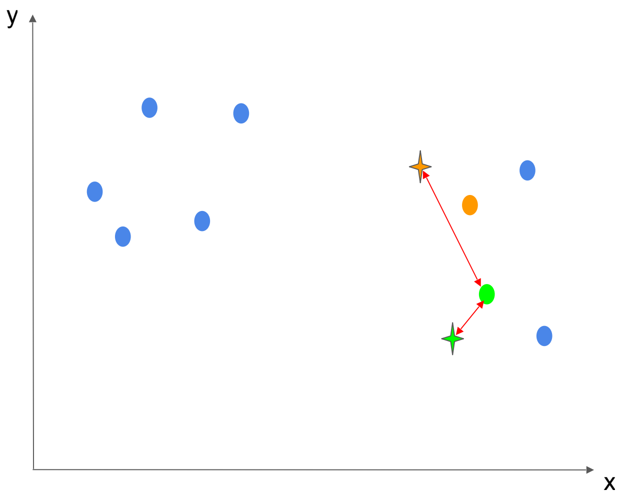
K-means - compute distances with prototypes



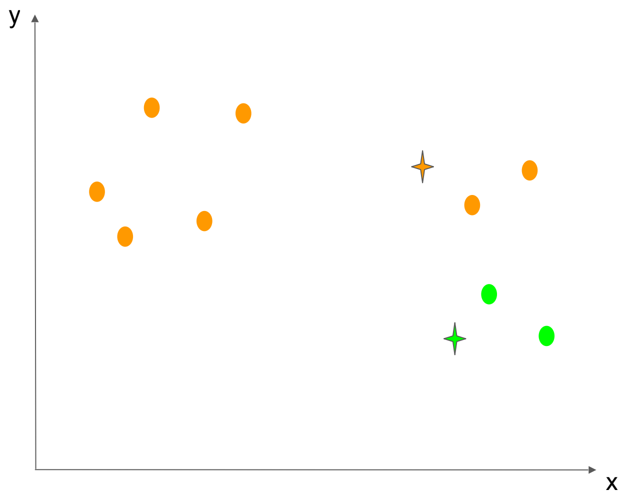
K-means : Assignment



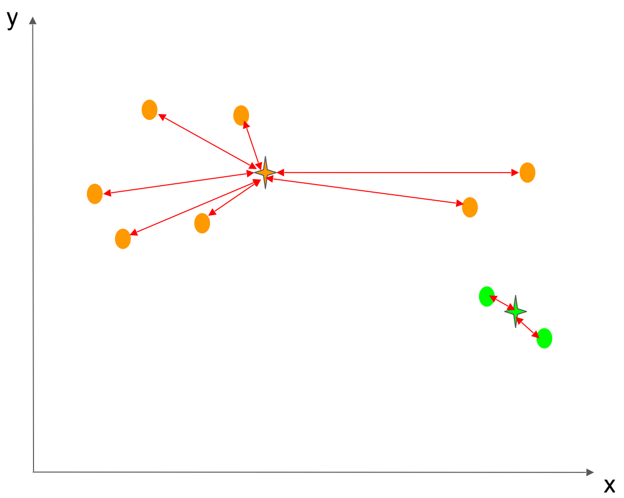
K-means : Assignment



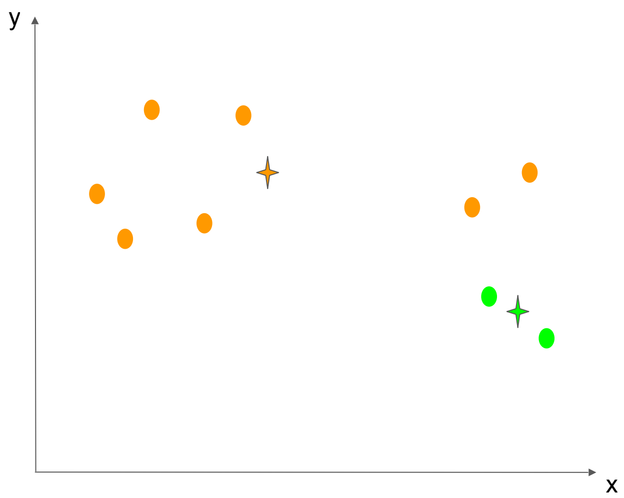
K-means : Assignment



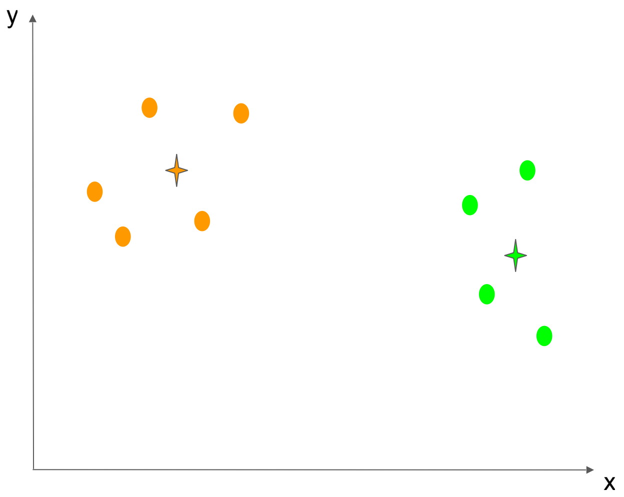
K-means - Reduce (prototype update)



K-means : iteration 1



K-means : iteration 2



K-means

- Minimize the sum of local inertia with respect to ϕ and **W**

$$I(\mathcal{W},\phi) = \sum_{\mathbf{x}_i} \|\mathbf{x}_i - \mathbf{w}_{\phi(\mathbf{x}_i)}\|^2 = \sum_c \sum_{\mathbf{x}_i \in P_c} \|\mathbf{x}_i - \mathbf{w}_c\|^2$$

