

732A54 Big Data Analytics

Lecture 10: Machine Learning with MapReduce

Jose M. Peña

IDA, Linköping University, Sweden

Contents

- ▶ MapReduce Framework
- ▶ Machine Learning with MapReduce
 - ▶ Neural Networks
 - ▶ Support Vector Machines
 - ▶ Mixture Models
 - ▶ *K*-Means
- ▶ Summary

- ▶ Main sources

- ▶ Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107-113, 2008.
- ▶ Chu, C.-T. et al. Map-Reduce for Machine Learning on Multicore. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 281-288, 2006.

- ▶ Additional sources

- ▶ Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.
- ▶ Yahoo tutorial at <https://developer.yahoo.com/hadoop/tutorial/module4.html>.
- ▶ Slides for 732A95 Introduction to Machine Learning.

MapReduce Framework

- ▶ Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of nodes.
- ▶ Easy to use, since the parallelization happens automatically.
- ▶ Easy to speed up by using/adding more nodes to the cluster.
- ▶ Typical uses at Google:
 - ▶ Large-scale machine learning problems, e.g. clustering documents from Google News.
 - ▶ Extracting properties of web pages, e.g. web access log data.
 - ▶ Large-scale graph computations, e.g. web link graph.
 - ▶ Statistical machine translation.
 - ▶ Processing satellite images.
 - ▶ Production of the indexing system used for Google's web search engine.
- ▶ Google replaced it with Cloud Dataflow, since it could not process the amount of data they produce.
- ▶ However, it is still the processing core of Apache Hadoop, another framework for distributed storage and distributed processing of large datasets on computer clusters.
- ▶ Moreover, it is a straightforward way to adapt some machine learning algorithms to cope with big data.
- ▶ Apache Mahout is a project to produce distributed implementations of machine learning algorithms. Many available implementations build on Hadoop's MapReduce. However, such implementations are not longer accepted.

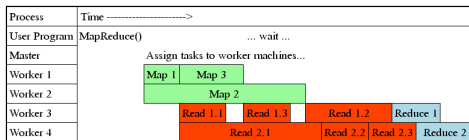
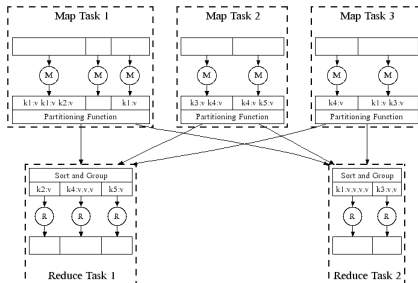
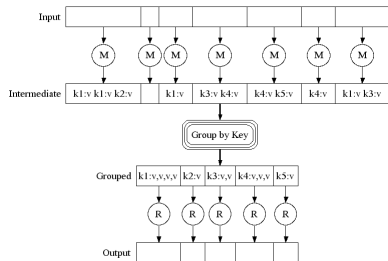
MapReduce Framework

