Large-scale Machine Learning in Distributed Environments

Chih-Jen Lin

Talk at K. U. Leuven Optimization in Engineering Center,

January 16, 2013

Outline

1. Why distributed machine learning?
2. Distributed classification algorithms
   - Kernel support vector machines
   - Linear support vector machines
   - Parallel tree learning
3. Distributed clustering algorithms
   - *k*-means
   - Spectral clustering
   - Topic models
4. Discussion and conclusions

Outline

Why Distributed Machine Learning

- The usual answer is that data are too big to be stored in one computer
- Some say that because "Hadoop" and "MapReduce are buzzwords

  No, we should never believe buzzwords
- I will argue that things are more complicated than we thought

- In this talk I will consider only machine learning in data-center environments

  That is, clusters using regular PCs
- I will not discuss machine learning in other parallel environments:

  GPU, multi-core, specialized clusters such as supercomputers

Let's Start with An Example

- Using a linear classifier LIBLINEAR (Fan et al., 2008) to train the rcv1 document data sets (Lewis et al., 2004).
- # instances: 677,399, # features: 47,236
- On a typical PC

  ```
  $time ./train rcv1_test.binary
  ```
- Total time: 50.88 seconds

  Loading time: 43.51 seconds
- For this example

  loading time $\gg$ running time

Loading Time Versus Running Time I

- To see why this happens, let's discuss the complexity
- Assume the memory hierarchy contains only disk and number of instances is l
- Loading time: $l \times$ (a big constant)

  Running time: $l^q \times$ (some constant), where $q \geq 1$.
- Running time is often larger than loading because $q > 1$ (e.g., $q = 2$ or 3)

  Example: kernel methods

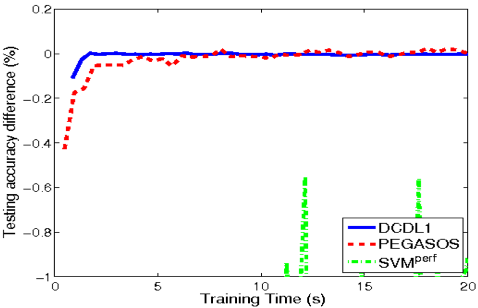
Loading Time Versus Running Time II

- Therefore,

$$l^{q-1} > \text{ a big constant}$$

  and traditionally machine learning and data mining papers consider only running time
- For example, in this ICML 2008 paper (Hsieh et al., 2008), some training algorithms were compared for rcv1

Loading Time Versus Running Time III



- DCDL1 is what LIBLINEAR used
- We see that in 2 seconds, final testing accuracy is achieved

Loading Time Versus Running Time IV

- But as we said, this 2-second running time is misleading
- So what happened? Didn't you say that

$$l^{q-1} > \text{ a big constant?}$$

- The reason is that when l is large, we usually can afford using only $q = 1$ (i.e., linear algorithm)
- For some problems (in particular, documents), going through data a few times is often enough
- Now we see different situations

Loading Time Versus Running Time V

- If running time dominates, then we should design algorithms to reduce number of operations

- If loading time dominates, then we should design algorithms to reduce number of data accesses

- Distributed environment is another layer of memory hierarchy

  So things become even more complicated

Data in a Distributed Environment

- One apparent reason of using distributed systems is that data are too large for one disk
- But in addition to that, what are other reasons of using distributed environments?
- On the other hand, now disk is large. If you have several TB data, should we use one or several machines?
- We will try to answer this question in the following slides

Possible Advantages of Distributed Data Classification

- Parallel data loading

  Reading several TB data from disk $\Rightarrow$ a few hours

  Using 100 machines, each has $1/100$ data in its local disk $\Rightarrow$ a few minutes
- Fault tolerance

  Some data replicated across machines: if one fails, others are still available

But how to efficiently/effectively achieve these is a challenge

An Introduction of Distributed Systems I

Distributed file systems

- We need it because a file is now managed at different nodes
- A file split to chunks and each chunk is replicated
  $\Rightarrow$ if some nodes fail, data still available
- Example: GFS (Google file system), HDFS (Hadoop file system)

Parallel programming frameworks

- A framework is like a language or a specification. You can then have different implementations

An Introduction of Distributed Systems II

- Example:
  MPI (Snir and Otto, 1998): a parallel programming framework
  MPICH2 (Gropp et al., 1999): an implementation
- Sample MPI functions

MPI_Bcast:	Broadcasts to all processes.
MPI_AllGather:	Gathers the data contributed by each process on all processes.
MPI_Reduce:	A global reduction (e.g., sum) to the specified root.
MPI_AllReduce:	A global reduction and sending result to all processes.

An Introduction of Distributed Systems III

They are reasonable functions that we can think about

- MapReduce (Dean and Ghemawat, 2008). A framework now commonly used for large-scale data processing
- In MapReduce, every element is a (key, value) pair

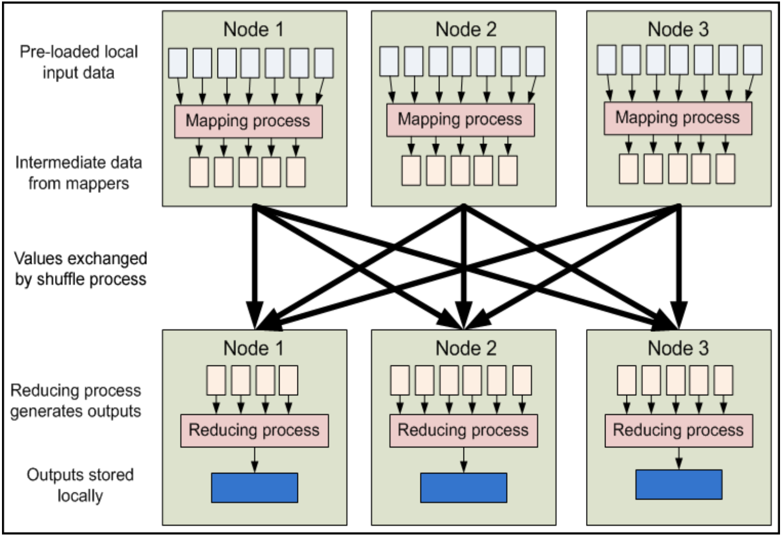
  Mapper: a list of data elements provided. Each element transformed to an output element