- The inertia I_c represents the quantisation error obtained if we replace each observation of P_c by its prototype $\mathbf{w}_c$

K-means

- Iterative minimization which fixes the partition (c) alternately and then minimizes the inertia

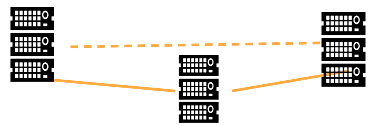
Assignment step:
For a set W of fixed referents, the minimization of I with respect to Φ is obtained by assigning each observation x to the referent $\mathbf{w}_c$ according to the new assignment function Φ

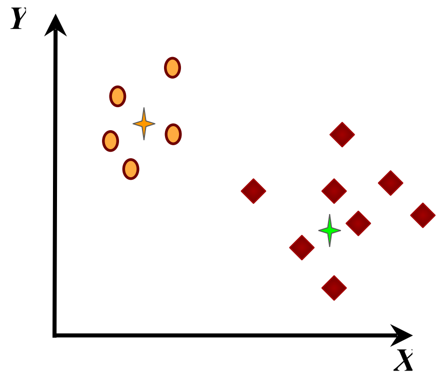
$$\phi(\mathbf{x}) = \arg\min_r \|\mathbf{x} - \mathbf{w}_r\|^2$$

Quantisation step:
The partition Φ is fixed. The function $I(\mathcal{W},\phi)$ is quadratic and convex with respect with W. The minimum is

$$\frac{\partial I}{\partial W} = \left[\frac{\partial I}{\partial \mathbf{w}_1}, \frac{\partial I}{\partial \mathbf{w}_2}, \dots, \frac{\partial I}{\partial \mathbf{w}_p}\right]^p = 0 \qquad \mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in P_c} \mathbf{x}_i}{|P_c|}$$

Decomposition of K-Means



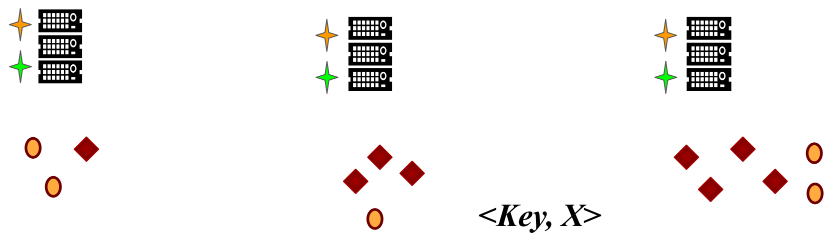


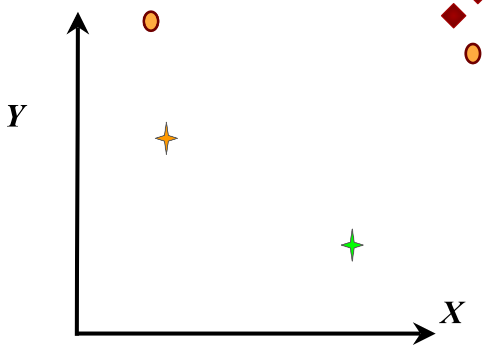
1. Select *k prototypes*
2. **Repeat**
3. -assign each observation to the nearest prototype
4. -Update the prototypes
5. **Until** no change

$C = 1$ et 2 $(k=2)$

$$\mathbf{w}_c = \frac{\sum_1^{n_c} \mathbf{x}_i}{n_c}$$

Decomposition of K-Means (1st solution)





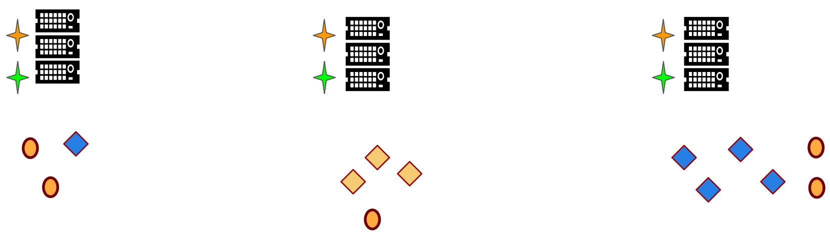
-**Map** : Assign each observation to the nearest prototype

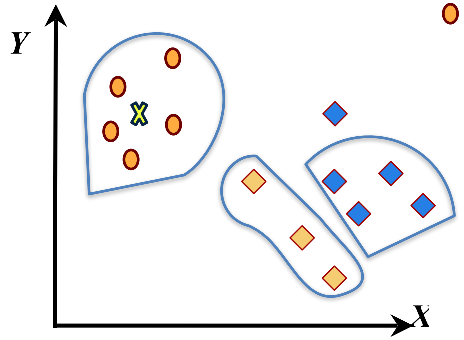
X => (c, X, 1) // c is the cluster ID

-**Reduce** :

$$\mathbf{w}_c = \frac{\sum_1^{n_c} \mathbf{x}_i}{n_c}$$

Decomposition of K-Means (2sd solution)

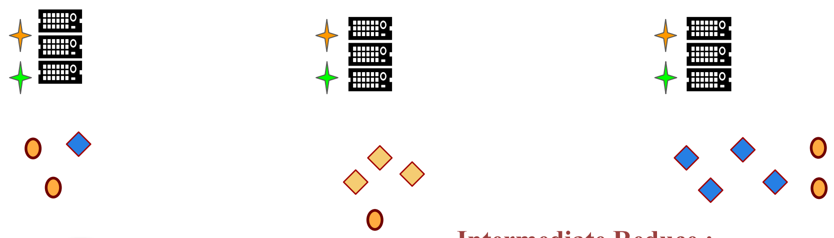


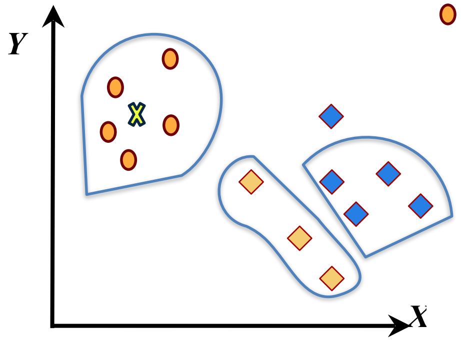


-**Map** : Assign each observation to the nearest prototype

X => (c, X, 1) // c is the cluster ID

Decomposition of K-Means (2sd solution)