- ▶ The user only has to implement the following two functions:
 - ▶ Map function:
 - ▶ Input: A pair (*in_key*, *in_value*).
 - ▶ Output: A list *list(out_key, intermediate_value)*.
 - ▶ Reduce function:
 - ▶ Input: A pair (*out_key*, *list(intermediate_value)*).
 - ▶ Output: A list *list(out_value)*.
- ▶ All intermediate values associated with the same intermediate key are grouped together before passing them to the reduce function.
- ▶ Example for counting word occurrences in a collection of documents:

```
map(String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:  
        EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

MapReduce Framework



MapReduce Framework

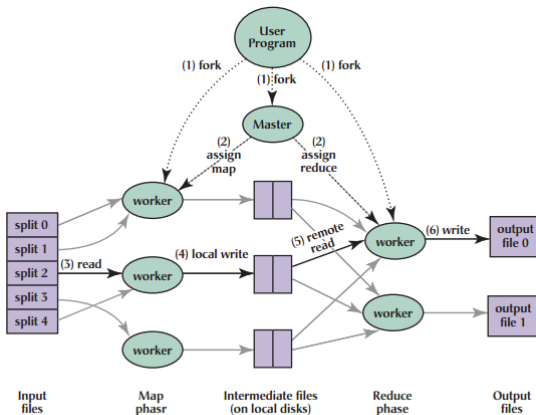


Fig. 1. Execution overview.

1. Split the input file in M pieces and store them on the local disks of the nodes of the cluster. Start up many copies of the user's program on the nodes.
2. One copy (the master) assigns tasks to the rest of the copies (the workers). To reduce communication, it tries to assign map workers to nodes with input data.

MapReduce Framework

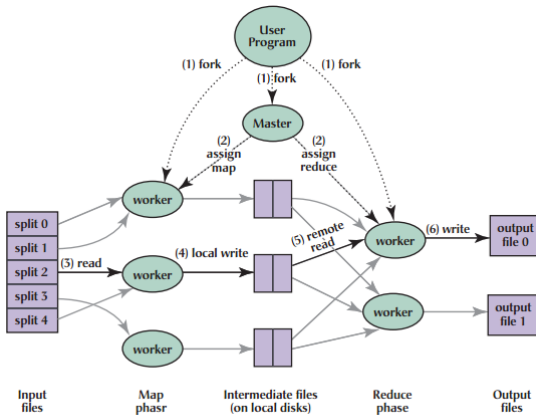


Fig. 1. Execution overview.

- Each map worker processes a piece of input data, by passing each pair key/value to the user's map function. The results are buffered in memory.
- The buffered results are written to **local** disk. The disk is **partitioned** in R pieces. The location of the partitions on disk are passed back to the master so that they can be forwarded to the reduce workers.

MapReduce Framework

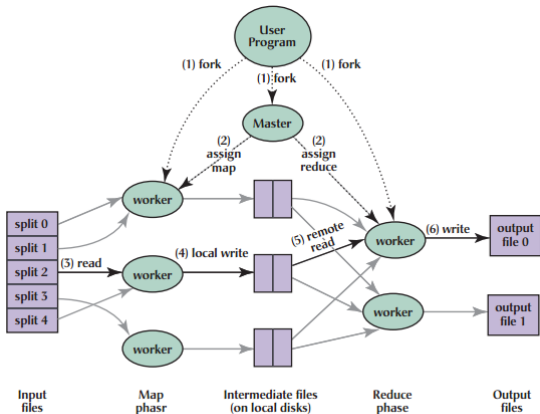
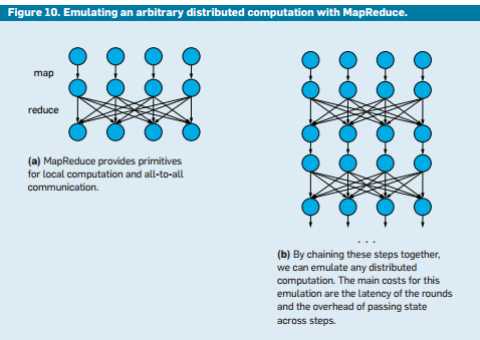


Fig. 1. Execution overview.

5. The reduce worker reads its partition remotely. This implies **shuffle** and sort by key.
6. The reduce worker processes each key using the user's reduce function. The result is written to the **global** file system.
7. The output of a MapReduce call may be the input to another. Note that we have performed M map tasks and R reduce tasks.

MapReduce Framework

- ▶ MapReduce can emulate any distributed computation, since this consists of nodes that perform local computations and occasionally exchange messages.
- ▶ Therefore, any distributed computation can be divided into sequence of MapReduce calls:
 - ▶ First, nodes perform local computations (map), and
 - ▶ then, they exchange messages (reduce).

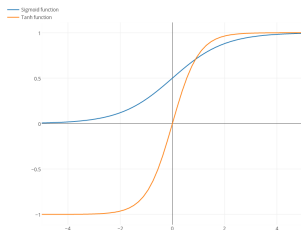
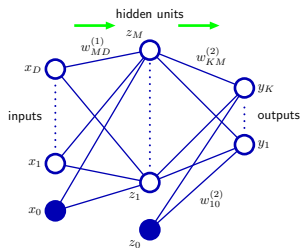


- ▶ However, the emulation may be inefficient since the message exchange relies on external storage, e.g. disk.

MapReduce Framework

- ▶ Fault tolerance:
 - ▶ Necessary since thousands of nodes may be used.
 - ▶ The master pings the workers periodically. No answer means failure.
 - ▶ If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
 - ▶ Note the importance of storing several copies (typically 3) of the input data on different nodes.
 - ▶ If a worker fails then its in-progress **reduce** task is re-executed. The results of its completed reduce tasks are stored on the global file system and, thus, they are accessible.
