INFO-H-515:
PART BIG DATA ANALYTICS

Map-reduce analytics

Gianluca Bontempi

Machine Learning Group
Boulevard de Triomphe - CP 212
http://mlg.ulb.ac.be

COARSE-GRAINED PARALLELISM

We will discuss how to distribute machine learning computing in the following setting

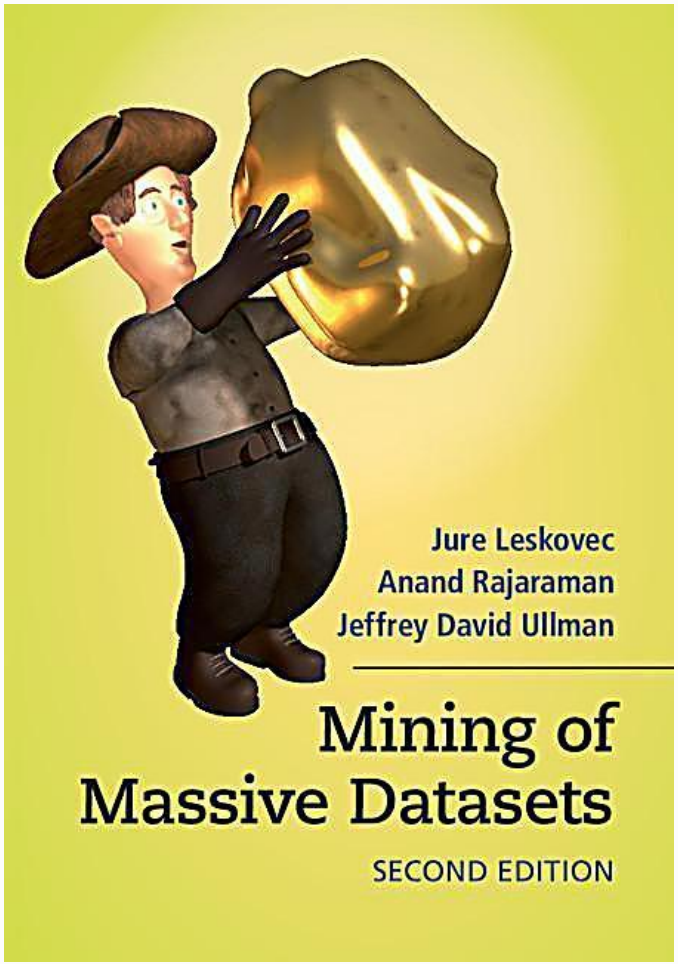
- Training dataset that does not fit in the main memory of a single machine
- Computing cluster infrastructure, i.e. a large collection of commodity hardware connected by Ethernet or inexpensive switches
- Distributed file system (e.g. HDFS), providing replication and redundancy
- Map Reduce programming model partitioning the computation in tasks (that can be restarted without affecting the others) allowing then scalability and robustness.
- Map = Scatter pieces of a problem across hosts
- Reduce= Gather/Aggregate partial solution into a final result
- Increasing performance is not necessarily the main goal. Fault tolerance and scalability (avoid bottlenecks) are more important ones!

FROM THE CREATOR OF MAP-REDUCE

Part of the reason we didn't develop MapReduce earlier was probably because when we were operating at a smaller scale, then our computations were using fewer machines, and therefore robustness wasn't quite such a big deal: it was fine to periodically checkpoint some computations and just restart the whole computation from a checkpoint if a machine died. Once you reach a certain scale, though, that becomes fairly untenable since you'd always be restarting things and never make any forward progress. (Jeff Dean)

DISCLAIMER

Much of the following material is taken from the book [7]



MAP-REDUCE EXAMPLES

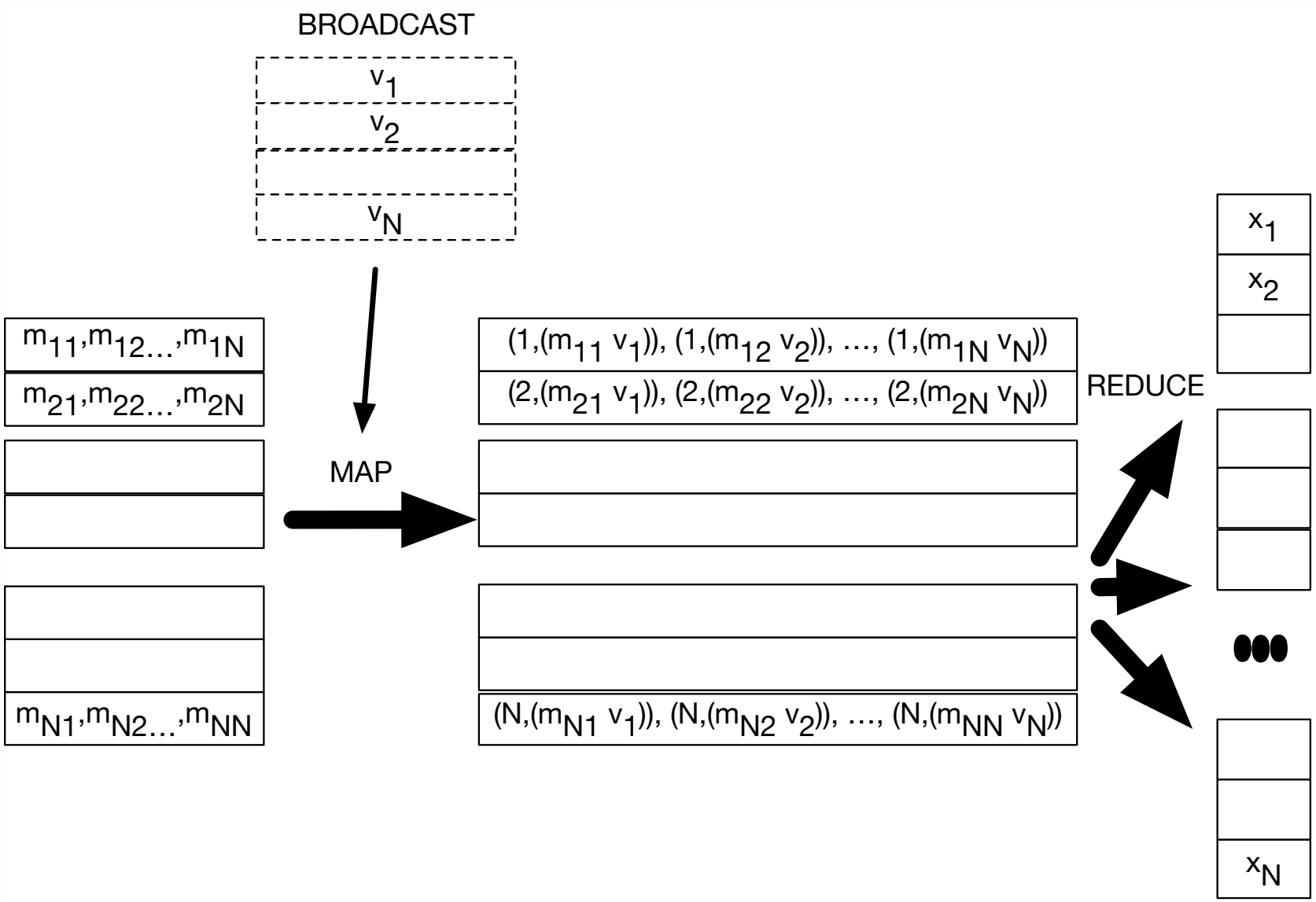
MATRIX-VECTOR MULTIPLICATION BY MR

Let us consider a $N \times N$ matrix M and a $N \times 1$ vector v. The result is a $N \times 1$ vector whose ith element is

$$x_i = \sum_{j=1}^{N} m_{ij} v_j$$

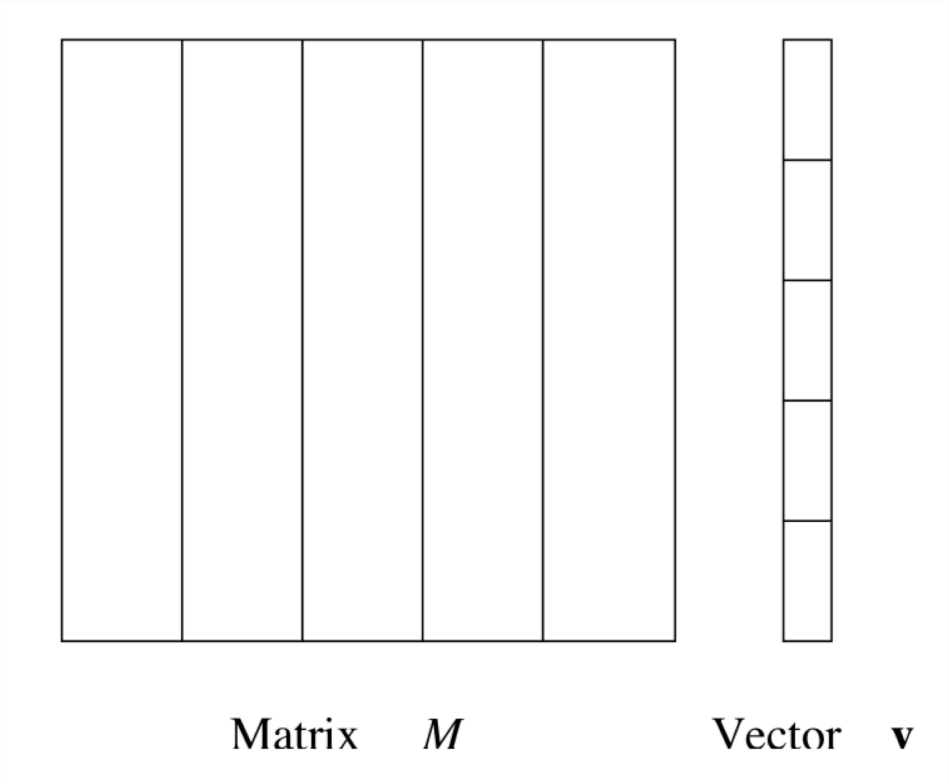
If the vector can fit in memory it can be broadcasted to each chunk and made available to all applications of the Map function

MATRIX-VECTOR MULTIPLICATION BY MR



WHAT IF n IS TOO LARGE?

Decomposition into S stripes to have the vector fitting into memory.



$$x_i = \sum_{j=1}^{N} m_{ij} v_j = \sum_{s=1}^{S} \sum_{j=s[1]}^{s[N_s]} m_{ij} v_j$$

where $s[1], \ldots, s[N_s]$ are the indices of the sth stripe.

RELATIONAL DATABASES

- A relation is a table whose rows are called tuples and column headers attributes. The set of attributes is called schema.
- Relational algebra operations: selection, projection, union/intersection/difference, join.
- Natural join: given two relations, it is the set of joint tuples such that the original ones agree on all the common attributes
- Grouping and aggregation: it partition tuples according to the values of the grouping attribute and computes an aggregated value (MAX, MIN, COUNT, SUM, AVG) related to an attribute which is not in the grouping set. The output is a tuple per group having as grouping attribute the common value and a component for each aggregation

MR IMPLEMENTATION OF RELATIONAL OPERATIONS

[7] presents MR implementations of relational operations:

- Selection:
  - Map: returns (t, t) if the selection condition is satisfied.
  - Red: returns the value.
- Projection:
  - Map: returns (t', t') where t' is the projected subset of the tuple t.
  - Red: it eliminates duplicates (e.g. two different original tuples which became the same after projection).
- Union of two relations having the same schema:
  - Map: return (t, t) for each tuple t.
  - Red: If key t has value list t or [t, t], return t.
- Intersection of two relations having the same schema:
  - Map: return (t, t) for each tuple t.
  - Red: If key t has value list [t, t], then return t. Otherwise, return nothing.

MR IMPLEMENTATION OF RELATIONAL OPERATIONS

- Difference $R - S$:
  - Map: For a tuple t in R, produce key-value pair (t, R), and for a tuple t in S, produce key-value pair (t, S).
  - Red: For each key t, if the associated value list is [R], then return t. Otherwise, return nothing.
- Join $R(A, B) \bowtie_B S(B, C)$:
  - Map: For each tuple (a, b) of R, produce the key-value pair $(b, (R, a))$. For each tuple (b, c) of S, produce the key-value pair $(b, (S, c))$.
  - Red: Each key value b will be associated with a list of pairs of the form (R, a) or (S, c). Construct from those pairs all possible triples of the form (a, b, c).

MR IMPLEMENTATION OF RELATIONAL OPERATIONS

Consider the relation R(A, B, C)

- Grouping on A and Aggregation on B: $\gamma_{A,\theta(B)}(R(A, B, C))$
  - Map: For each tuple (a, b, c) produce the key-value pair (a, b).
  - Red: Each key a represents a group. Apply the aggregation operator to the list of B-values associated with key a. The output is the pair (a, x) where x is the result of aggregation.

  In case of several grouping attributes, the key is a list of values of a tuple for those attributes. In case of several aggregations, the result of aggregation is a list of values too.

MATRIX MULTIPLICATION IN RELATIONAL FORM

Let us consider the product $P = MN$ where M is $[M_r, M_c]$, N is $[N_r, N_c]$ and $M_c = N_r$.

- The generic element of P is

$$p_{ik} = \sum_{j=1}^{M_c} m_{ij} n_{jk}$$

- This multiplication can be interpreted in a relational form where matrix M is a relation $M(I, J, V)$ with tuples (i, j, m_{ij}) and N is $N(J, K, V)$ with tuples (j, k, n_{jk}). This representation is particularly useful in the case of sparse matrices
- Multiplication as a natural join on attribute J followed by grouping on I and K and aggregation (sum).

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \end{bmatrix} = \begin{bmatrix} m_{11}n_{11} + m_{12}n_{21} & m_{11}n_{12} + m_{12}n_{22} & m_{11}n_{13} + m_{12}n_{23} \\ m_{21}n_{11} + m_{22}n_{12} & m_{21}n_{12} + m_{22}n_{22} & m_{21}n_{13} + m_{22}n_{23} \end{bmatrix}$$

$R_1 =$

I	J	M
1	1	m_{11}
1	2	m_{12}
2	1	m_{21}
2	2	m_{22}

$R_2 =$

J	K	N
1	1	n_{11}
1	2	n_{12}
1	3	n_{13}
...	...	...
3	1	n_{31}
3	2	n_{32}
3	3	n_{33}

$R_1 \bowtie_J R_2 =$

I	J	K	M	N
1	1	1	m_{11}	n_{11}
1	1	2	m_{11}	n_{12}
1	1	3	m_{11}	n_{13}
2	1	1	m_{21}	n_{11}
2	1	2	m_{21}	n_{12}
2	1	3	m_{21}	n_{13}
1	2	1	m_{12}	n_{21}
1	2	2	m_{12}	n_{22}
1	2	3	m_{12}	n_{23}
2	2	1	m_{22}	n_{21}
2	2	2	m_{22}	n_{22}
2	2	3	m_{22}	n_{23}

$\gamma_{I,K,SUM(M*N)} =$

I	K	P
1	1	$m_{11}n_{11} + m_{12}n_{21}$
1	2	$m_{11}n_{12} + m_{12}n_{22}$
1	3	$m_{11}n_{13} + m_{12}n_{23}$
2	1	$m_{21}n_{11} + m_{22}n_{21}$
2	2	$m_{21}n_{12} + m_{22}n_{22}$
2	3	$m_{21}n_{13} + m_{22}n_{23}$

MATRIX MULTIPLICATION IN MR (TWO STEPS)

Step 1:

- Map: for each element of M produce $(j, (M, i, m_{ij}))$ and for each element of N produce $(j, (N, k, n_{jk}))$
- Reduce: for each key j consider all the pairs of elements coming from M and N and compute $m_{ij} n_{jk}$ and return $((i, k), m_{ij} n_{jk})$

Step 2:

- Map: identity function
- Reduce: for each key (i, k) return the sum of the list of values associated to the key.