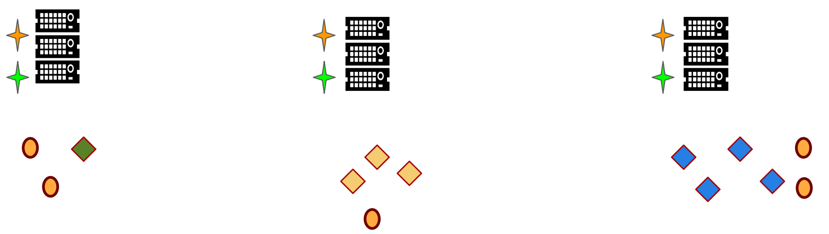
-Intermediate Reduce :

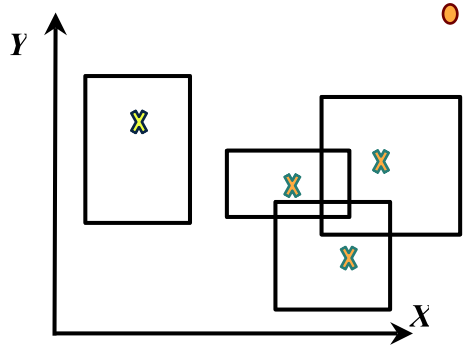
$$\mathbf{w}_1^{M1} = \frac{\sum_1^{n_1^{M1}} \mathbf{x}_i}{n_1^{M1}}$$

$$\mathbf{w}_1^{M2} = \frac{\sum_1^{n_1^{M2}} \mathbf{x}_i}{n_1^{M2}}$$

$$\mathbf{w}_1^{M3} = \frac{\sum_1^{n_1^{M3}} \mathbf{x}_i}{n_1^{M3}}$$

Decomposition of K-Means (2sd solution)



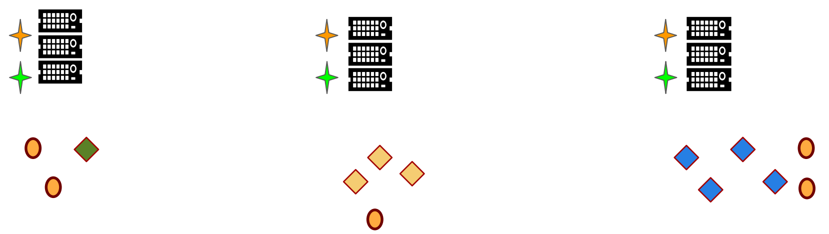


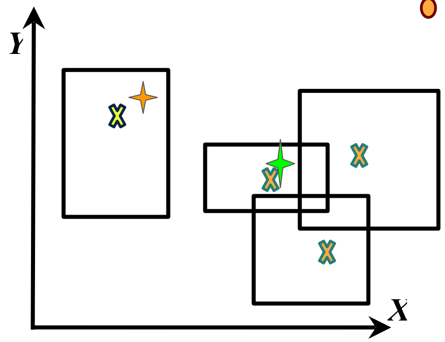
-Global Reduce :

$$\mathbf{w}_1 = \frac{\sum_{i=1}^{3} n_1^{Mi} \times w_1^{Mi}}{n_1^{M3} + n_1^{M2} + n_1^{M3}}$$

C = 1 et 2

Decomposition of K-Means (2sd solution)





-Global Reduce :

$$\mathbf{w}_1 = \frac{\sum_{i=1}^{3} n_1^{Mi} \times w_1^{Mi}}{n_1^{M3} + n_1^{M2} + n_1^{M3}}$$

$C = 1$ et 2

K-Means : The best solution

$$\mathbf{w}_1 = \frac{\sum_{i=1}^{3} n_1^{Mi} \times w_1^{Mi}}{n_1^{M3} + n_1^{M2} + n_1^{M3}}$$

$$\mathbf{w}_c = \frac{\sum_1^{n_c} \mathbf{x}_i}{n_c}$$

<Key, X>
<N°cluster, X, 1>

Do not forget we do not have two tuples with the same key

Initialisation : select K prototypes
Repeat
-**Map** : Assign each observation X to the nearest prototype
X => (c, X, 1) // c is the cluster ID

-**Reduce**: Addition of X belonging to the same cluster c and count the data
(c, X1,1), (c, X2, 1) => (c, X1+X2, 1+1)

// at the last reduce we obtain (c, $\sum_1^{n_c} \mathbf{x}_i$ *nc)*

-**Collect**: Update the prototype $\mathbf{w}_c = \frac{\sum_1^{n_c} \mathbf{x}_i}{n_c}$

Until no change

K-means, scala-Spark

```
for (i <- 1 until 10) {  // 10 is the number of epoch

// Assignment step

 val closest = data.map(x =>
   (closestCentroid(x, centroids), (x, 1))
 )

// Quantisation step
// 1- addition of X and count the number of observation assigned
to each cluster

 val pointStats=closest.reduceByKey{
   case ((x1, nb1), (x2, nb2)) => (x1 + x2, nb1 + nb2)
 }
// 1- Compute the gravity centers (prototypes)
 pointStats.foreach{case(id, value) =>
 centroids(id) = value._1 / value._2
 }
}
```