 - ▶ To be able to recover from the unlikely event of a master failure, the master periodically saves the state of the different tasks (idle, in-progress, completed) and the identify of the worker for the non-idle tasks.
- ▶ Task granularity:
 - ▶ M and R are larger than the number of nodes available.
 - ▶ Large M and R values benefit dynamic load balance and fast failure recovery.
 - ▶ Too large values may imply too many scheduling decisions, and too many output files.
 - ▶ For instance, $M = 200000$ and $R = 5000$ for 2000 available nodes.

Machine Learning with MapReduce: Neural Networks



- ▶ Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- ▶ Hidden units and activation function: $z_j = h(a_j)$
- ▶ Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- ▶ Output activation function for regression: $y_k(\mathbf{x}) = a_k$
- ▶ Output activation function for classification: $y_k(\mathbf{x}) = \sigma(a_k)$
- ▶ Sigmoid function: $\sigma(a) = \frac{1}{1 + \exp(-a)}$
- ▶ Two-layer NN:

$$y_k(\mathbf{x}) = \sigma\left(\sum_j w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- ▶ Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- ▶ All the previous is, of course, generalizable to more layers.

Machine Learning with MapReduce: Neural Networks

- ▶ Consider regressing an K -dimensional continuous random variable on a D -dimensional continuous random variable.
- ▶ Consider a training set $\{(\mathbf{x}_n, \mathbf{t}_n)\}$ of size N . Consider minimizing the error function

$$E(\mathbf{w}^t) = \sum_n E_n(\mathbf{w}^t) = \sum_n \frac{1}{2} (\mathbf{y}(\mathbf{x}_n) - \mathbf{t}_n)^2$$

- ▶ The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- ▶ Batch gradient descent

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla E(\mathbf{w}^t)$$

where $\eta > 0$ is the learning rate, and $\nabla E(\mathbf{w}^t)$ can be computed efficiently thanks to the backpropagation algorithm.

- ▶ Sequential gradient descent

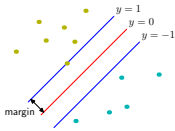
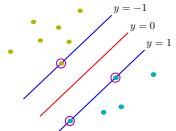
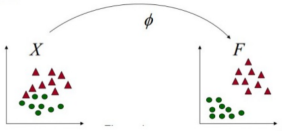
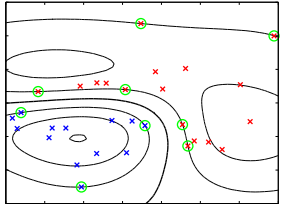
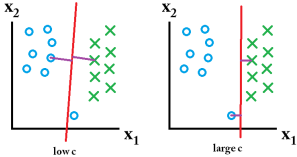
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla E_n(\mathbf{w}^t)$$

where n is chosen randomly or sequentially.

Machine Learning with MapReduce: Neural Networks

- ▶ Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.
- ▶ Unfortunately, sequential gradient descent cannot be casted into MapReduce terms: Each iteration must wait until the previous iterations are done.
- ▶ However, **each iteration** of batch gradient descent can easily be casted into MapReduce terms:
 - ▶ Map function: Compute the gradient for the samples in the piece of input data. Note that this implies forward and backward propagation.
 - ▶ Reduce function: Sum the partial gradients and update \mathbf{w} accordingly.
- ▶ Note that $1 \leq M \leq n$, whereas $R = 1$.

Machine Learning with MapReduce: Support Vector Machines

Margin	Largest margin
	
Feature space (kernel trick)	Input space
	
Trading errors for margin	
	

Machine Learning with MapReduce: Support Vector Machines

- Consider binary classification with input space \mathbb{R}^D . Consider a training set $\{(\mathbf{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$. Consider using the linear model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