MATRIX MULTIPLICATION IN MR (SINGLE STEP)

- Map: for each element of M produce $((i, k), (j, M, m_{ij}))$ for $k = 1, 2, \ldots, N_c$ and for each element of N produce $((i, k), (j, N, n_{jk}))$ for $i = 1, 2, \ldots, M_r$
- Reduce: for each key (i, k) consider all the pairs (j, M, m_{ij}) and (j, N, n_{jk}). Sort them according to the j , multiply the values corresponding to the same j and sum them.

ROADMAP TO SCALABILITY

BIG DATA ANALYTICS IN PRACTICE

Two possible solutions

1. Use some recent open source machine learning platforms for big data
   - Mahout on top of Hadoop. It implements classification (perceptron, logistic regression, Naive Bayes, random forest, SVM), clustering, locally weighted regression, SVD (singular value decomposition), PCA, ICA
   - Trident ML on top of Storm. It implements linear classification (Perceptron), linear regression, clustering, feature scaling (standardization, normalization)
   - MLLib on top of Spark. It implements in iterative manner binary classification (SVM and Logistic Regression), linear regression, clustering (k-mean), collaborative filtering for recommender system.
2. Redesign and/or adapt algorithms in a scalable manner according to the MR paradigm.

SCALABLE DESIGN OF BIG DATA ALGORITHMS

Suppose we redesign our data analysis methods/algorithms in a scalable manner

- What can be reused in a big data setting and what not?
- We should first ask ourselves what makes data big: is it the number of features, the number of samples or both? This determines the distribution of the dataset.
- In case of vertical (horizontal) big datasets, each sample (feature) is an element and a chunk is a collection of samples (features)
- For instance in life science the critical dimension is the number of features, essentially because of the burden of making experiments or recruiting patients.
- However, if data is big in both directions, potential solutions have recourse to specific aspects of machine learning, like summation forms, divide-and-conquer modelling, subsampling and averaging

DISTRIBUTION OF LEARNING ALGORITHMS

We will consider a number of algorithms

- Supervised learning
  - Regression: least squares
  - Classification: Naive Bayes classification
  - Regression/Classification: KNN
  - Averaging algorithms
- Unsupervised learning: Kmeans
- Feature selection:
  - Ranking/correlation
  - mRMR feature selection

NOTA BENE

- MR is not a solution to every problem, not even every problem that profitably can use many compute nodes operating in parallel
- The use of a distributed-file-system makes sense only when files are very large and are rarely updated in place.
- The communication cost (i.e. size of the input) is the dominant cost, since each single task is typically very simple and the interconnect speed of a computing cluster is much slower than the processor execution speed.
- Different MR implementations of the same algorithm may induce different communication costs

ROUTES TO SCALABILITY

We can make several (not necessarily all) ML algorithms scalable by using one or more of these principles

1. Summation principle
   - expectation
   - cost function
   - gradient
   - log-likelihood
   - frequency
   - mean, variance
2. Independent tasks
   - feature ranking
   - cross-validation
   - testing
   - model selection (racing)
3. Statistical sampling properties (estimation)
4. Divide-and-conquer strategy: a complex nonlinear dependency can be decomposed in a number of local simpler dependencies.

SUMMATION FORM PRINCIPLE

- Machine learning and sums are strongly related.
- The accuracy of a learner (e.g. Mean-squared-error) is typically expressed as a function (typically expectation) of the data distribution.
- The estimation of the accuracy on the basis of observed data requires the computation of sampled versions of the expectation, i.e. means or sums over the data.
- Averaging approaches (e.g. bagging) rely on averages.
- Probability estimations via frequency require computations of sums as well.
- The majority of learning algorithms minimizes cost functions that can be written as sums: for instance sum of squared errors.
- If the error cost function is a sum, the gradient of the error is a sum too.

SUMMATION FORM PRINCIPLE

- Once an algorithm does sums over the data we can easily distribute the calculations over multiple workers
- We first divide the dataset into as many pieces as workers (partitioning)
- Then we give each worker a block of data (scatter)
- Eventually we aggregate the results (gather).
- Well-known example of algorithm in summation form is least-squares.

MONOIDS AND MR

- Monoid is an algebric structure with a single associative binary operation and an identity element.
- A monoid is composed by a set S, a binary mapping $\cdot : S \times S \to S$ and an identity element.
- An associative operation $\cdot$ satisfies the following equality for all a, b, c

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- In MR, a mapper is not constrained while the reducer may be distributed by implementing (the iterated application of) an associative operation
- Monoids: max, addition, multiplication, string concatenation, union , intersection
- Non monoids: mean, median

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

INDEPENDENCY

Several phases of the machine learning pipeline may be carried out independently, where independence can be interpreted both in computational (no communication) and statistical (no information) sense

- Prediction for different test values; note that prediction is the most expensive step in some algorithms (lazy).
- Cross-validation
- Averaging of estimators (bagging)
- Univariate ranking of features (assumption of independence)
- Generation of i.i.d. subsamples.

STATISTICAL SAMPLING

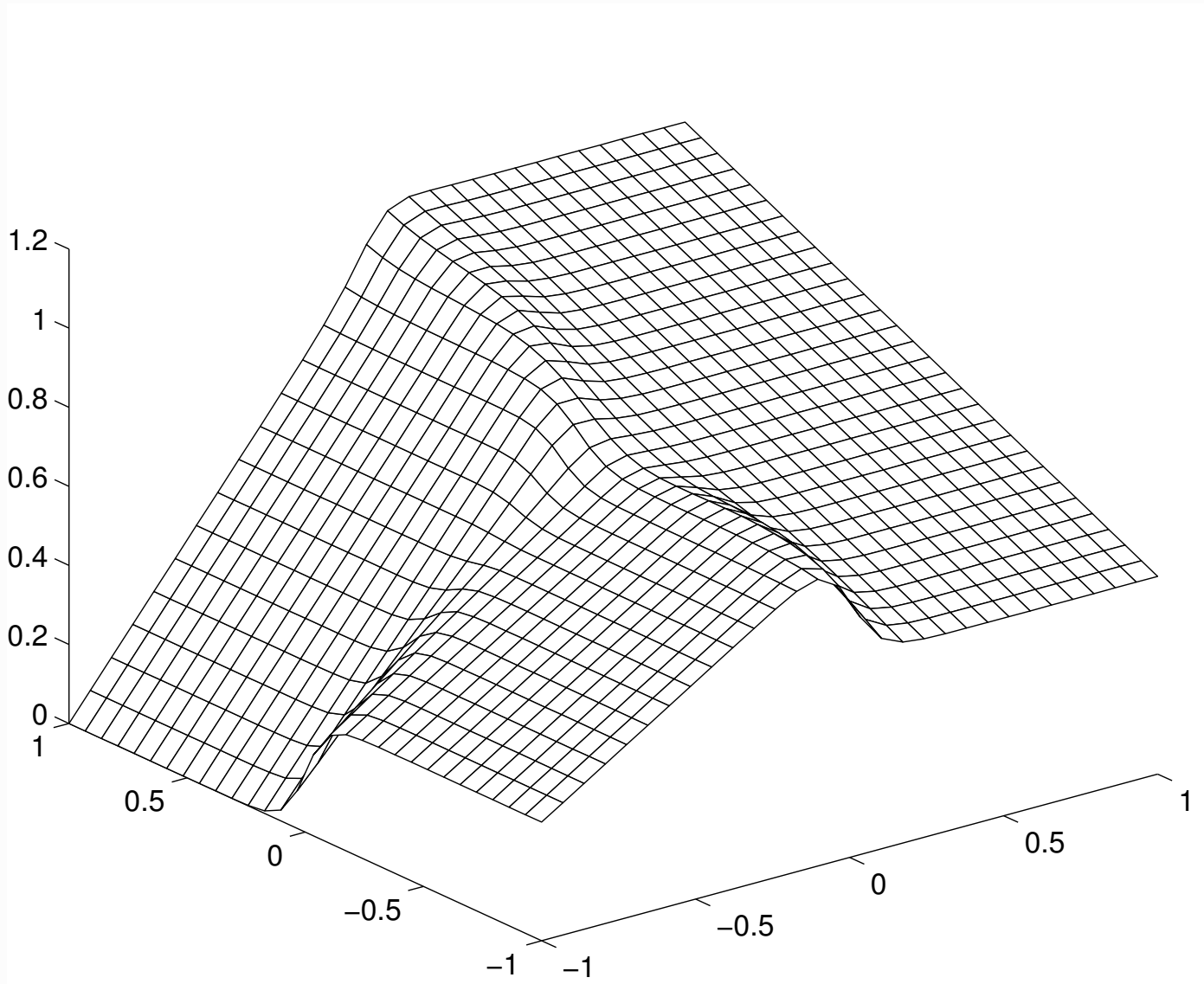
- Statistical sampling is the property process that allows inferences about a large collection of things (population), to be made from observations made on a relatively small number of individuals belonging to the population (the sample).
- The idea is to use the original big dataset as the population and portions of it as samples to infer properties of the original dataset

DIVIDE-AND-CONQUER

Learning strategy that attacks a complex problem by dividing it into simpler problems whose solutions can be combined to yield a solution to the original problem. This principle presents two main advantages.

1. simpler problems can be solved with simpler estimation techniques; in statistics this means to adopt linear techniques, well studied and developed over the years.
2. the learning method can better adjust to the properties of the available dataset.

Examples: Radial Basis Function, KNN, local learning


1.2
1
0.8
0.6
0.4
0.2
0
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

SOME MR ML ALGORITHMS

MULTIPLE LINEAR DEPENDENCY

- Consider a linear relation between an independent variable $x \in \mathcal{X} \subset \mathbb{R}^n$ and a dependent random variable $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}$

$$\mathbf{y} = \beta_0 + \beta_1 x_{\cdot 1} + \beta_2 x_{\cdot 2} + \cdots + \beta_n x_{\cdot n} + \mathbf{w}$$

  where $\mathbf{w}$ represents a random variable with mean zero and constant variance $\sigma^2_{\mathbf{w}}$.
- In matrix notation the equation can be written as:

$$\mathbf{y} = x^T \beta + \mathbf{w}$$

  where x stands for the $[p \times 1]$ vector $x = [1, x_{\cdot 1}, x_{\cdot 2}, \ldots, x_{\cdot n}]^T$, $\beta = [\beta_0, \ldots, \beta_n]^T$ is the vector of parameters and $p = n + 1$ is the total number of model parameters.
- NB: in the following $x_{\cdot i}$ will denote the ith variable of the vector x, while x_i will denote the ith observation of the vector x.

THE MULTIPLE LINEAR REGRESSION MODEL

Consider N observations $D_N = \{\langle x_i, y_i\rangle : i = 1, \ldots, N\}$, where $x_i = (1, x_{i1}, \ldots, x_{in})$, generated according to the previous model. We suppose that the following multiple linear relation holds

$$Y = X\beta + W$$

where Y is the $[N \times 1]$ response vector, X is the $[N \times p]$ data matrix, whose j^{th} column of X contains readings on the j^{th} regressor, β is the $[p \times 1]$ vector of parameters

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ . \\ . \\ . \\ x_N^T \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ . \\ \beta_n \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_N \end{bmatrix}$$

where w_i are assumed uncorrelated, with mean zero and constant variance $\sigma^2_{\mathbf{w}}$ (homogeneous variance). Then $\mathrm{Var}[\mathbf{w}_1, \ldots, \mathbf{w}_N] = \sigma^2_{\mathbf{w}} I_N$.

NORMAL EQUATIONS

The **least-squares estimator** $\hat{\boldsymbol{\beta}}$ minimizes the cost function

$$\hat{\beta} = \arg\min_{b} \sum_{i=1}^{N} (y_i - x_i^T b)^2 = \arg\min_{b} \left((Y - Xb)^T (Y - Xb) \right)$$

It can be shown that it satisfies the least-squares normal equations

$$(X^T X)\hat{\beta} = X^T Y$$

Assuming X is of full column rank, we obtain

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

where the $X^T X$ matrix is a symmetric $[p \times p]$ matrix.

MR DISTRIBUTION OF THE LEAST-SQUARES SOLUTION

Two settings

- very large N $>>$ n with n small: distribution of the computations of $X^T X$ and $X^T Y$
- very large N and n: iterative solution by stochastic gradient descent (next class)

LEAST-SQUARES SUMMATION FORM

Suppose $N = 3$, $p = 2$

$$X^T X = \begin{bmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} =$$

$$= \begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 & x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} \\ x_{12}x_{11} + x_{22}x_{21} + x_{32}x_{31} & x_{12}^2 + x_{22}^2 + x_{32}^2 \end{bmatrix}$$

$$x_1^T x_1 = \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \end{bmatrix} = \begin{bmatrix} x_{11}^2 & x_{11}x_{12} \\ x_{11}x_{12} & x_{12}^2 \end{bmatrix}$$

then

$$X^T X = \sum_{i=1}^{N} x_i^T x_i$$

where x_i is the $[1, p]$ vector denoting the ith row of the matrix X.

LEAST-SQUARES SUMMATION FORM

Analogously it can be shown that

$$X^T Y = \sum_{i=1}^{N} x_i^T y_i$$

where y_i is the ith value of the vector Y.