K-means : MLlib - Spark

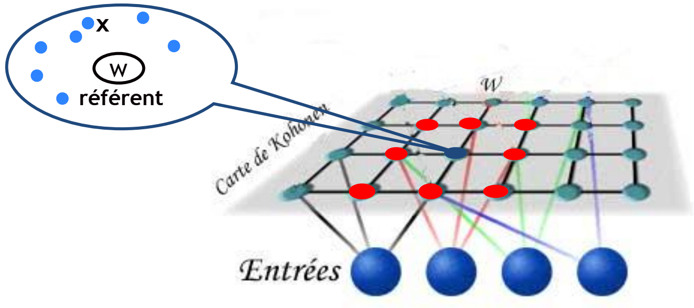
```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/mllib/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split('
').map(_.toDouble))).cache()

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)

// Save and load model
clusters.save(sc, "myModelPath")
val sameModel = KMeansModel.load(sc, "myModelPath")
```

First experience with MapReduce

Batch version

SOM: Self-organizing Map

Topological map :self-organizing Map



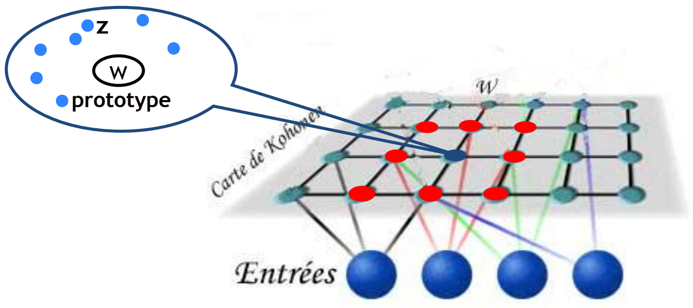
$K(\delta(c_1,c_2))$

Fonction noyau

$\delta(c_1,c_2)$ *The short path on the graph*

- It is a discrete space (C) of small dimension for visualization purposes (1-D, 2-D).
- *C* set of cells (nodes, neurons) connected by a non-oriented graph structure provided with a discrete distance δ on C and a neighborhood structure

SOM



$K(\delta(c_1,c_2))$

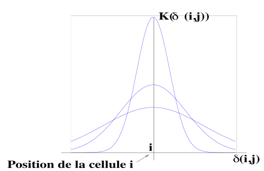
Kernel function

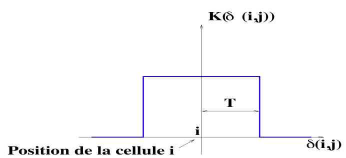
$$J^T_{som}(\mathcal{W},\phi) = \sum_{\mathbf{x}_i \in \mathcal{A}} \sum_{c \in C} K^T(\delta(c,\phi(\mathbf{x}_i)))||\mathbf{x}_i - \mathbf{w}_c||^2$$

Topological Map

Cost Function

$$J^T_{som}(\mathcal{W},\phi) = \sum_{\mathbf{x}_i \in \mathcal{A}} \sum_{c \in C} K^T(\delta(c,\phi(\mathbf{x}_i)))||\mathbf{x}_i - \mathbf{w}_c||^2$$

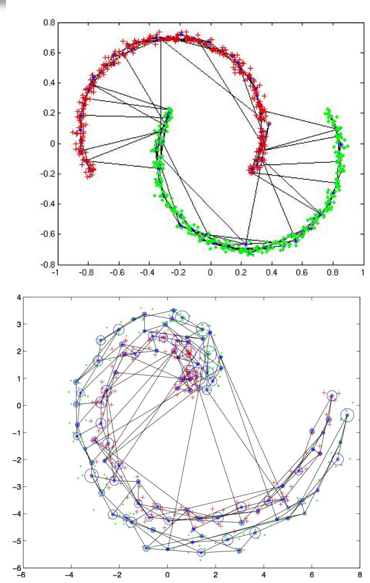




$$K^T(\delta(c,\phi(\mathbf{x}_i))) = \exp(-0.5\frac{d}{T})$$

$V_c^T = \{r \in C / K^T(\delta(c,r)) > \alpha\}$. The value of T determines the size of the neighborhood

Illustrations / visualisation



Batch version

Iterative minimization $J^T_{som}(\mathcal{W},\phi)$ For a fixed parameter T:

Assignment step

The set $\mathcal{W}$ of the referents is fixed, the minimization is obtained by assigning each observation x to the referent wc according to the new assignment function Φ^T

$$\phi^T(\mathbf{x}) = \arg\min_{r \in C} \left(\sum_{c \in C} K^T(\delta(c,r)) \|\mathbf{x}_i - \mathbf{w}_c\|^2 \right)$$

Minimization step:

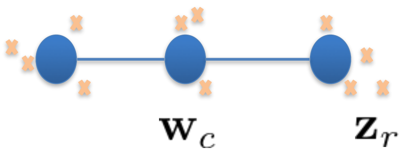
The partition is fixed Φ^T. $J^T_{som}(\mathcal{W},\phi)$ **is minimized with respect to all referents $\mathcal{W}$.**

$$\mathbf{w}_c^T = \frac{\sum_{r \in C} K(\delta(c,r))\mathbf{Z}_r}{\sum_{r \in C} K(\delta(c,r)) n_r}$$

$\mathbf{Z}_r$ represents the sum of all observations assigned to the cell r

n_r number of observation

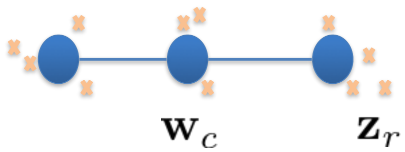
SOM, MapReduce



$$\phi(\mathbf{x}_i)$$

$$\mathbf{w}_c = \frac{\sum_{r \in C} \mathcal{K}(c,r) \sum_{\mathbf{x}_i \in C_r} \mathbf{x}_i}{\sum_{r \in C} \mathcal{K}(c,r) m_r}$$

SOM, MapReduce



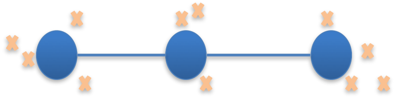
$$\phi(\mathbf{x}_i)$$

$$\mathbf{w}_c = \frac{\sum_{r \in C} \mathcal{K}(c,r) \sum_{\mathbf{x}_i \in C_r} \mathbf{x}_i}{\sum_{r \in C} \mathcal{K}(c,r) m_r}$$

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c,\phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c,\phi(\mathbf{x}_i))}$$

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



Map Step : Numerator

$\mathbf{x}_1 \rightarrow \phi(\mathbf{x}_1)$

$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$

$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$

$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$

....