so that a new point \mathbf{x} is classified according to the sign of $y(\mathbf{x})$.

- The optimal separating hyperplane is given by

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n$$

subject to $t_n y(\mathbf{x}_n) \geq 1 - \xi_n$ and $\xi_n \geq 0$ for all n , where

$$\xi_n = \begin{cases} 0 & \text{if } t_n y(\mathbf{x}_n) \geq 1 \\ |t_n - y(\mathbf{x}_n)| & \text{otherwise} \end{cases}$$

are slack variables to penalize (almost-)misclassified points.

- We usually work with the dual representation of the problem, in which we maximize

$$\sum_n a_n - \frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to $a_n \geq 0$ and $a_n \leq C$ for all n .

- The reason is that the dual representation makes use of the kernel trick, i.e. it allows working in a more convenient feature space without constructing it.

Machine Learning with MapReduce: Support Vector Machines

- Assume that we are interested in linear support vector machines, i.e $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \mathbf{w}^T \mathbf{x} + b$. Then, we can work directly with the primal representation.
- We can even consider a quadratic penalty for (almost-)misclassified points, in which case the optimal separating hyperplane is given by

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \xi_n^2 = \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n \in E} (\mathbf{w}^T \mathbf{x}_n - t_n)^2$$

where $n \in E$ if and only if $t_n y(\mathbf{x}_n) < 1$.

- Note that the previous expression is a quadratic function with linear inequality constraints and, thus, it is concave (up) and, thus, "easy" to minimize.
- For instance, we can use again batch gradient descent. The gradient is now given by

$$\mathbf{w} + 2C \sum_{n \in E} (\mathbf{w}^T \mathbf{x}_n - t_n) \mathbf{x}_n$$

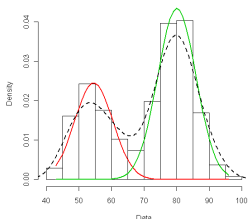
- Again, **each iteration** of batch gradient descent can easily be casted into MapReduce terms:
 - Map function: Compute the gradient for the samples in the piece of input data.
 - Reduce function: Sum the partial gradients and update \mathbf{w} accordingly.
- Note that $1 \leq M \leq n$, whereas $R = 1$.

Machine Learning with MapReduce: Mixture Models

- Sometimes the data do not follow any known probability distribution but a mixture of known distributions such as

$$p(\mathbf{x}) = \sum_{k=1}^K p(k)p(\mathbf{x}|k)$$

where $p(\mathbf{x}|k)$ are called mixture components, and $p(k)$ are called mixing coefficients, usually denoted by π_k .



- Mixture of multivariate Gaussian distributions:

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \text{ and } \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}$$

Machine Learning with MapReduce: Mixture Models

- Mixture of multivariate Bernoulli distributions:

$$p(\mathbf{x}) = \sum_k \pi_k \text{Bern}(\mathbf{x}|\boldsymbol{\mu}_k)$$

where

$$\text{Bern}(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_i \mu_{ki}^{x_i} (1 - \mu_{ki})^{(1-x_i)}$$