DISTRIBUTED COMPUTATION OF X^tX AND X^ty

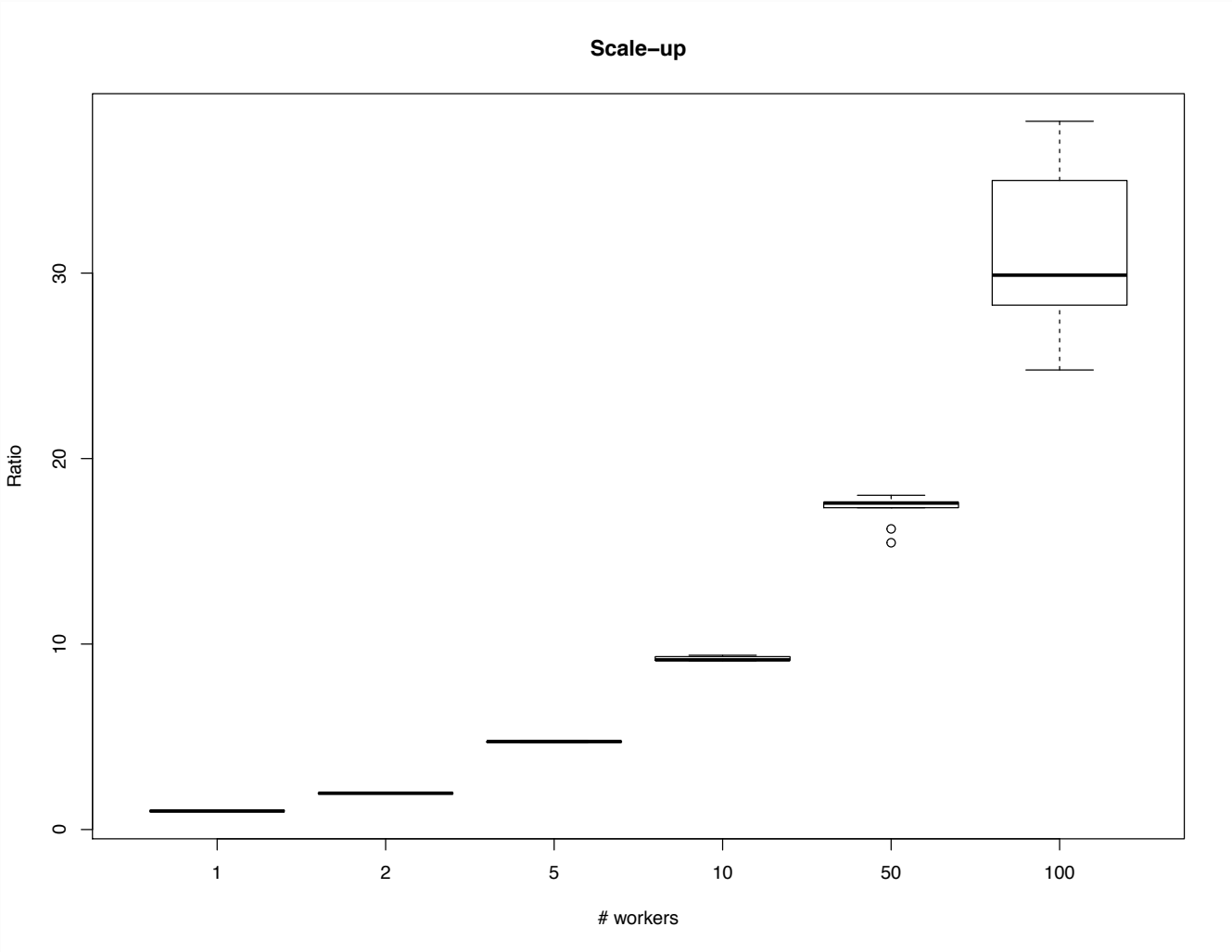
Create a matrix Z of size $[N, p+1]$ by concatenating X and Y

1. Store Z as a Spark Resilient Distributed Dataset (RDDs).
2. Computation of partial X^TX by mapping. For each ith row of X compute $x_i^Tx_i$ of size $[p, p]$
3. Reduce. Sum all $x_i^Tx_i$.
4. Computation of partial X^TY by mapping. For each ith row of X $(i = 1, \ldots, N)$ compute $x_i^Ty_i$ of size $[p, 1]$
5. Reduce. Sum all $x_i^Ty_i$.
6. Collect. Collect the reduced matrices X^TX and X^TY, and compute $\hat{\beta}$ in a conventional manner.

SOME CONSIDERATIONS

- Note that instead of a row-wise computation it is possible to implement a partition-wise computation
- Summation form allows a linear speed up with the number of workers.
- Note that some intermediate results of least-squares are useful in other tasks, too.
- X^TX is an estimation of the covariance matrix (if all the columns of X are normalized, i.e. zero mean and unit variance). This matrix is very frequently used in linear statistics, notably in Principal Component Analysis, Linear Discriminant Analysis
- X^TY is an estimation of the correlation input/output vector (again in normalized settings). This vector is very frequently used in linear statistics, notably in ranking of features
- Note that above computations are scalable with N but not n.

SCALE-UP ASSESSMENT



NAIVE BAYES CLASSIFIER

- The Naive Bayes (NB) classifier has shown in some domains a performance comparable to that of neural networks and decision tree learning.
- Consider a binary classification problem with n inputs and a random output variable $\mathbf{y}$ that takes values in the set $\{c_1, \ldots, c_K\}$.
- The Bayes optimal classifier should return

$$c^*(x) = \arg \max_{k=1,\ldots,K} \text{Prob}\{\mathbf{y} = c_k | x\}$$

- We can use the Bayes theorem to rewrite this epression as

$$c^*(x) = \arg \max_{k=1,\ldots,K} \frac{\text{Prob}\{x|\mathbf{y} = c_k\}\,\text{Prob}\{\mathbf{y} = c_k\}}{\text{Prob}\{x\}} =$$

$$= \arg \max_{k=1,\ldots,K} \text{Prob}\{x|\mathbf{y} = c_k\}\,\text{Prob}\{\mathbf{y} = c_k\}$$

NAIVE BAYES CLASSIFIER

- How to estimate these two terms with finite dataset?
- It is easy to estimate Prob $\{\mathbf{y} = c_k\}$ with the frequency with which each target class occurs in the training set. The estimation of Prob $\{x|\mathbf{y} = c_k\}$ is much harder.
- NB is based on the simplifying assumption that inputs are conditionally independent given the target value:

$$\text{Prob}\{x|\mathbf{y} = c_k\} = \text{Prob}\{x_1, \ldots, x_n|\mathbf{y} = c_k\} = \prod_{j=1}^{n} \text{Prob}\{x_j|\mathbf{y} = c_k\}$$

- The NB classification is then

$$c_{NB}(x) = \arg \max_{k=1,\ldots,K} \text{Prob}\{\mathbf{y} = c_k\} \prod_{j=1}^{n} \text{Prob}\{x_j|\mathbf{y} = c_k\}$$

DISTRIBUTED COMPUTATION OF NAIVE BAYES CLASSIFIER

Let us suppose that all data are categorical

- If inputs x_j are binaries the estimation of Prob $\{x_j | \mathbf{y} = c_k\}$ boils down to the counting of the frequencies of the occurrences of the different values of x_j for a class c_k.
- We need to sum over $x_j = 0$ and $x_j = 1$ for each label c_k
- Once data are partitioned, it is enough to compute a sum for each block and then aggregate the results

EXAMPLE

Let us consider the following dataset (example from [6])

Day	Outlook	Temperature	Humidity	Wind	PlayTennis (classificati
D_1	Sunny	Hot	High	Weak	No
D_2	Sunny	Hot	High	Strong	No
D_3	Overcast	Hot	High	Weak	Yes
D_4	Rain	Mild	High	Weak	Yes
D_5	Rain	Cool	Normal	Weak	Yes
D_6	Rain	Cool	Normal	Strong	No
D_7	Overcast	Cool	Normal	Strong	Yes
D_8	Sunny	Mild	High	Weak	No
D_9	Sunny	Cool	Normal	Weak	Yes
D_{l0}	Rain	Mild	Normal	Weak	Yes
D_{11}	Sunny	Mild	Normal	Strong	Yes
D_{12}	Overcast	Mild	High	Strong	Yes
D_{13}	Overcast	Hot	Normal	Weak	Yes
D_{14}	Rain	Mild	High	Strong	No

EXAMPLE: MAP

The map() function counts the attributes and their associations with the classification classes. For instance the map of the first row will return:

```
(<Sunny,No>, <1>)
(<Hot,No>, <1>)
(<High,No>, <1>)
(<Weak,No>, <1>)
(<CLASS,No>, <1>)
```

EXAMPLE: REDUCER INPUT

Key	Value
<CLASS,No>	[<1>,<1>,<1>,<1>,<1>]
<CLASS,Yes>	[<1>,<1>,<1>,<1>,<1>,<1>,<1>,<1>,<1>]
<Cool,No>	[<1>]
<Cool,Yes>	[<1>,<1>,<1>]
<High,No>	[<1>,<1>,<1>,<1>]
<High,Yes>	[<1>,<1>,<1>]
<Hot,No>	[<1>,<1>]
<Hot,Yes>	[<1>,<1>]
<Mild,No>	[<1>,<1>]
<Mild,Yes>	[<1>,<1>,<1>,<1>]
<Normal,No>	[<1>]
<Normal,Yes>	[<1>,<1>,<1>,<1>,<1>,<1>]
<Overcast,Yes>	[<1>,<1>,<1>,<1>]
<Rain,No>	[<1>,<1>]
<Rain,Yes>	[<1>,<1>,<1>]
<Strong,No>	[<1>,<1>,<1>]
<Strong,Yes>	[<1>,<1>,<1>]
<Sunny,No>	[<1>,<1>,<1>]
<Sunny,Yes>	[<1>,<1>]
<Weak,No>	[<1>,<1>]
<Weak,Yes>	[<1>,<1>,<1>,<1>,<1>,<1>]

EXAMPLE: REDUCER OUTPUT

Key	Value
<CLASS,No>	5
<CLASS,Yes>	9
<Cool,No>	1
<Cool,Yes>	3
<High,No>	4
<High,Yes>	3
<Hot,No>	2
<Hot,Yes>	2
<Mild,No>	2
<Mild,Yes>	4
<Normal,No>	1
<Normal,Yes>	6
<Overcast,Yes>	4
<Rain,No>	2
<Rain,Yes>	3
<Strong,No>	3
<Strong,Yes>	3
<Sunny,No>	3
<Sunny,Yes>	2
<Weak,No>	2
<Weak,Yes>	6

EXAMPLE: CONDITIONAL PROBABILITY

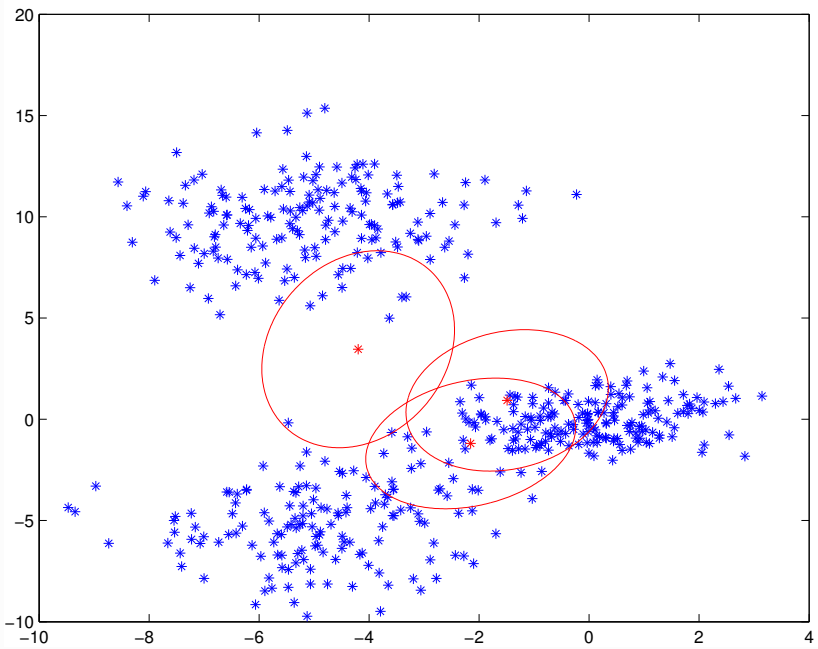
From the output of the reducer it is easy to compute the conditional probability.
For instance for the variable $\mathbf{x}_1$=Outlook that can take as values {Sunny, Overcast, Rain} we obtain

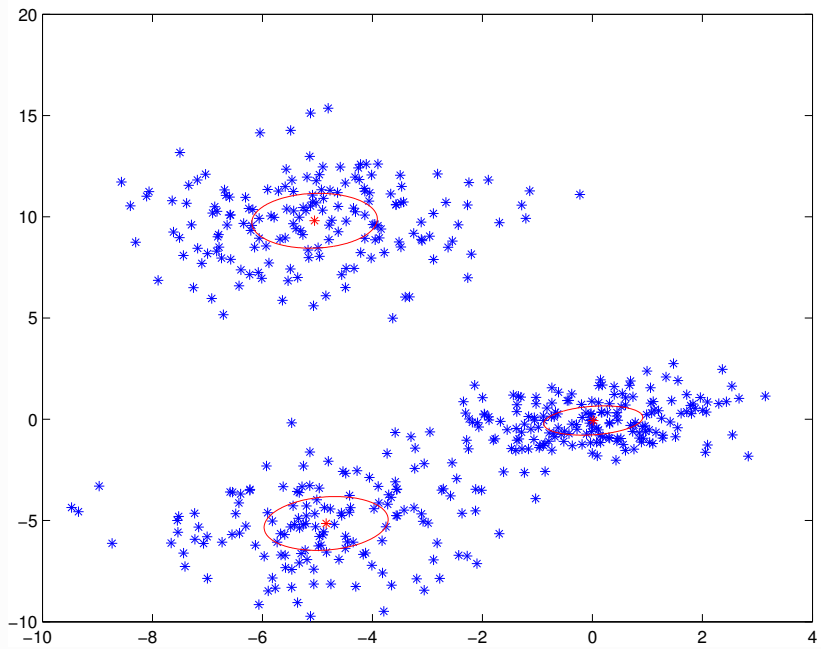
$$\text{Prob}\left\{\mathbf{x}_1 = \text{Sunny}|\mathbf{y} = \text{No}\right\} = \frac{3}{3+0+2} = \frac{3}{5}$$

Once the probability table is available it is possible to use it to perform Naive Bayes classification on on new input vectors.

K-MEANS

- Given $K > 0$ (where K is the number of clusters) and a set of N n-dimensional objects, clustering is the process of grouping a set of N n-dimensional into K clusters of similar objects.
- Objects should be similar to one another within the same cluster and dissimilar to those in other clusters.
- K-Means is a distance-based unsupervised clustering algorithm.
- K-Means clustering has many useful applications. For example, it can be used to find a group of consumers with common behaviors, or to cluster documents based on the similarity of their contents.
- Selection of K is specific to the application or problem domain. There is no magic formula to find it.


20
15
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4


20
15
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4

K-MEANS

- Initially, K points $C = \{c_1, \ldots, c_K\}$ are chosen as cluster centers; these are called cluster centroids.
- There are many ways to initialize the cluster centroids, one of which is to choose the K points randomly from the sample of n points.
- Once the K initial cluster centroids are chosen, we calculate the distance from every point in the input set to each of the K centers, and then assign each point to the specific cluster center whose distance is closest.
- When all objects have been assigned, then we again recalculate the positions of the K centroids.
- These two steps are repeated until the cluster centroids no longer change (or change very little).