$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$

....

$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$

$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$

$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$

$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$

....

$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$

....

$\mathbf{x}_N \rightarrow \phi(\mathbf{x}_N)$

$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$

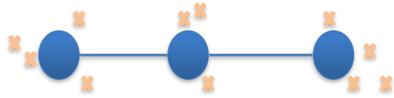
$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$

$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$

$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



Map Step : Numerator

$\mathbf{x}_1 \rightarrow \phi(\mathbf{x}_1)$

$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$

$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$

$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$

....

$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$

....

$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$

$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$

$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$

$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$

....

$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$

....

$\mathbf{x}_N \rightarrow \phi(\mathbf{x}_N)$

$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$

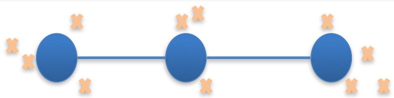
$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$

$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$

$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



Map Step : Numerator

$\mathbf{x}_1 \rightarrow \phi(\mathbf{x}_1)$ Key

$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$; 1

$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$; 2

$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$; 3

....

$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$; C

....

$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$

$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$; 1

$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$; 2

$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$; 3

....

$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$; C

....

$\mathbf{x}_N \rightarrow \phi(\mathbf{x}_N)$

$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$; 1

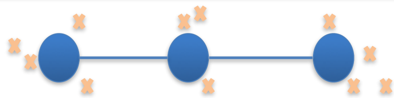
$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$; 2

$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$; 3

$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$; C

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



Map Step : Numerator

$\mathbf{x}_1 \rightarrow \phi(\mathbf{x}_1)$ Key

$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$; 1

$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$; 2

$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$; 3

....

$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$; C

....

$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$

$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$; 1

$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$; 2

$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$; 3

....

$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$; C

....

$\mathbf{x}_N \rightarrow \phi(\mathbf{x}_N)$

$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$; 1

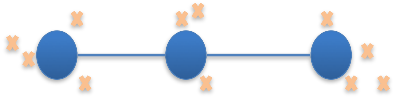
$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$; 2

$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$; 3

$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$; C

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



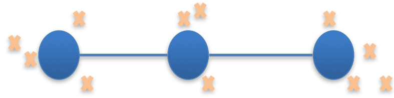
$\mathbf{w}_c$

Map Step : Numerator

$\mathbf{x}_1$	Key	$\mathbf{x}_i$		$\mathbf{x}_N$	
$\phi(\mathbf{x}_1)$		$\phi(\mathbf{x}_i)$		$\phi(\mathbf{x}_N)$	
$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$	; 1	$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$	; 1	$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$	; 1
$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$	; 2	$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$	; 2	$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$	; 2
	....		....		
$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$	; 3	$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$	; 3	$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$	; 3
....		....			
$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$	; C	$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$	; C	$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$	; C

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



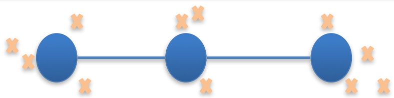
$\mathbf{w}_c$

Map Step : Denominator

$\mathbf{x}_1$	Key	$\mathbf{x}_i$		$\mathbf{x}_N$	
$\phi(\mathbf{x}_1)$		$\phi(\mathbf{x}_i)$		$\phi(\mathbf{x}_N)$	
$\mathcal{K}(1, \phi(\mathbf{x}_1))$	; 1	$\mathcal{K}(1, \phi((\mathbf{x}_i)))$	; 1	$\mathcal{K}(1, \phi(\mathbf{x}_N))$	; 1
$\mathcal{K}(2, \phi(\mathbf{x}_1))$	; 2	$\mathcal{K}(2, \phi((\mathbf{x}_i)))$	; 2	$\mathcal{K}(2, \phi(\mathbf{x}_N))$	; 2
	....		....		
$\mathcal{K}(3, \phi(\mathbf{x}_1))$	; 3	$\mathcal{K}(3, \phi((\mathbf{x}_i)))$	; 3	$\mathcal{K}(3, \phi(\mathbf{x}_N))$	; 3
...		....			
$\mathcal{K}(C, \phi(\mathbf{x}_1))$	; C	$\mathcal{K}(C, \phi((\mathbf{x}_i)))$	; C	$\mathcal{K}(C, \phi(\mathbf{x}_N))$	; C

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



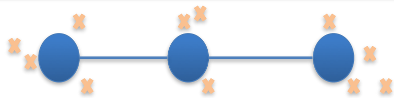
$\mathbf{w}_c$

Reduce Step : Numerator

$\mathbf{x}_1$	Key		$\mathbf{x}_i$			$\mathbf{x}_N$	
$\phi(\mathbf{x}_1)$			$\phi(\mathbf{x}_i)$			$\phi(\mathbf{x}_N)$	
$\mathbf{x}_1 \times \mathcal{K}(1, \phi(\mathbf{x}_1))$	; 1	+	$\mathbf{x}_i \times \mathcal{K}(1, \phi((\mathbf{x}_i)))$	; 1	+	$\mathbf{x}_N \times \mathcal{K}(1, \phi(\mathbf{x}_N))$	; 1
$\mathbf{x}_1 \times \mathcal{K}(2, \phi(\mathbf{x}_1))$	; 2	+	$\mathbf{x}_i \times \mathcal{K}(2, \phi((\mathbf{x}_i)))$	; 2	+	$\mathbf{x}_N \times \mathcal{K}(2, \phi(\mathbf{x}_N))$	; 2
		....			....		
$\mathbf{x}_1 \times \mathcal{K}(3, \phi(\mathbf{x}_1))$	; 3	+	$\mathbf{x}_i \times \mathcal{K}(3, \phi((\mathbf{x}_i)))$	; 3		$\mathbf{x}_N \times \mathcal{K}(3, \phi(\mathbf{x}_N))$	; 3
....			....				
$\mathbf{x}_1 \times \mathcal{K}(C, \phi(\mathbf{x}_1))$	; C	+	$\mathbf{x}_i \times \mathcal{K}(C, \phi((\mathbf{x}_i)))$	; C	+	$\mathbf{x}_N \times \mathcal{K}(C, \phi(\mathbf{x}_N))$	; C