Figure 9.10 Illustration of the Bernoulli mixture model in which the top row shows examples from the digits data set after converting the pixel values from grey scale to binary using a threshold of 0.5. On the bottom row the first three images show the parameters μ_{ki} for each of the three components in the mixture model. As a comparison, we also fit the same data set using a single multivariate Bernoulli distribution, again using maximum likelihood. This amounts to simply averaging the counts in each pixel and is shown by the right-most image on the bottom row.

Machine Learning with MapReduce: Mixture Models

- Given a sample $\{\mathbf{x}_n\}$ of size N from a mixture of multivariate Bernoulli distributions, the expected log likelihood function is maximized when

$$\begin{aligned}\pi_k^{ML} &= \frac{\sum_n p(z_{nk}|\mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{N} \\ \mu_{ki}^{ML} &= \frac{\sum_n x_{ni} p(z_{nk}|\mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}{\sum_n p(z_{nk}|\mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})}\end{aligned}$$

where \mathbf{z}_n is a K -dimensional binary vector indicating (fractional) component memberships.

- This is not a closed form solution, but it suggests the following algorithm.

EM algorithm

Set $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ to some initial values

Repeat until $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ do not change

 Compute $p(z_{nk}|\mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi})$ for all n /* E step */

 Set π_k to π_k^{ML} , and μ_{ki} to μ_{ki}^{ML} for all k and i /* M step */

Machine Learning with MapReduce: Mixture Models

- ▶ **Each iteration** of the EM algorithm can easily be casted into MapReduce terms:

- ▶ Map function: Compute

$$\sum_{n \in PID} p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) \quad (1)$$

and

$$\sum_{n \in PID} x_{ni} p(z_{nk} | \mathbf{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}) \quad (2)$$

where PID is the piece of input data.

- ▶ Reduce function: Sum up the results (1) of the map tasks and divide it by N . Sum up the results (2) of the map tasks and divide it by the sum of the results (1).
- ▶ Note that $1 \leq M \leq N$, whereas $R = 1$.

Machine Learning with MapReduce: K -Means Algorithm

- Recall that

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

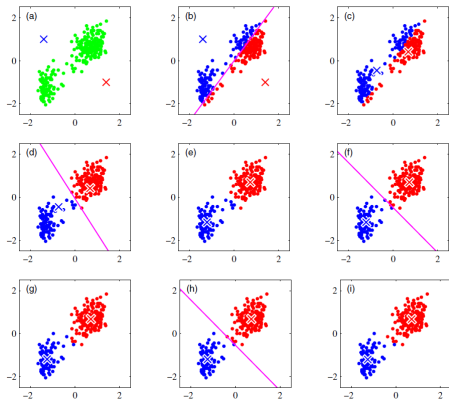
- Assume that $\boldsymbol{\Sigma}_k = \epsilon \mathbf{I}$ where ϵ is a variance parameter and \mathbf{I} is the identity matrix. Then,

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp\left(-\frac{1}{2\epsilon}\|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$$

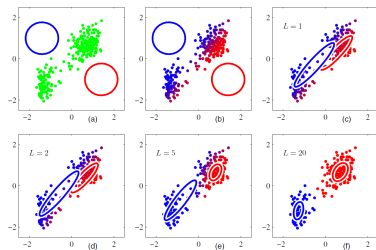
$$p(k|\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\pi}) = \frac{\pi_k \exp\left(-\frac{1}{2\epsilon}\|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)}{\sum_k \pi_k \exp\left(-\frac{1}{2\epsilon}\|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)}$$

- As $\epsilon \rightarrow 0$, the smaller $\|\mathbf{x} - \boldsymbol{\mu}_k\|^2$ the slower $\exp\left(-\frac{1}{2\epsilon}\|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$ goes to 0.