DISTRIBUTED COMPUTATION OF K-MEANS

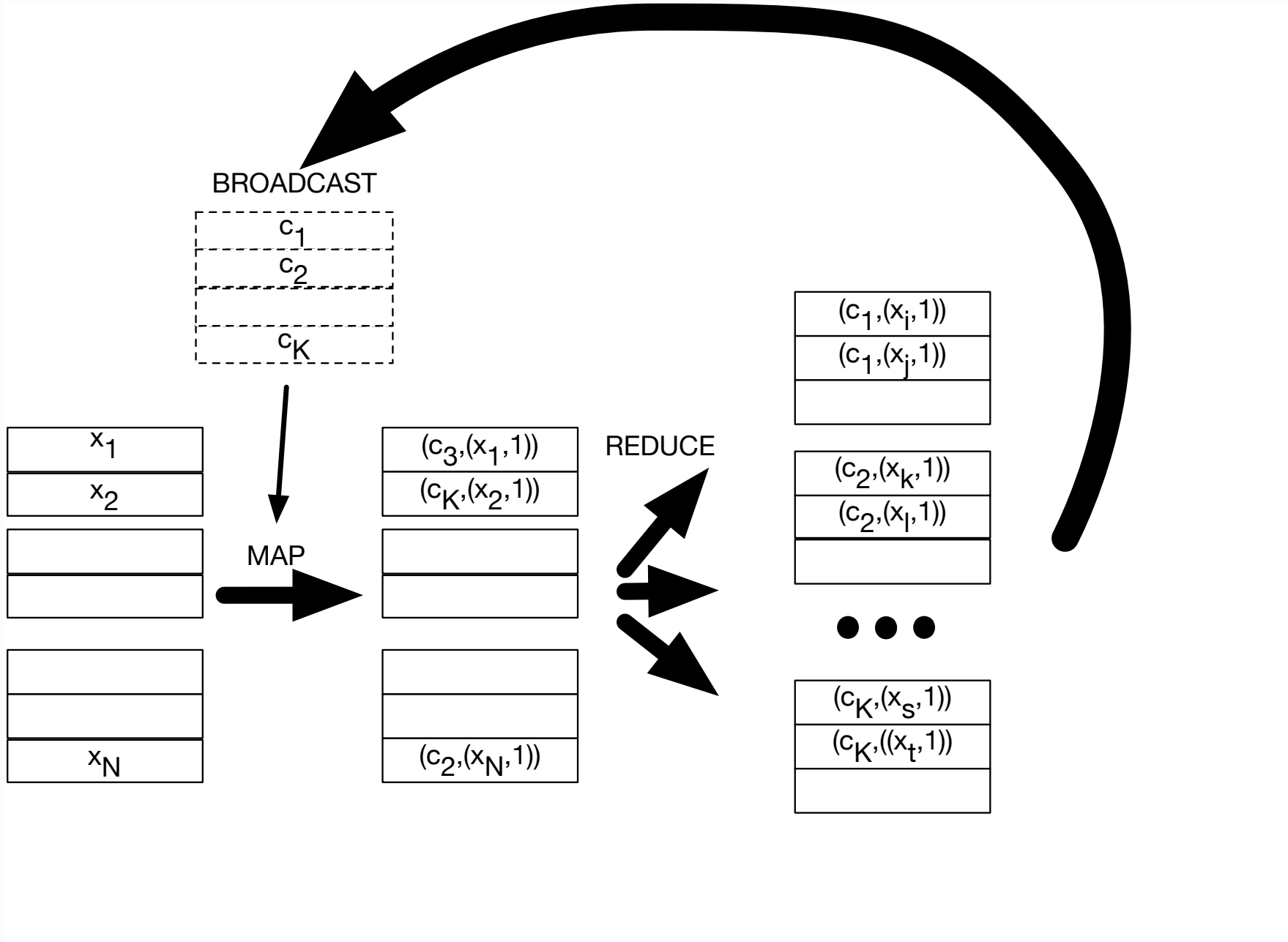
- Cost function is again a sum

$$\arg \min_{c_1,\ldots,c_K} \sum_{k=1}^{K} \sum_{x_i \in \mathcal{C}_k} \|x_i - c_k\|$$

  where $\mathcal{C}_k$ is the set of points which belong tothe kth cluster.
- For a very large N the bottleneck is the computation of the distances of the points to the centroids

MR COMPUTATION OF K-MEANS

1. We partition the dataset and we broadcast the centroid coordinates.
2. We distribute the computation of the distances to a number of parallel workers
3. A map function returns for each x_i a (key,value) pair where the key is the index k_i of the closest centroid, and the value is a pair $(x_i, 1)$ that will allow to compute the sum and number of observations in each cluster.
4. A reduce step returns the updated coordinates of each cluster centroid
5. By iterating the steps above we converge to the solution.


BROADCAST
c_1
c_2
c_K
x_1
x_2
x_N
MAP
$(c_3,(x_1,1))$
$(c_K,(x_2,1))$
$(c_2,(x_N,1))$
REDUCE
$(c_1,(x_i,1))$
$(c_1,(x_j,1))$
$(c_2,(x_k,1))$
$(c_2,(x_l,1))$
$(c_K,(x_s,1))$
$(c_K,((x_t,1))$

A KNN CLASSIFIER

Suppose a training set is available and that the classification is required for a $[1, n]$ query vector. The classification procedure of a $\mathcal{K}$-NN classifier can be summarized in these steps:

1. Compute the distance between the query and the training samples according to a predefined metric.
2. Rank the neighbors on the basis of their distance to the query.
3. Select a subset of the $\mathcal{K}$ nearest neighbors. Each of these neighbors has an associated class.
4. Return the class which characterizes the majority of the $\mathcal{K}$ nearest neighbors.

DISTRIBUTED KNN

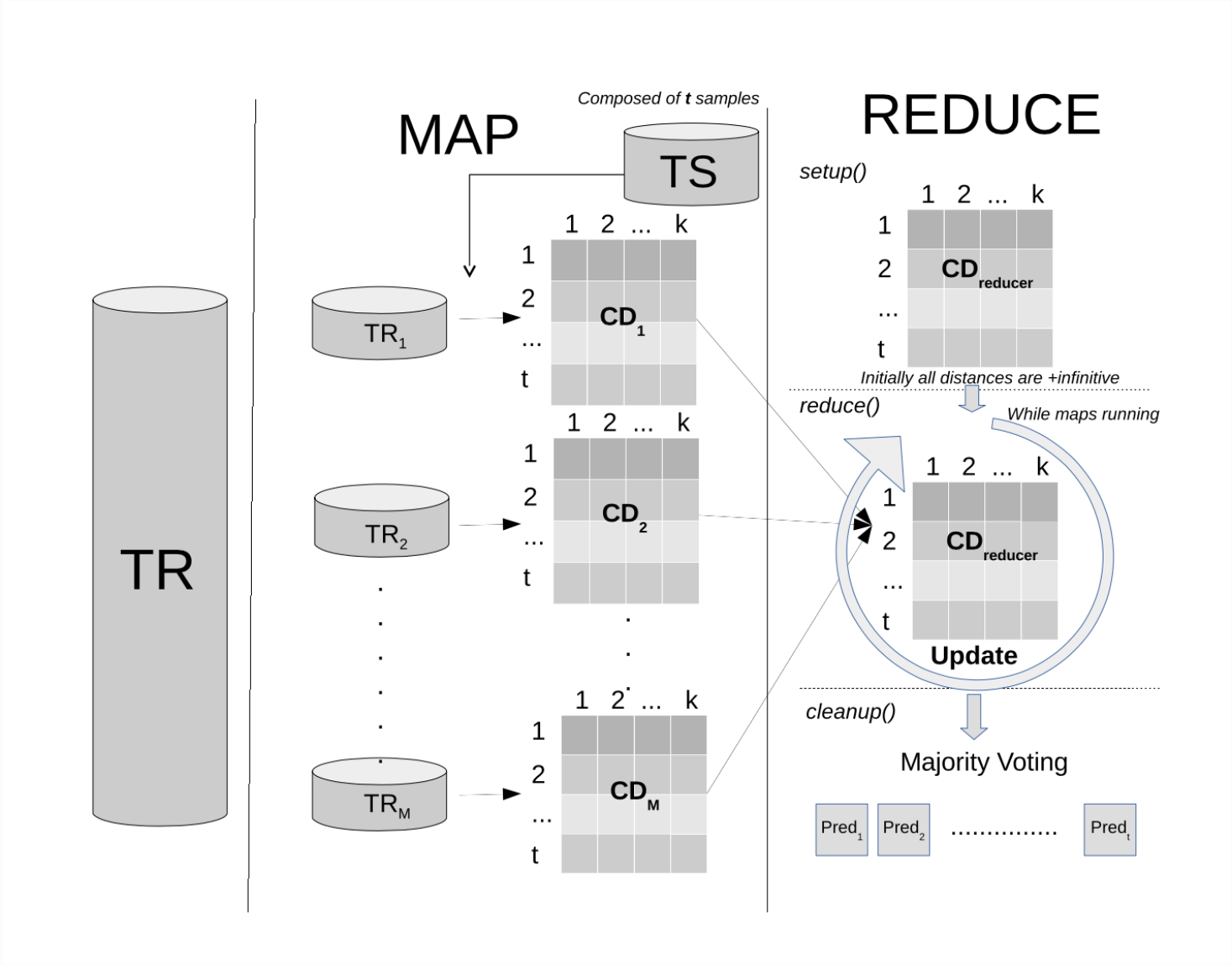
- Lazy algorithm: nothing is done until a query is done.
- No training computational cost.
- All computational effort concerns the prediction step: for each query point, neighbours have to be identified and classification returned
- Problem: comparing a query point with each sample in a database is infeasible because of the linear complexity O(N).
- Curse of dimensionality makes the problem still worse.
- However, retrieving a set of approximate nearest neighbours (ANN) is often sufficient.
- Distribution of the prediction computational effort.

MR-KNN

In [5], the authors propose a MapReduce-based approach for k-Nearest neighbor classification.
In this approach the test best is broadcast to all workers.

- Map: for each chunk of data, it computes similarity between all the test examples and a portion of the training set. As a result of each map, the k nearest neighbors together with their computed distance values will be emitted to the reduce stage.
- Reduce: it determines which are the final k nearest neighbors from the list provided by the maps. It is implemented as the merging of two sorted lists of size k.

MR-KNN


MAP
Composed of t samples
TS
TR
TR1
TR2
TRM
CD1
CD2
CDM
1 2 ... k
1
2
...
t
REDUCE
setup()
CDreducer
Initially all distances are +infinitive
reduce()
While maps running
Update
cleanup()
Majority Voting
Pred1
Pred2
Predt

From [5]

PASTING AND RANDOM SUBSPACES

Let us consider a supervised learning problems for which the dataset is so large that it cannot be loaded into memory

- Breiman (the genial inventor of regression tree, random forest, bagging,...) proposed the Pasting method [2] to tackle this problem by learning an ensemble of models individually built on random subsets of the training examples, hence alleviating the memory requirements since the base models would be built on only small parts of the whole dataset.
- Ho [3] proposed to learn an ensemble of models individually built on random on random subsets of the input variables (or features).

RANDOM PATCHES

- The two ideas has been merged by [4] in the Random Patch algorithm, which consists in creating a number of datasets (covering a portion of the features and a portion of the observations), fit them with a model and combine the predictions.
- The portion of the feature and sample space is controlled by two hyper-parameters.
- Highly parallel: each patch can be managed independently by a processor.
- If the size of the patch is compatible with the central memory, conventional memory architectures may be used.
- Open challenges: the analysis of subsets of data may present different statistical properties than the overall dataset. For example, confidence intervals based on subsets of data will generally be wider than confidence intervals based on the original data (papers of M. Jordan on statistics, computation and scalability).

HASHES

- An hash function takes a hash-key value (of any type) as input and returns a bucket integer number in a range $[0, B - 1]$ as result.
- Aim of hash function is to randomize hash-keys, i.e. to send approximately an equal number of hash keys to each bucket number
- If the hash-key is not an integer conversion techniques are used to transform it to an integer value.
- Hash table is a simple way to construct an index. All the elements with the same bucket number are placed in the same bucket linked list.

LOCALITY SENSITIVITY HASHING

Suppose we have a very large set of elements and we wish to compute the similarity of every pair

- Idea: hash each element several times in such a way that similar elements are more likely to be hashed to the same bucket than dissimilar ones.
- Any pair hashed to the same bucket is a candidate pair for similarity
- False positives: dissimilar elements in the same bucket
- False negatives: truly similar elements not hashing to the same bucket.
- LSH also known as near-neighbour search

PROPERTIES OF LOCALLY SENSITIVE FUNCTIONS

Set of functions that can be combined to distinguish between pairs at a low distance from pairs at a high distance. A LSH function hashes the two items and the decision is based on whether or not the hashing result is the same.
Conditions [7]

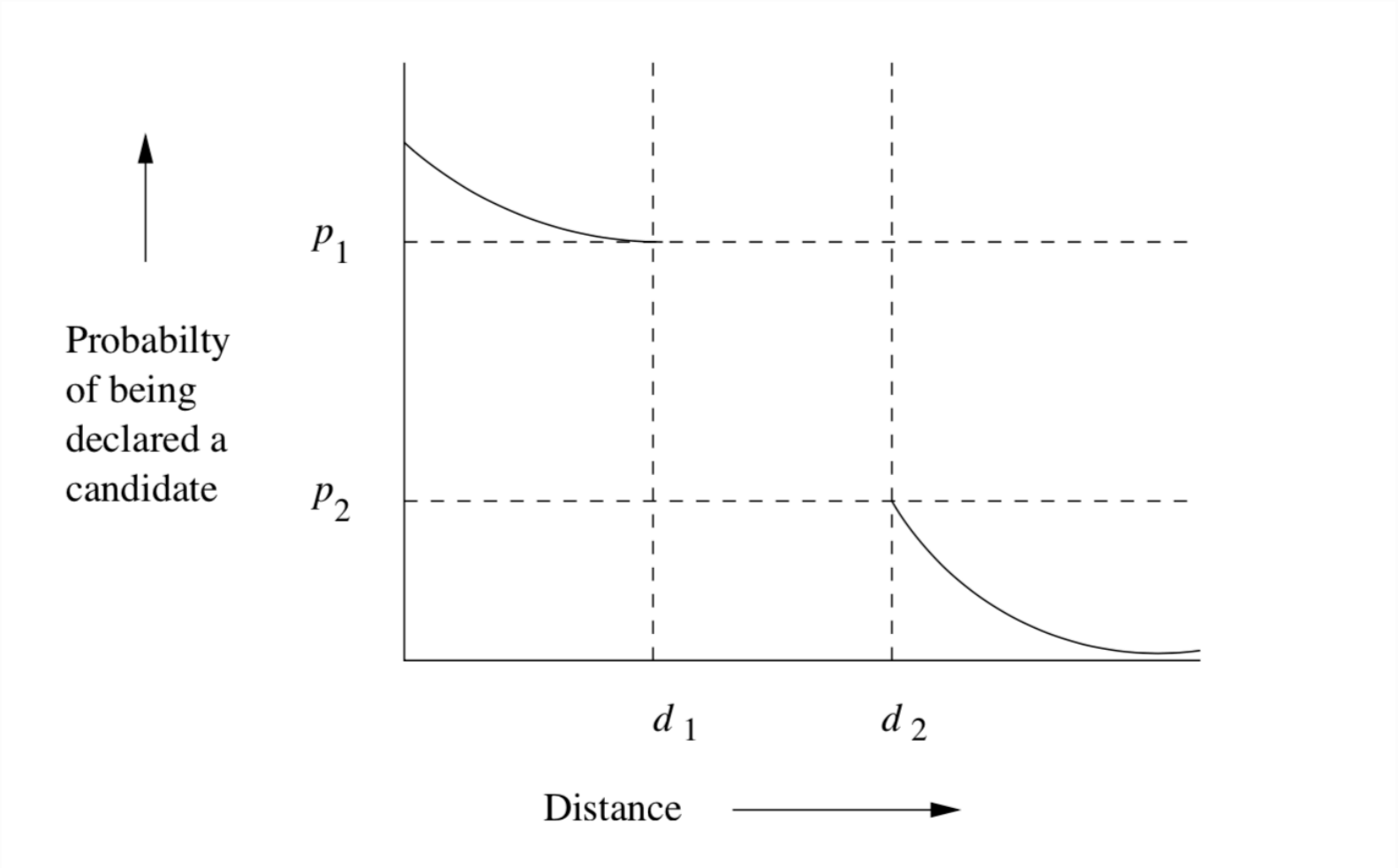
- Probability that make candidate pairs close should be higher than the probability of making distant pairs close.
- Statistically independent
- Computationally efficient

FAMILY OF LOCALLY SENSITIVE FUNCTIONS

Let us consider a pair of items (x, y), a family F of hashing functions and two distances $d_1 < d_2$. The family is said to be (d_1, d_2, p_1, p_2)-sensitive if for every $f \in F$

1. if $d(x, y) \leq d_1$ then the probability that $f(x) = f(y)$ is at least p_1
2. if $d(x, y) \geq d_2$ then the probability that $f(x) = f(y)$ is at most p_2

Note that $1 - p_1$ denotes the probability of a false negative (i.e. the two items are closer than d_1 but the hashing does not return equality) while p_2 denotes the probability of a false positive (i.e. the two items are farther than d_2 but the hashing returns an equality).