SOM, MapReduce

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



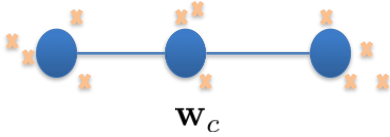
$\mathbf{w}_c$

Reduce Step : Denominator

$\mathbf{x}_1$	Key		$\mathbf{x}_i$			$\mathbf{x}_N$	
$\phi(\mathbf{x}_1)$			$\phi(\mathbf{x}_i)$			$\phi(\mathbf{x}_N)$	
$\mathcal{K}(1, \phi(\mathbf{x}_1))$	; 1	+	$\mathcal{K}(1, \phi((\mathbf{x}_i)))$	; 1	+	$\mathcal{K}(1, \phi(\mathbf{x}_N))$	; 1
$\mathcal{K}(2, \phi(\mathbf{x}_1))$	; 2	+	$\mathcal{K}(2, \phi((\mathbf{x}_i)))$	; 2	+	$\mathcal{K}(2, \phi(\mathbf{x}_N))$	; 2
		....			....		
$\mathcal{K}(3, \phi(\mathbf{x}_1))$	; 3	+	$\mathcal{K}(3, \phi((\mathbf{x}_i)))$	; 3		$\mathcal{K}(3, \phi(\mathbf{x}_N))$	; 3
...			....				
$\mathcal{K}(C, \phi(\mathbf{x}_1))$	; C	+	$\mathcal{K}(C, \phi((\mathbf{x}_i)))$	; C	+	$\mathcal{K}(C, \phi(\mathbf{x}_N))$	; C

SOM, MapReduce

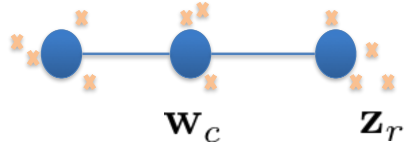
$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$



Reduce Step :

Numerator	Denominator	
$\sum_{\mathbf{x}_i \in \mathcal{A}} K(1, \phi(\mathbf{x}_i))\mathbf{x}_i$	$\sum_{\mathbf{x}_i \in \mathcal{A}} K(1, \phi(\mathbf{x}_i))$	; 1
$\sum_{\mathbf{x}_i \in \mathcal{A}} K(2, \phi(\mathbf{x}_i))\mathbf{x}_i$	$\sum_{\mathbf{x}_i \in \mathcal{A}} K(2, \phi(\mathbf{x}_i))$	; 2
$\sum_{\mathbf{x}_i \in \mathcal{A}} K(3, \phi(\mathbf{x}_i))\mathbf{x}_i$	$\sum_{\mathbf{x}_i \in \mathcal{A}} K(3, \phi(\mathbf{x}_i))$	; 3
$\sum_{\mathbf{x}_i \in \mathcal{A}} K(C, \phi(\mathbf{x}_i))\mathbf{x}_i$	$\sum_{\mathbf{x}_i \in \mathcal{A}} K(C, \phi(\mathbf{x}_i))$	; C

SOM, MapReduce



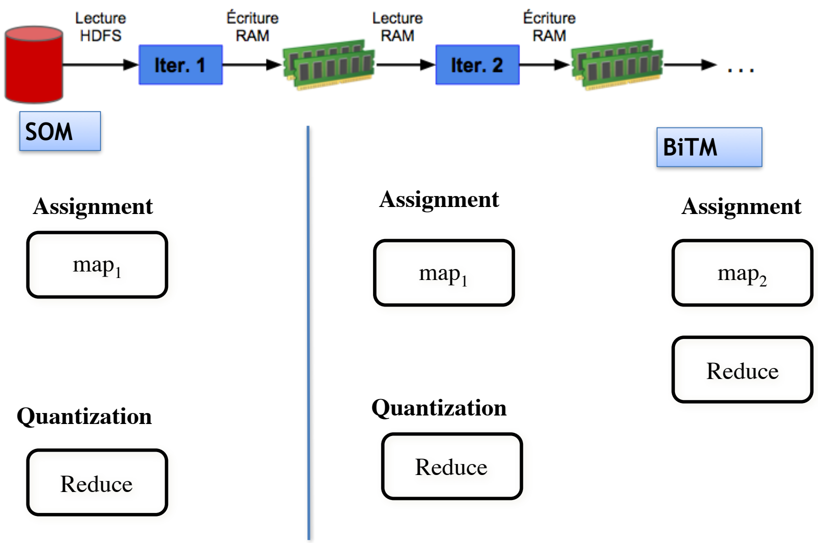
$$\mathbf{w}_c = \frac{\sum_{r \in C} \mathcal{K}(c, r) \sum_{\mathbf{x}_i \in C_r} \mathbf{x}_i}{\sum_{r \in C} \mathcal{K}(c, r) m_r}$$

$$\mathbf{w}_c = \frac{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{A}} \mathcal{K}(c, \phi(\mathbf{x}_i))}$$