- As $\epsilon \rightarrow 0$, individuals are hard-assigned (i.e. with probability 1) to the population with closest mean. This clustering technique is known as K -means algorithm.
- Note that $\boldsymbol{\pi}$ and $\boldsymbol{\Sigma}$ play no role in the K -means algorithm whereas, in each iteration, $\boldsymbol{\mu}$ is updated to the average of the individuals assigned to population k .

Machine Learning with MapReduce: K -Means Algorithm



K -means algorithm



EM algorithm

Machine Learning with MapReduce: K -Means Algorithm

- ▶ **Each iteration** of the K -means algorithm can easily be casted into MapReduce terms:
 - ▶ Map function: Choose the population with the closest mean for each sample in the piece of input data.
 - ▶ Reduce function: Recalculate the population means from the results of the map tasks.
- ▶ Note that $1 \leq M \leq N$, whereas $R = 1$.

Machine Learning with MapReduce

Data Sets	samples (m)	features (n)
Adult	30162	14
Helicopter Control	44170	21
Corel Image Features	68040	32
IPUMS Census	88443	61
Synthetic Time Series	100001	10
Census Income	199523	40
ACIP Sensor	229564	8
KDD Cup 99	494021	41
Forest Cover Type	581012	55
1990 US Census	2458285	68

	lwr	gda	nb	logistic	pca	ica	svm	nn	kmeans	em
Adult	1.922	1.801	1.844	1.962	1.809	1.857	1.643	1.825	1.947	1.854
Helicopter	1.93	2.155	1.924	1.92	1.791	1.856	1.744	1.847	1.857	1.86
Corel Image	1.96	1.876	2.002	1.929	1.97	1.936	1.754	2.018	1.921	1.832
IPUMS	1.963	2.23	1.965	1.938	1.965	2.025	1.799	1.974	1.957	1.984
Synthetic	1.909	1.964	1.972	1.92	1.842	1.907	1.76	1.902	1.888	1.804
Census Income	1.975	2.179	1.967	1.941	2.019	1.941	1.88	1.896	1.961	1.99
Sensor	1.927	1.853	2.01	1.913	1.955	1.893	1.803	1.914	1.953	1.949
KDD	1.969	2.216	1.848	1.927	2.012	1.998	1.946	1.899	1.973	1.979
Cover Type	1.961	2.232	1.951	1.935	2.007	2.029	1.906	1.887	1.963	1.991
Census	2.327	2.292	2.008	1.906	1.997	2.001	1.959	1.883	1.946	1.977
avg.	1.985	2.080	1.950	1.930	1.937	1.944	1.819	1.905	1.937	1.922

Table 3: Speedups achieved on a dual core processor, without load time. Numbers reported are dual-core time / single-core time. Super linear speedup sometimes occurs due to a reduction in processor idle time with multiple threads.

Machine Learning with MapReduce

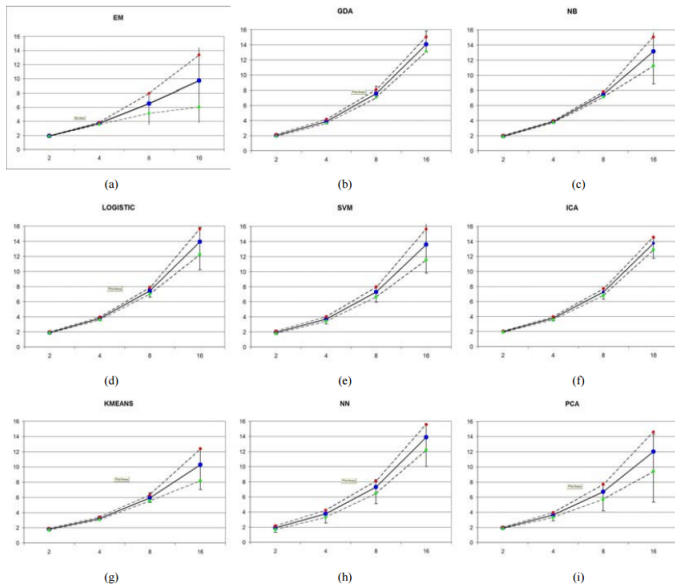


Figure 2: (a)-(i) show the speedup from 1 to 16 processors of all the algorithms over all the data sets. The Bold line is the average, error bars are the max and min speedups and the dashed lines are the variance.

Summary

- ▶ MapReduce is a framework to process large datasets by parallelizing computations.
- ▶ The user only has to specify the map and reduce functions, and parallelization happens automatically.
- ▶ Many machine learning algorithms (e.g. SVMs, NNs, MMs, K -means) can easily be reformulated in terms of such functions.
- ▶ This does not apply for algorithms based on sequential gradient descent.
- ▶ Moreover, MapReduce is inefficient for iterative tasks on the same dataset: Each iteration is a MapReduce call that loads the data anew from disk.
- ▶ Such iterative tasks are common in many machine learning algorithms, e.g. gradient descent, backpropagation algorithm, EM algorithm, K -means.
- ▶ Solution: Spark framework, in the next lecture.