Probabilty
of being
declared a
candidate
p_1
p_2
d_1
d_2
Distance

EXAMPLES OF LSH FUNCTIONS

There are families of LSH functions satisfying the (d_1, d_2, p_1, p_2)-sensitive property for several distance functions, like

- Jaccard distance (in case of sets): minhash functions
- Hamming distance
- Cosine distance (angle between vectors)

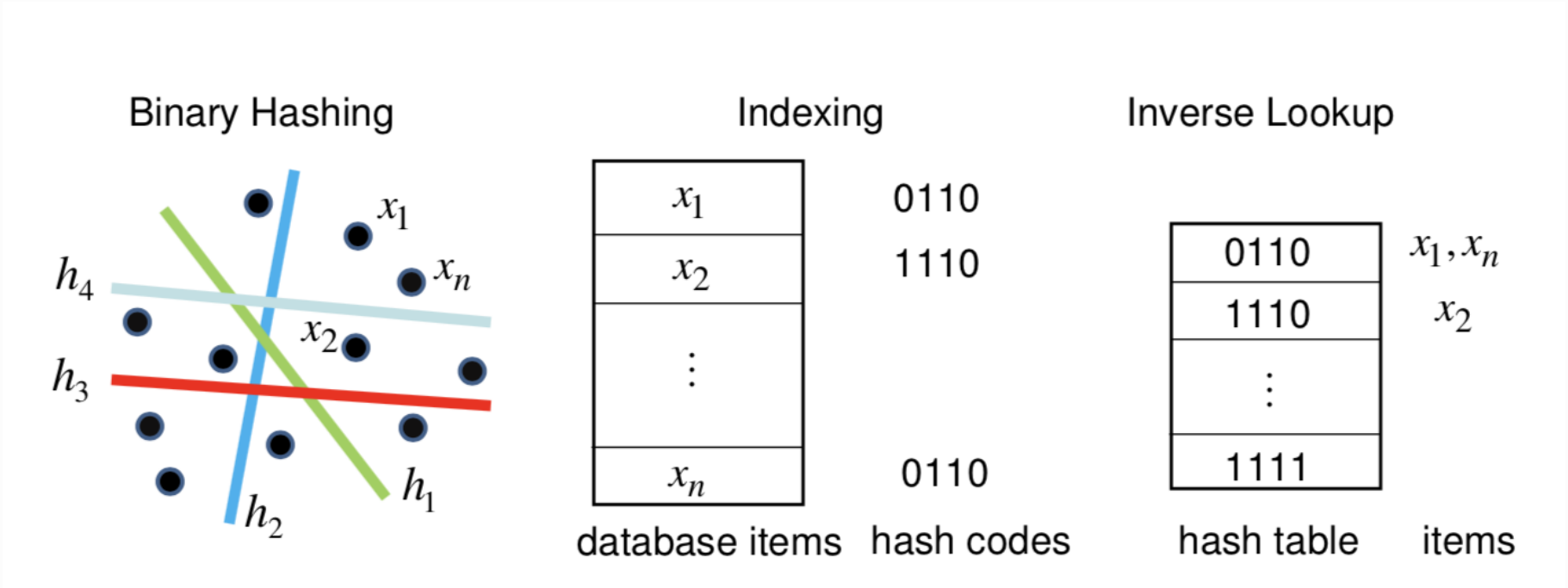
It is also possible to create new families of LSH functions by combining existing families by the use of AND or OR operator.

LSH FUNCTIONS FOR COSINE DISTANCE

Let us consider two vectors $x_1 \in \mathbb{R}^n, x_2 \in \mathbb{R}^n$ in a n-dimensional space and the cosine distance.

- The locality-sensitive family for the cosine distance is a set of B binary codes functions, each built from a randomly chosen vector $h_b, b = 1, \ldots, B$
- $f_b(x) = \text{sign}(h_b \cdot x)$, i.e. $f_b(x_1) = f_b(x_2)$ if and only if the dot products $h_b \cdot x_1$ and $h_b \cdot x_2$ have the same sign
- each n-dimensional vector is transformed in a B-dimensional binary code. Since $n >> B$ coding reduces storage. and Hamming distance can be calculated efficiently in a bitwise manner.
- a modified version is the sketch which consists in restricting to vectors h_b whose components are $+1$ and -1

n = 2 input dimensions, B = 4 bits binary code:



From [9].

MR DIVIDE AND CONQUER

In case of massive training sets of dimensionality n it could be effective to partition it according to a locality principle.
By defining a number B of sketch vectors $h_b \in \mathbb{R}^n$, it is is possible to project each training point in a s dimensional space and use the hashing for defining sets of neighboring points.
Training:

1. Broadcast the S matrix of dimensionality $[n, B]$ where each column corresponds to a sketch vector
2. Map: for each training input vector x_i compute the matrix product $x_i \cdot S$ and store the sign vector as hashing key

For each test point q:

1. compute the matrix product $q \cdot S$ and store the sign vector
2. Filter the training set by selecting only the subset with the same hashing
3. Make a conventional learning procedure by using a subset of the training set

LEARNING TO HASH

- LSH are data independent hash functions.
- In spite of their elegant theoretical properties their performance has been shown insufficient in many real-world settings.
- To achieve high precision long hash codes are required and semantic proximity is not always respected (semantic gap).
- ”Learning to hash” [9] are recent approaches which use machine learning to exploit data distribution or class labels to optimize the hash function.
- Particularly useful in images where the similarity (or distance) between image pairs is usually not defined via a simple metric.

FEATURE SELECTION AND MUTUAL INFORMATION

- In terms of mutual information the feature selection problem can be formulated as follows. Given an output target $\mathbf{y}$ and a set of input variables $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ selecting the optimal subset of d variables boils down to the following optimization problem

$$X^* = \arg \max_{X_S \subset X, |X_S| = d} I(X_S; \mathbf{y})$$

- This maximization task can be tackled by adopting an incremental approach.
- Let $X = \{x_i\}, i = 1, \ldots, n$ the whole set of variables and $\mathbf{X}_S$ the current set of selected variables. The task of adding a variable $x^* \in S - X$ can be addressed by solving

$$x^* = \arg \max_{\mathbf{x}_k \in X - X_S} I(\{\mathbf{X}, \mathbf{x}_k\}; \mathbf{y})$$

  This is known as the maximal dependency problem and requires a multivariate estimation of the mutual information.
- Filter approaches rely on some low variate approximation.

RANKING

- It assesses the importance (or relevance) of each variable with respect to the output by using a univariate measure. They are supervised techniques of complexity O(n).
- Measures of relevance which are commonly used are:
  - Pearson correlation (the greater the more relevant) which assumes linearity;
  - mutual information (the greater the more relevant).
- After the univariate assessment the method ranks the variables in a decreasing order of relevance.
- These methods are fast (complexity O(n)) and their output is intuitive and easy to understand. At the same time they disregard redundancies and higher order interactions between variables (e.g. genes).
- The best k features taken individually do not necessarily constitute the best k variate vector.

THE MRMR APPROACH

The mRMR (mimimum-Redundancy Maximum-Relevancy) feature selection strategy is a forward-selection strategies which approximates

$$\arg \max_{x_k \in X - X_S} I(\{X_S, x_k\}; y)$$

with

$$x^*_{MRMR} = \arg \max_{x_k \in X - X_S} \left[I(x_k; y) - \frac{1}{m} \sum_{x_i \in X_S} I(x_i; x_k) \right]$$

where X_S is the set of features which have been already selected and m is the size of X_S.

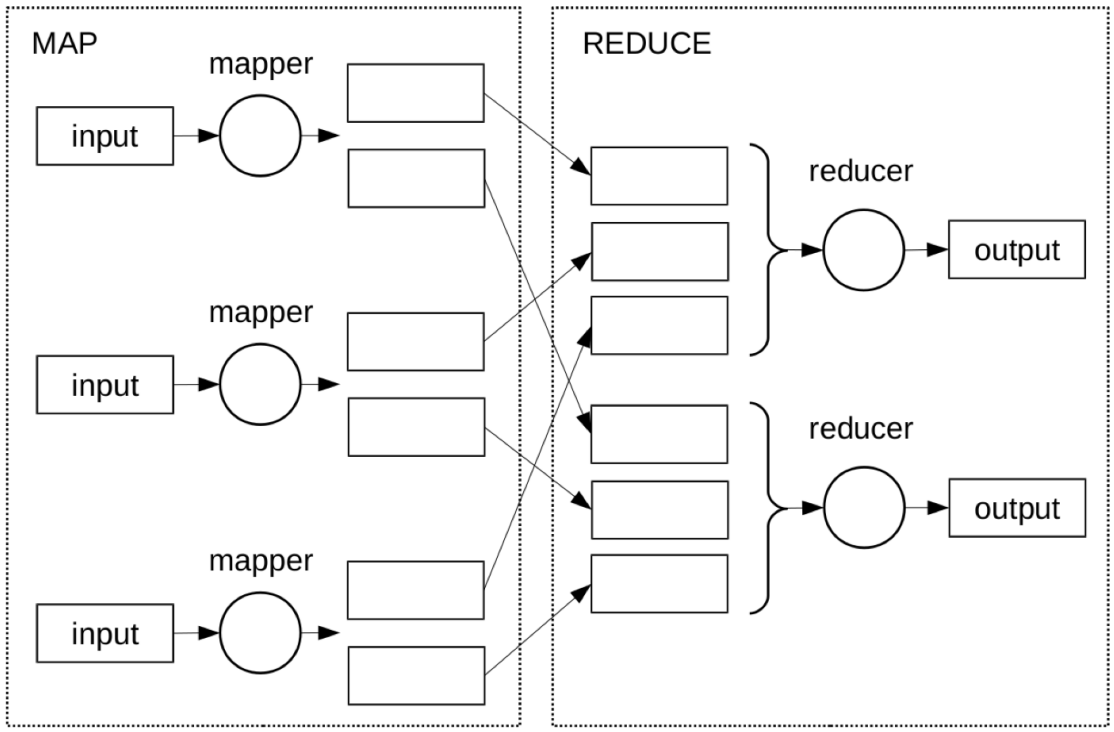
MR FOR FILTER SELECTION

Let us consider a supervised problem with n input features and one target. Assume the training input/output data are collected in a rectangular format and that each row contains both the input vector and the associated output.

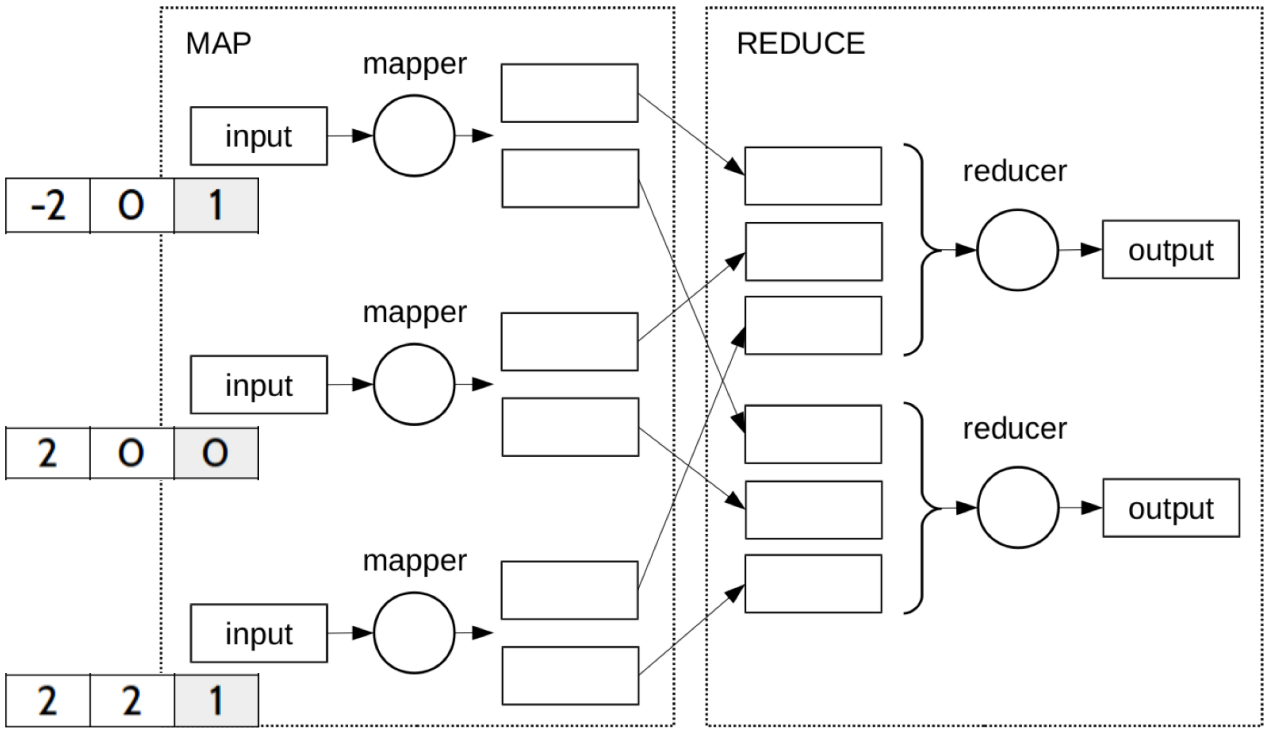
- Map: from each row it produces a set of key-value tuples where the key is the feature index and the value is the pair composed of the feature value and the corresponding target values.
- Reduce: the tuples with the same key (referring to the same feature) are grouped and the bivariate score (e.g. correlation or mutual information) computed.

The output is a vector of size n containing the association score of each input features.
For more advanced strategies look at [8].