Numerator

denominator

MapReduce / Spark

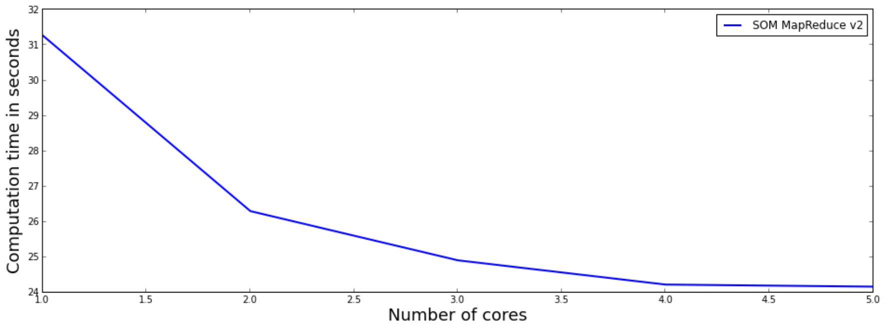


https://github.com/TugdualSarazin/spark-clustering

SOM MR

```
prototypes = data.map{x =>
  val closestc = closestPrototype(prototypes, x)

  prototypes.map{ c =>
                    val t = K(c, closestc)
                    (c,(x*t, t))
                }
}.reduce{ (a, b) =>
  a.zip(b).map{
            case ((aNum, aDenom),(bNum, bDenom)) =>
            (aNum+bNum, aDenom+bDenom)
              }
}.map(w => w._1 / w._2)
```

Topological Map for Mixed Data

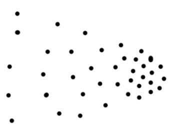


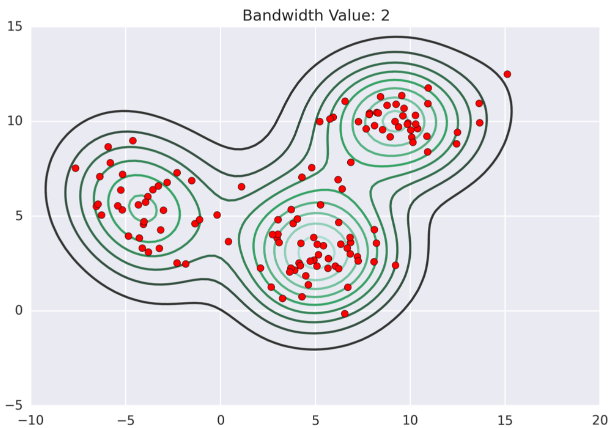
- 100 millions of observations
- SOM - Map : 10 x 10

Second experience

Mean-shift

Illustration





Modal Clustering

Clusters = basins of attractions of gradient ascent paths of probability density function (pdf) (Li et al, 2007, JMLR)

Mean shift clustering

- Assume density f and gradient field $\mathrm{D}f$ are known
- Initial condition y_0
- Update $y_{j+1} = y_j + \mathbf{A}\,\mathrm{D}f(y_j)/f(y_j), \quad j = 1, 2, \ldots$
- Sequence $\{y_0, y_1, \ldots, \}$ converges to local mode in f (for any $\mathbf{A}$)
- Assume density f and gradient field $\mathrm{D}f$ are unknown
- If $\hat{f}, \mathrm{D}\hat{f}$ are k-nn estimators then update simplifies to

$$y_{j+1} = \frac{1}{k} \sum_{X_i \in k-\mathrm{nn}(y_j)} X_i, \quad j = 1, 2, \ldots$$

(Fukunaga & Hostetler, 1975)

- Update y_{j+1} = mean of k nearest neighbours of current iterate y_j

Modal Clustering

Clusters = basins of attractions of gradient ascent paths of probability density function (pdf) (Li et al, 2007, JMLR)

Mean shift clustering

- Assume density f and gradient field $\mathrm{D}f$ are known
- Initial condition y_0
- Update $y_{j+1} = y_j + \mathbf{A}\,\mathrm{D}f(y_j)/f(y_j), \quad j = 1, 2, \ldots$
- Sequence $\{y_0, y_1, \ldots, \}$ converges to local mode in f (for any $\mathbf{A}$)
- Assume density f and gradient field $\mathrm{D}f$ are unknown
- If $\hat{f}, \mathrm{D}\hat{f}$ are k-nn estimators then update simplifies to

$$y_{j+1} = \frac{1}{k} \sum_{X_i \in k-\mathrm{nn}(y_j)} X_i, \quad j = 1, 2, \ldots$$

(Fukunaga & Hostetler, 1975)

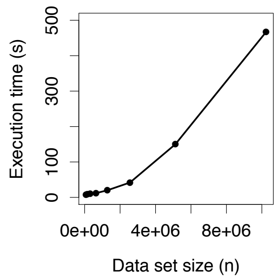
- Update y_{j+1} = mean of k nearest neighbours of current iterate y_j

Dissimilarity matrix

- Key to NNMS is $n \times n$ dissimilarity matrix

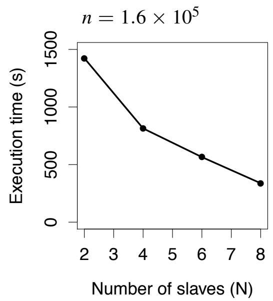
$$\begin{bmatrix} d(X_1, X_1) & d(X_1, X_2) & \ldots & d(X_1, X_n) \\ d(X_2, X_1) & d(X_2, X_2) & \ldots & d(X_2, X_n) \\ \vdots & & & \vdots \\ d(X_n, X_1) & d(X_n, X_2) & \ldots & d(X_n, X_n) \end{bmatrix}$$

- Dissimilarity matrix is $O(n^2)$
- Within i-th column, sort distances
- Data point $X_j, j \neq i$ with k-th smallest distances are k-nn to X_i
- Quick sort = $O(n \log n)$ for 1 column so k-nn is $O(n^2 \log n)$



Distributed way

- Each column in dissimilarity matrix can be computed independently
- Parallelise columns amongst N slave processes
- Time complexity reduced to $O((n^2 \log n)/N)$ but still insufficient for large n

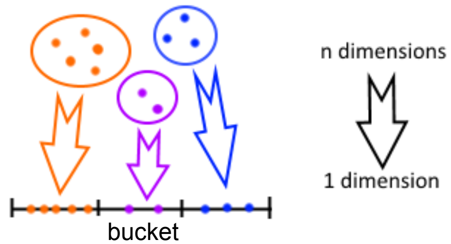


LSH : Approximate the k-NN

- Use approximate NN to reduce complexity from exact NN
- Locality Sensitive Hashing (LSH) based on random scalar projections of multivariate data

$$L(y) = Z^T y + U$$

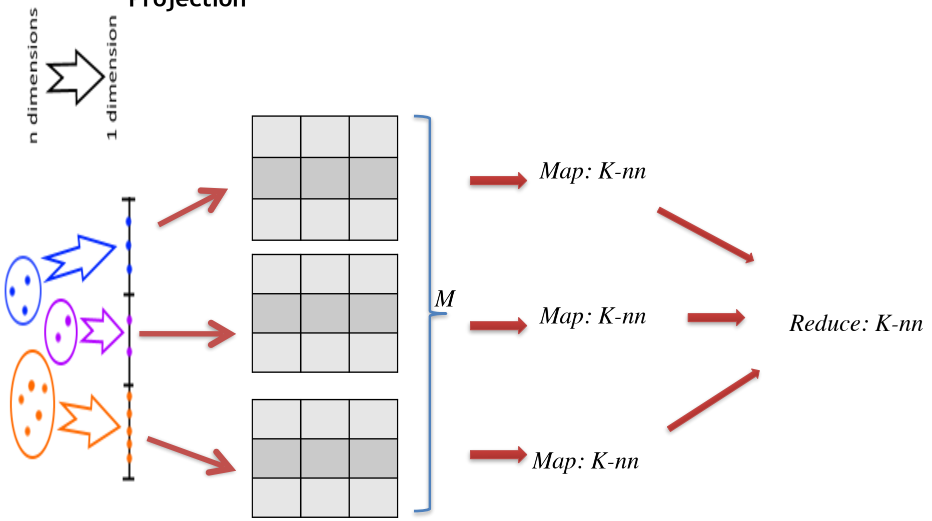
where Z is standard multivariate Gaussian, U is standard uniform
(Indyk & Motwani, 1998, *STOC '98*)



Illustration

$O(n^2 log\ n)$ → $O((n^2/M)log(n/M))$

Projection



- Spark Scala ecosystem
- Execution times for DNNMS-LSH-M

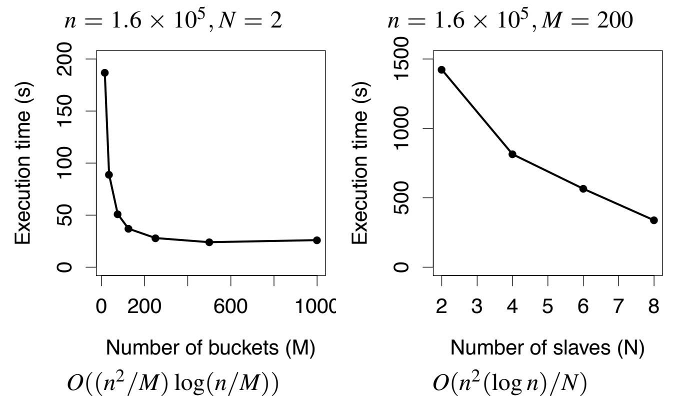


Image Segmentation

Nearest neighbour mean-shift clustering (NNMS) $O(n^2 log\ n)$ → $O((n^2/M)log(n/M))$

8h in R **5 min (4 cores)**

1min40 (24 cores)

Image Segmentation

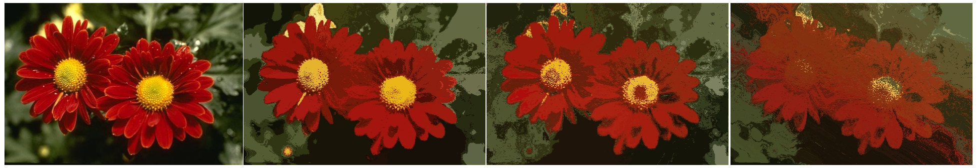


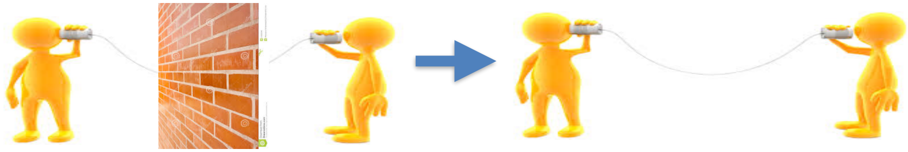
Image originale 481x321	Image : 43 clusters 200 buckets 5 min (4 cores) 1 min (20 cores)	Image : 40 clusters 500 buckets 8 min (4 cores)	Image : 34 clusters 1000 buckets 10 min (4 cores)

Mean Shift without LSH

A large number of buckets can impair quality.
It must be adapted according to the size of the dataset.

Conclusion : from a practical point of view

- It's time to use Scala in research and teaching. You will focus on your research problem



With Scala and associated package, researchers, data scientists and developers can work in the same environment to build machine learning applications quickly!

- The algorithms written with Scala are available on Spark-notebook

https://github.com/Spark-clustering-notebook/coliseum/wiki

Scala kensu.io

Spark MLlib

Conclusion : from a research point of view

- The MapReduce paradigm is not the only solution! (but it helps)
- It is very important to model within the ecosystem : a new algorithm
- Your algorithm would be valid on a small scale and on a large scale