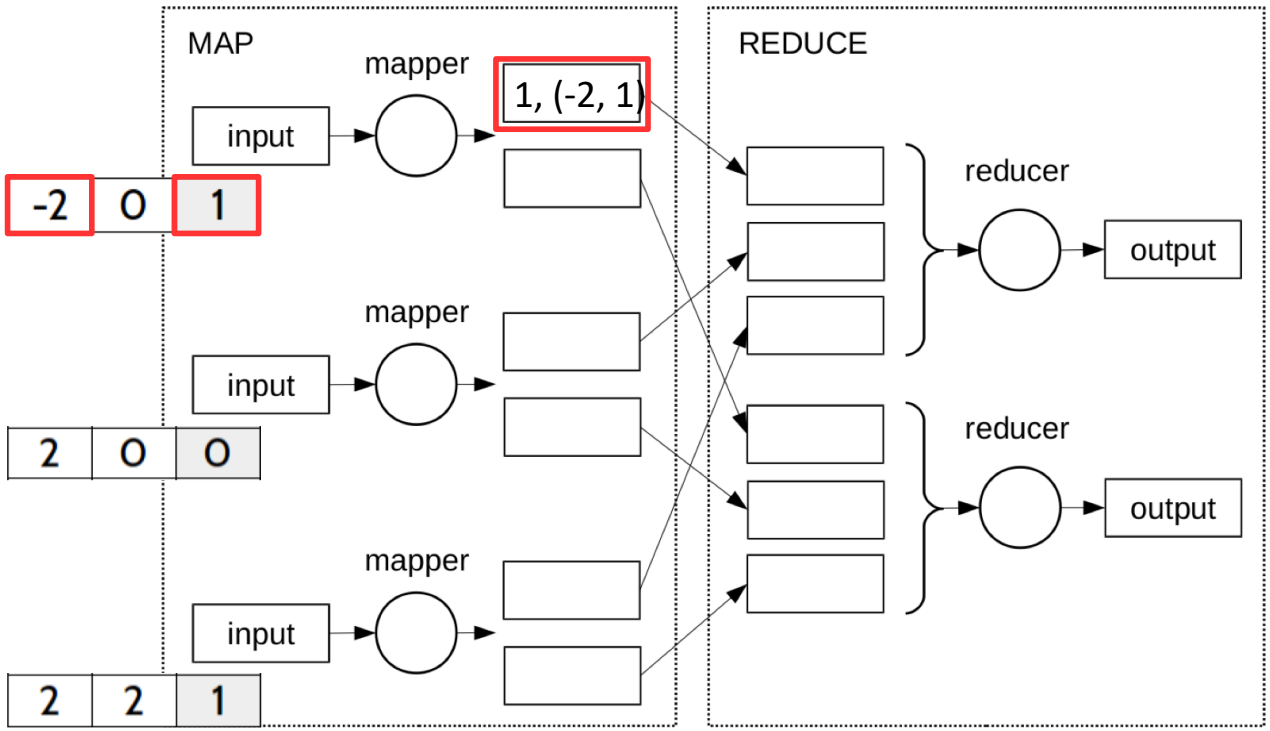
1	*2*	
-2	0	1
2	0	0
2	2	1



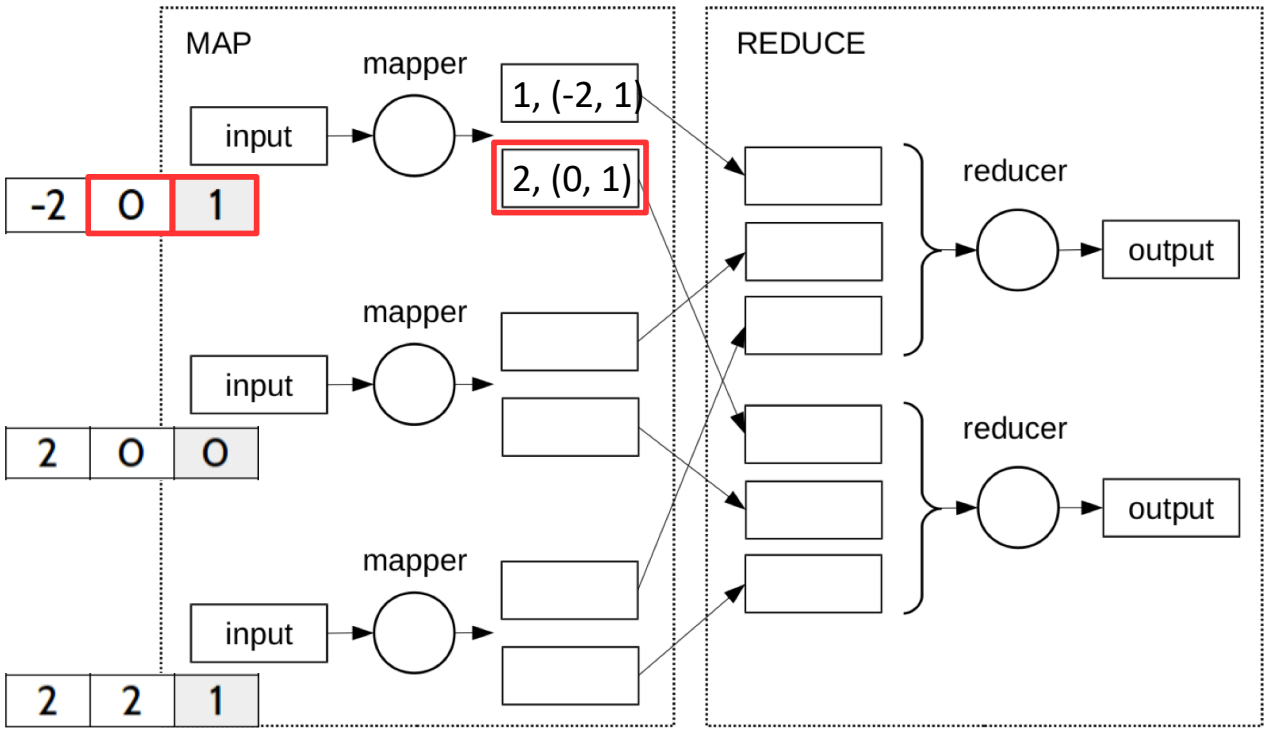
1	2	
-2	0	1
2	0	0
2	2	1

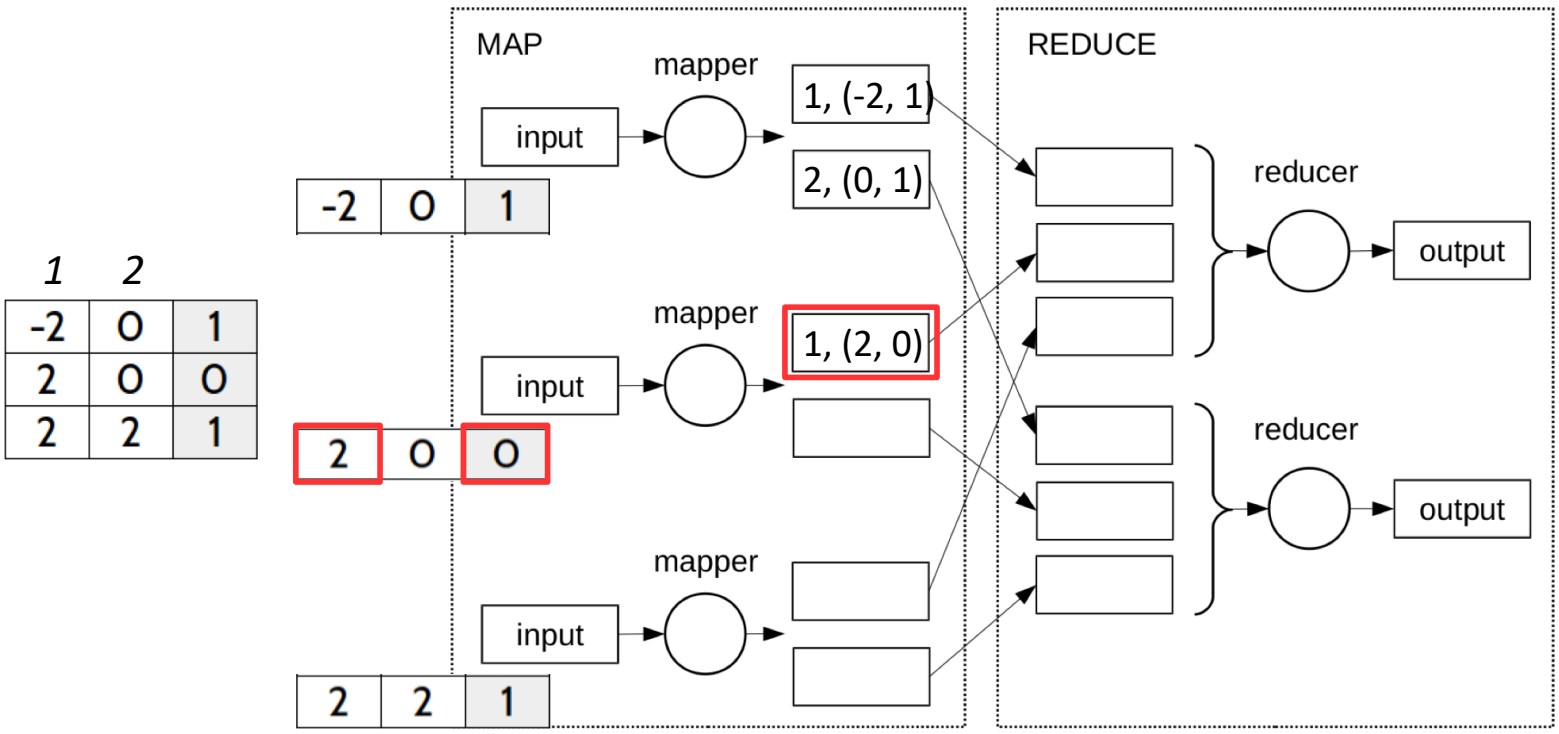


1	2	
-2	0	1
2	0	0
2	2	1

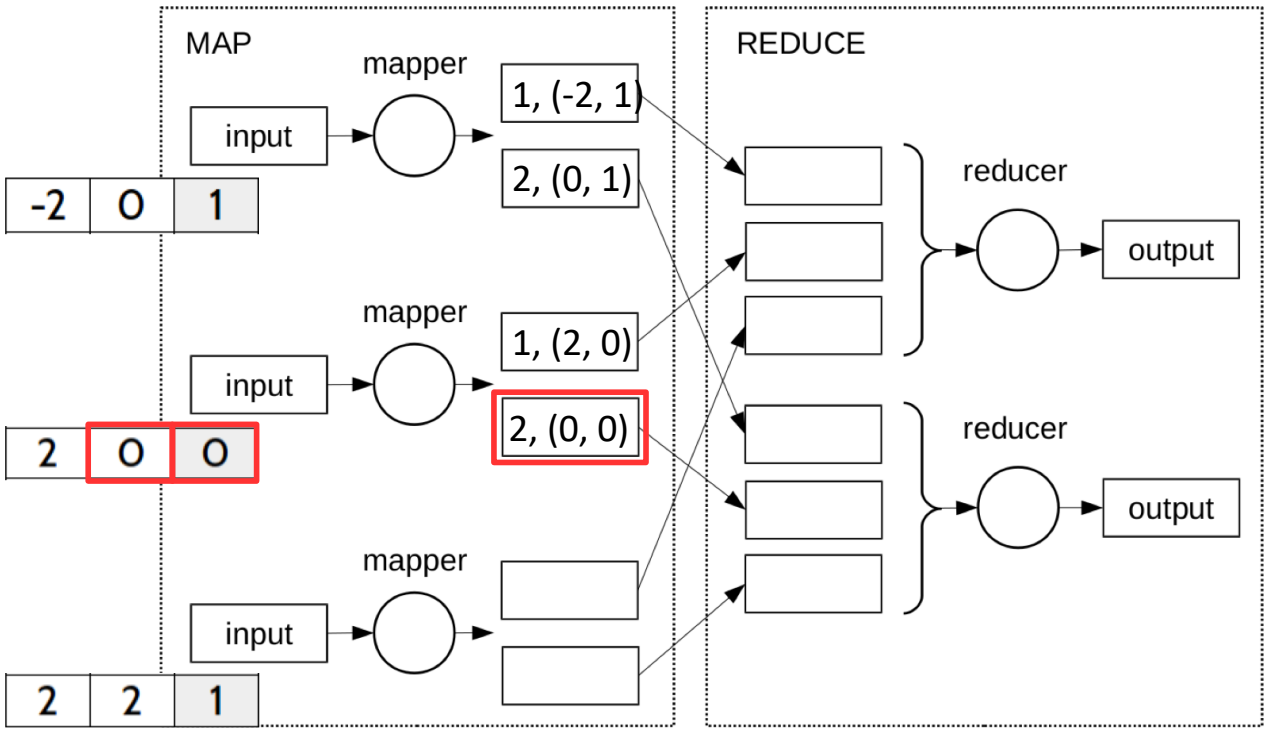


1	2	
-2	0	1
2	0	0
2	2	1

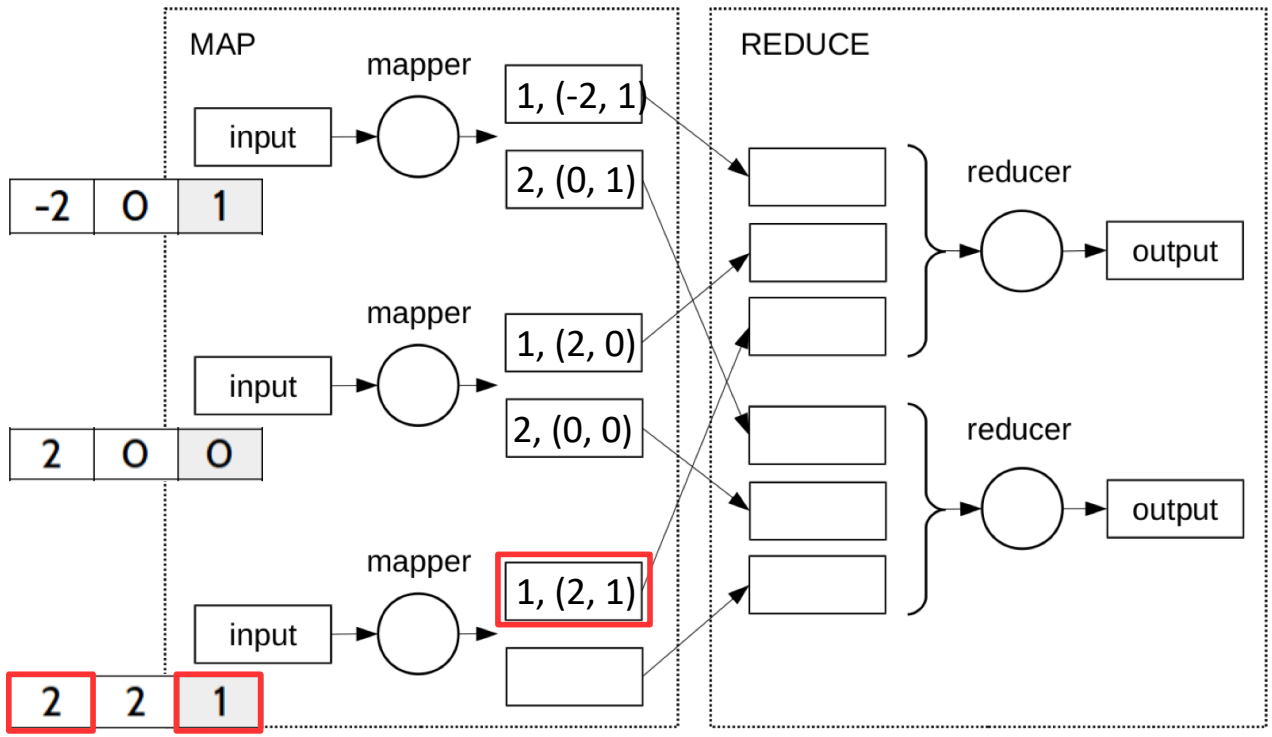



1 2
-2 0 1
2 0 0
2 2 1
MAP
mapper
input
-2 0 1
1, (-2, 1)
2, (0, 1)
mapper
input
2 0 0
1, (2, 0)
mapper
input
2 2 1
REDUCE
reducer
output
reducer
output

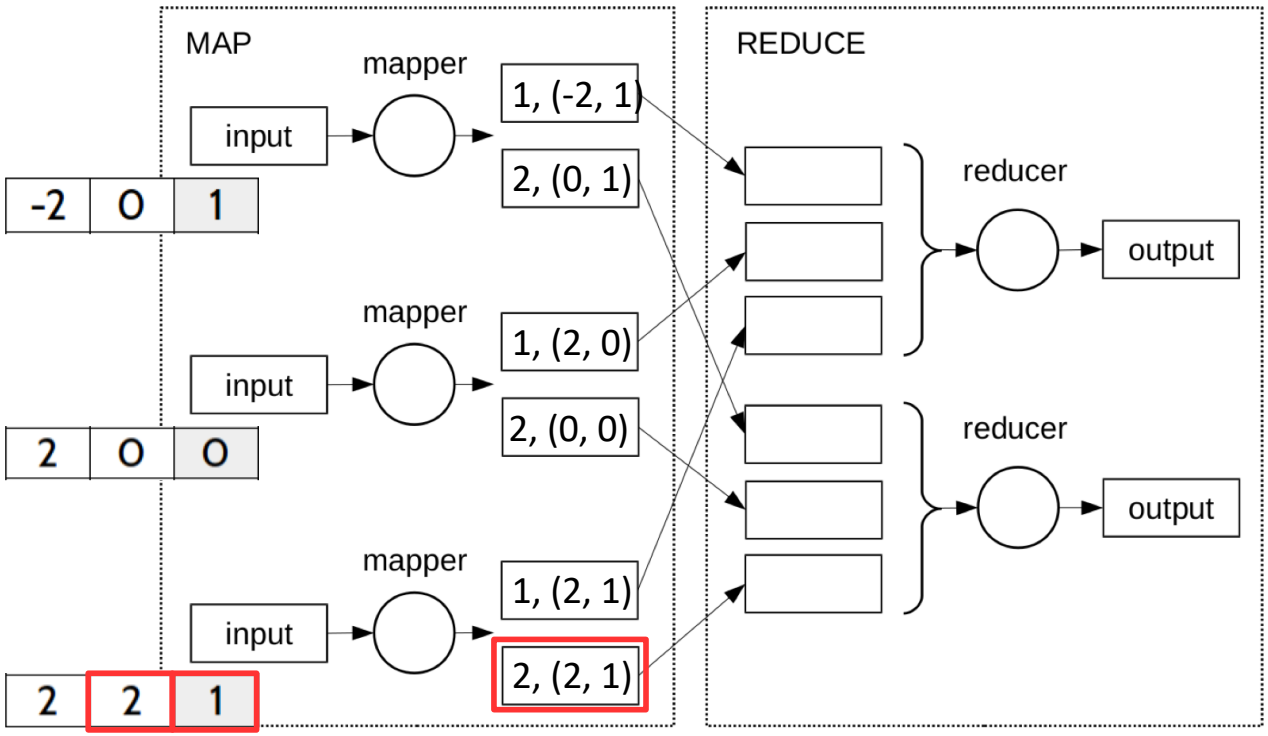
1	2	
-2	0	1
2	0	0
2	2	1



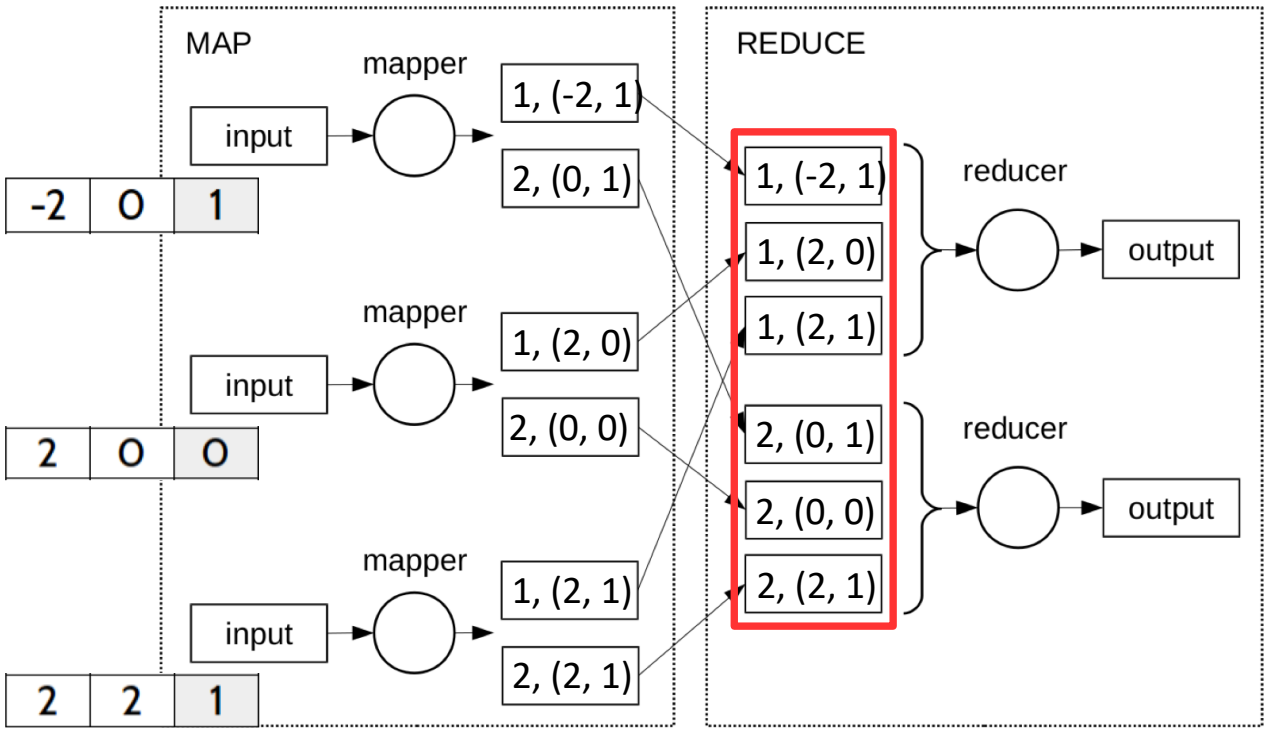
1	2	
-2	0	1
2	0	0
2	2	1

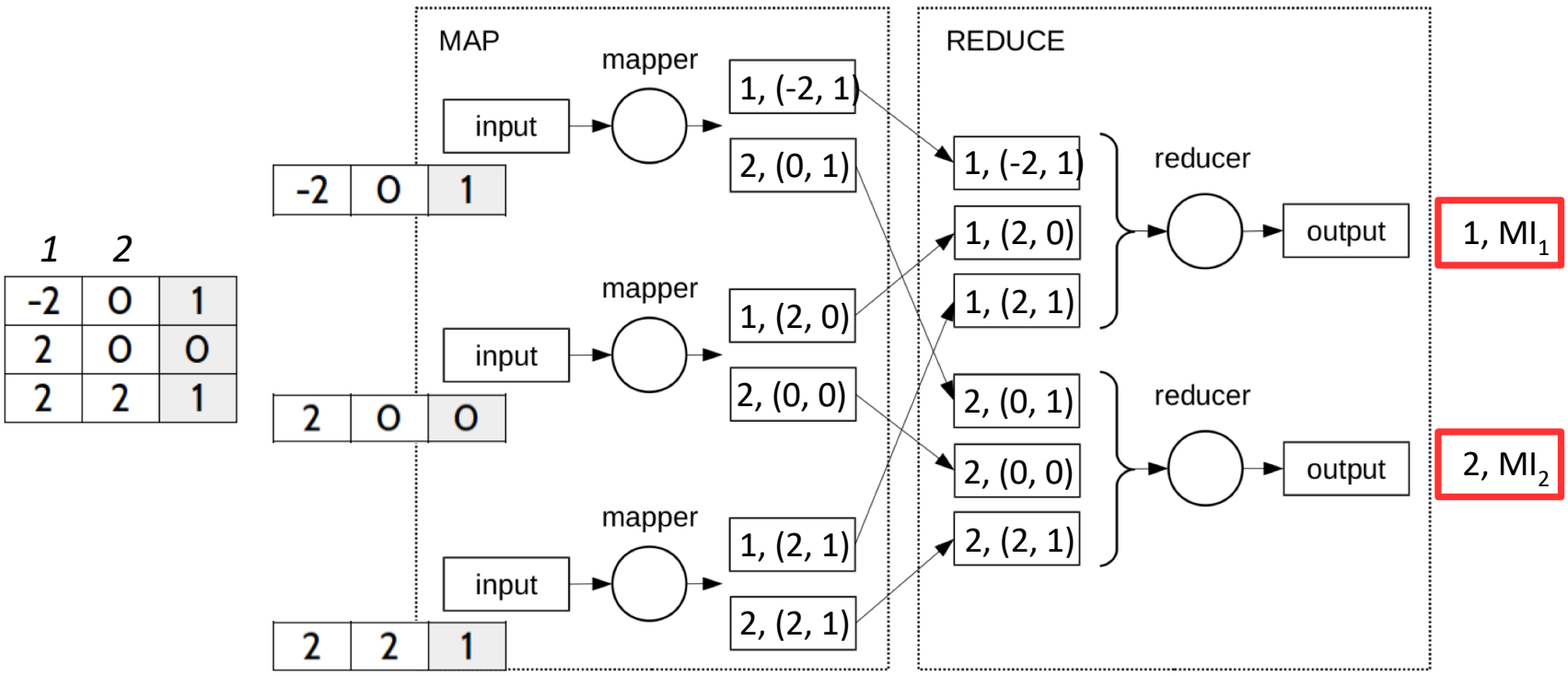


1	2	
-2	0	1
2	0	0
2	2	1



1	2	
-2	0	1
2	0	0
2	2	1

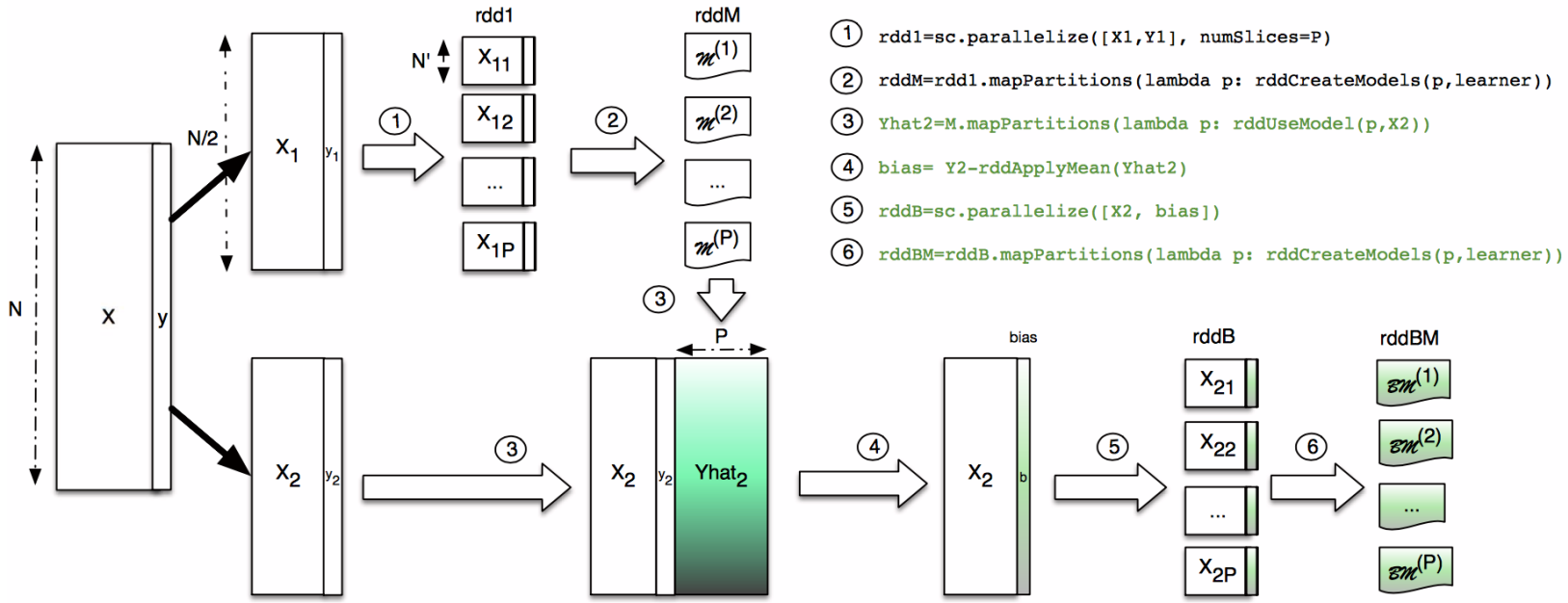



1
2
-2
0
1
2
0
0
2
2
1
MAP
mapper
input
-2
0
1
1, (-2, 1)
2, (0, 1)
mapper
input
2
0
0
1, (2, 0)
2, (0, 0)
mapper
input
2
2
1
1, (2, 1)
2, (2, 1)
REDUCE
1, (-2, 1)
1, (2, 0)
1, (2, 1)
reducer
output
1, MI1
2, (0, 1)
2, (0, 0)
2, (2, 1)
reducer
output
2, MI2

BIAS CORRECTION

- Naive expectation: ”Larger datasets imply better models. ”
- Nevertheless big datasets are no guarantee for optimal modeling since they do not automatically solve the issues of model design, validation and selection.
- Techniques of cross-validation and model assessment are computationally prohibitive when the size of the dataset explodes.
- The benefit of a massive dataset is not only related to the size of the training set but to the possibility of assessing in an accurate and scalable manner the properties of the learner itself (e.g. bias and variance).
- Idea: splitting data for computing a learner and its bias (and consequently correct the bias) is better than using the whole dataset for fitting a model.
- [1] proposes a scalable implementation of a bias correction strategy to improve the accuracy of learning techniques for regression in a big data setting.

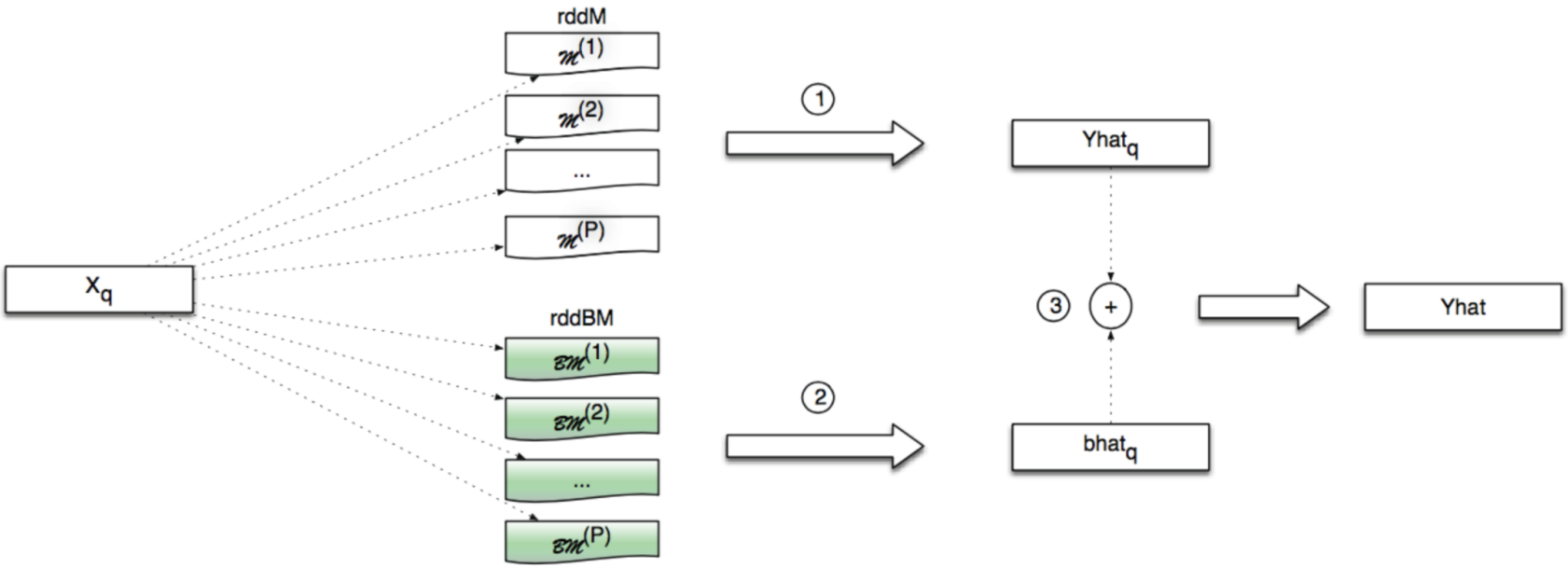
SCALABLE BIAS ESTIMATION



$$\text{Bias}[\hat{\boldsymbol{\theta}}] = \theta - \text{E}_{\text{D}_\text{N}}[\hat{\boldsymbol{\theta}}]$$

From [1]

SCALABLE BIAS CORRECTION




```
1 Yhatq=rddApplyMean(rddM.mapPartitions(lambda p: rddUseModel(p,xq)))
2 bYhatq=rddApplyMean(rddBM.mapPartitions(lambda p: rddUseModel(,xq)))
3 Yhat=Yhatq+bYhatq
```

$$\hat{\boldsymbol{\theta}}_{\text{corr}} = \hat{\boldsymbol{\theta}} + \widehat{\text{Bias}}(\hat{\boldsymbol{\theta}})$$

COMPLEXITY ISSUES

GRAPH MODEL FOR MR PROBLEMS

Let us consider a computing problem characterized by

1. a set of p inputs
2. a set of outputs
3. a many-many relationship linking outputs to inputs

This problem can be solved by a single MR job if for every output of the problem there is at least a reducer that is assigned all the inputs that are related to that output.

REDUCER SIZE AND REPLICATION RATE

For many problems it is possible to define a number of algorithms with different amount of communications. In [7] the analysis of complexity of a MR algorithm discusses the tradeoff of two quantities:

1. Reducer size: the number q of inputs of reducer (fan-in in graph terminology). This quantity is related to the complexity of the reducer and the number of reducers. The larger q, the smaller the number of reducers (and the parallelism) and the higher their complexity.
2. Replication rate r: number of key-value pairs produced by Map tasks divided by the number of inputs. It is equivalent to the average number of reducers using an input (fan-out) . This quantity has an impact on the communication cost.

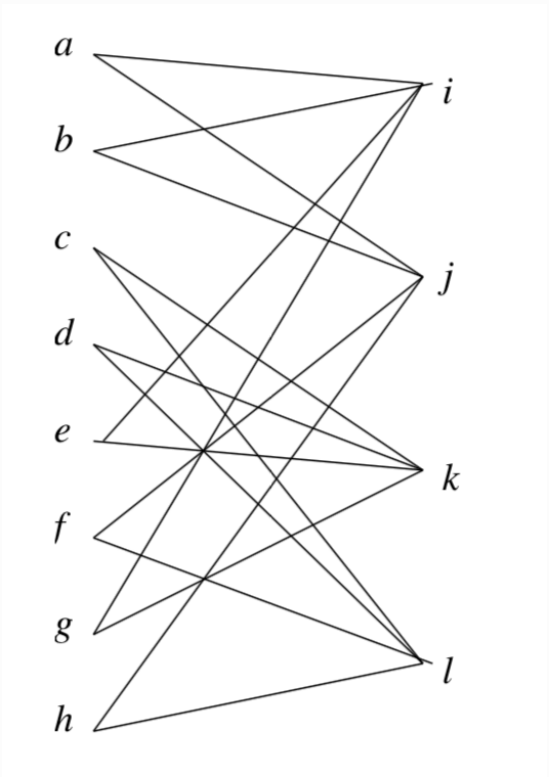
These two quantities are typically inversely proportional, i.e. the higher the complexity of the reducer the lower the communication cost and viceversa. For instance we can decrease the communication cost by grouping inputs.

GRAPH MODEL FOR MR PROBLEMS

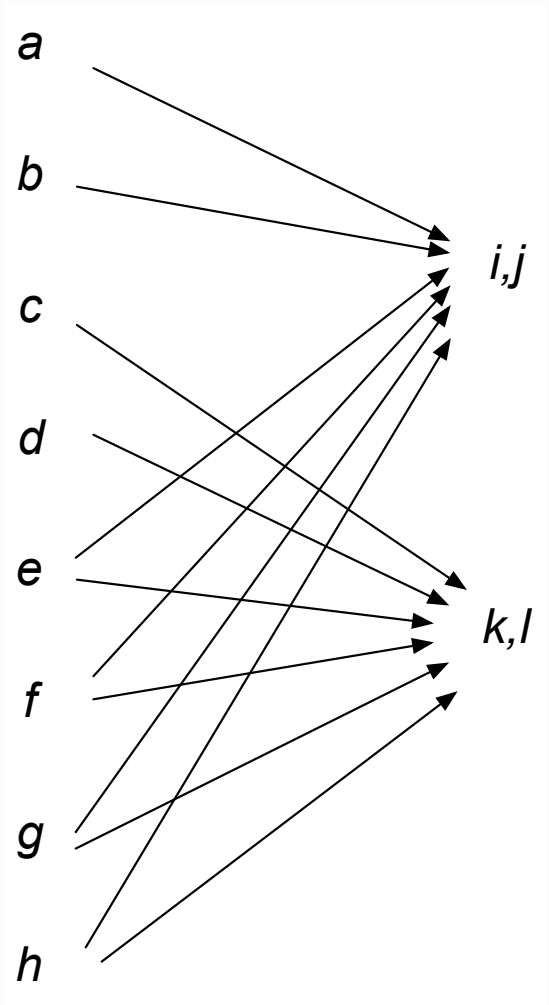
Example:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} = \begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

$p = 2n^2$ inputs, n^2 outputs, each output depends on $q = 2n$ inputs and each input is used by $r = n$ outputs. Note that $qr = p$.



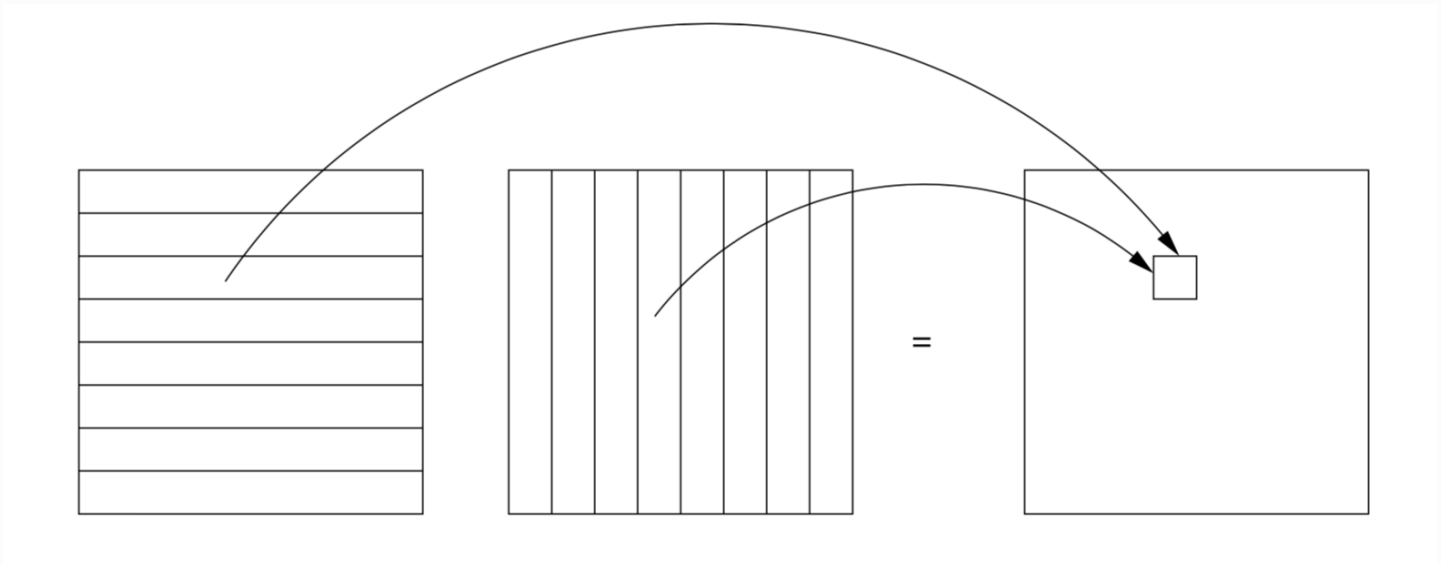
FIRST ALTERNATIVE GRAPH FOR MATRIX MULTIPLICATION



Increased q (from 4 to 6) and reduced r (from 2 to 1.5). Still $qr = 8$.

SECOND ALTERNATIVE FOR MATRIX MULTIPLICATION

Split in g groups. Same single-step MR algorithm (seen before) with the following differences: 1) in Map step the indices i and k refer to group of lines and not to a single lines 2) in Reduce step a submatrix (instead of a single element) is returned



$p = 2n^2$ inputs, $(n/g)^2$ reducers, each reducer depends on
$q = 2n * (n/g) = 2n^2/g$ inputs and each input is used by $r = g$ outputs.
Again $qr = p$.

Lower communication rate at the cost of a higher reducer complexity!

PAIRED ASSESSMENT

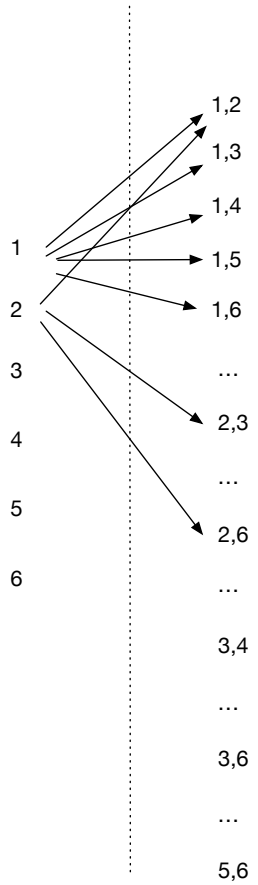
- Suppose we have a very large number p of objects (e.g. images or variables) and we want to compute some statistics related to all pairs (e.g. most similar or most correlated): complexity $O(p^2)$.
- We have p inputs divided in $2 \leq g \leq p$ equal-sized groups and $\binom{p}{2}$ outputs. We have $\binom{g}{2}$ reducers (one for each pair of groups) and each reducer get inputs from two groups, so it will have a number of inputs $q = 2p/g$.
- Each object (belonging to a group) is sent to all $r = g - 1$ reducers containing the original group and one of the other groups. If we make the approximation $r \approx g$ we see that the reducer size

$$q = \frac{2p}{r}$$

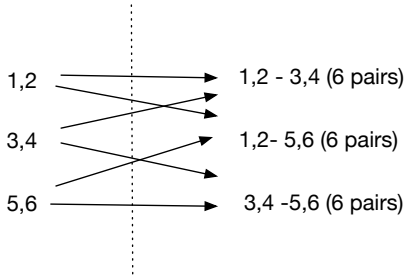
is inversely proportional to the replication

TRADE-OFF COMMUNICATION VS REDUCER COMPLEXITY

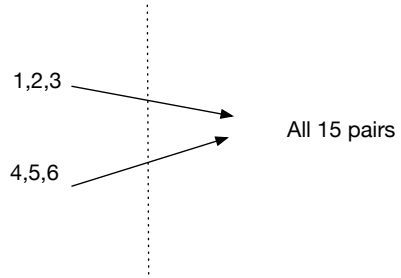
p=6 elements



g=6 groups (one per element)
replication rate r=5,
15 reducers with q=2 inputs



g=3 groups,
replication rate r=2,
3 reducers with q=4 inputs



g=2 groups,
replication rate r=1,
1 reducer with q=6 inputs

q(r+1)=2p


G. Bontempi and Y. A. L. Borgne.
Predictive modeling in a big data distributed setting: A scalable bias correction approach.
In 2016 IEEE International Congress on Big Data (BigData Congress), pages 68–74, June 2016.


Leo Breiman.
Pasting small votes for classification in large databases and on-line.
Machine Learning, 36(1):85–103, Jul 1999.


Tin Kam Ho.
The random subspace method for constructing decision forests.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, Aug 1998.


Gilles Louppe and Pierre Geurts.
Ensembles on random patches.
In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, Machine Learning and Knowledge Discovery in Databases, pages 346–361, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

J. Maillo, I. Triguero, and F. Herrera.
A mapreduce-based k-nearest neighbor approach for big data classification.
In 2015 IEEE Trustcom/BigDataSE/ISPA, volume 2, pages 167–172, Aug 2015.

M. Parsian.
Data Algorithms: Recipes for Scaling Up with Hadoop and Spark.
O'Reilly Media, 2015.

Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman.
Mining Massive Datasets.
2014.

Claudio Reggiani, Yann-Aël Le Borgne, and Gianluca Bontempi.
Feature selection in high-dimensional dataset using mapreduce.
In Bart Verheij and Marco Wiering, editors, Artificial Intelligence, pages 101–115, Cham, 2018. Springer International Publishing.

J. Wang, W. Liu, S. Kumar, and S. F. Chang.
Learning to hash for indexing big data: A survey.

Proceedings of the IEEE, 104(1):34–57, Jan 2016.