  Reducer: values with same key presented to a single reducer

An Introduction of Distributed Systems IV

- See the following illustration from Hadoop Tutorial `http://developer.yahoo.com/hadoop/tutorial`

An Introduction of Distributed Systems V



An Introduction of Distributed Systems VI

- Let's compare MPI and MapReduce
- MPI: communication explicitly specified

  MapReduce: communication performed implicitly

  In a sense, MPI is like an assembly language, but MapReduce is high-level
- MPI: sends/receives data to/from a node's memory

  MapReduce: communication involves expensive disk I/O
- MPI: no fault tolerance

  MapReduce: support fault tolerance

An Introduction of Distributed Systems VII

- Because of disk I/O, MapReduce can be inefficient for iterative algorithms

  To remedy this, some modifications have been proposed
- Example: Spark (Zaharia et al., 2010) supports
  - MapReduce and fault tolerance
  - Cache data in memory between iterations
- MapReduce is a framework; it can have different implementations

An Introduction of Distributed Systems VIII

For example, shared memory (Talbot et al., 2011) and distributed clusters (Google's and Hadoop)

- An algorithm implementable by a parallel framework

$\neq$

You can easily have efficient implementations

- The paper (Chu et al., 2007) has the following title

  Map-Reduce for Machine Learning on Multicore

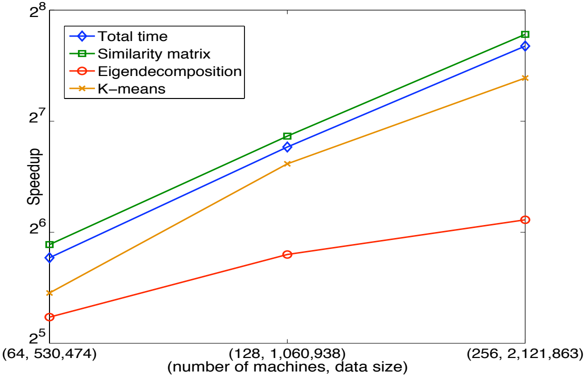
- The authors show that many machine learning algorithms can be implemented by MapReduce

An Introduction of Distributed Systems IX

- These algorithms include linear regression, k-means, logistic regression, naive Bayes, SVM, ICA, PCA, EM, Neural networks, etc
- But their implementations are on shared-memory machines; see the word "multicore" in their title
- Many wrongly think that their paper implies that these methods can be efficiently implemented in a distributed environment. But this is wrong

Evaluation I

- Traditionally a parallel program is evaluated by scalability



Evaluation II

- We hope that when (machines, data size) doubled, the speedup also doubled.

  64 machines, 500k data $\Rightarrow$ ideal speedup is 64

  128 machines, 1M data $\Rightarrow$ ideal speedup is 128
- That is, a linear relationship in the above figure
- But in some situations we can simply check throughput.

  For example, # documents per hour.

Data Locality I

- Transferring data across networks is slow.
- We should try to access data from local disk
- Hadoop tries to move computation to the data.

  If data in node A, try to use node A for computation
- But most machine learning algorithms are not designed to achieve good data locality.
- Traditional parallel machine learning algorithms distribute computation to nodes

  This works well in dedicated parallel machines with fast communication among nodes

Data Locality II

- But in data-center environments this may not work
  $\Rightarrow$ communication cost is very high
- Example: in Chen et al. (2011), for sparse matrix-vector products (size: 2 million)

#nodes	Computation	Communication	Synchronization
16	3,351	2,287	667
32	1,643	2,389	485
64	913	2,645	404
128	496	2,962	428
256	298	3,381	362

- This is by MPI. If using MapReduce, the situation will be worse

Data Locality III

- Another issue is whether users should be allowed to explicitly control the locality

- Now go back to machine learning algorithms
- Two major types of machine learning methods are classification and clustering
- I will discuss more on classification

How People Train Large-scale Data Now?

Two approaches

- Subsample data to one machine and run a traditional algorithm
- Run a distributed classification algorithm

I will discuss advantages and disadvantages of each approach

Training a Subset

- No matter how large the data set is, one can always consider a subset fitting into one computer
- Because subsampling may not downgrade the performance, very sophisticated training methods for small sets have been developed

Training a Subset: Advantages

- It is easier to play with advanced methods on one computer

  Many training data + a so so method

  may not be better than

  Some training data + an advanced method
- Also machines with large RAM (e.g., 1G) are now easily available

Training a Subset: Disadvantage

- Subsampling may not be an easy task

  What if this part is more complicated than training?
- It's not convenient if features are calculated using raw data in distributed environments

  We may need to copy data to the single machine several times (see an example later)
- The whole procedure becomes disconnected and ad hoc

  You switch between distributed systems and regular systems

Using Distributed Algorithms: Disadvantages

- It's difficult to design and implement a distributed algorithm
- Communication and data loading are expensive
- Scheduling a distributed task is sometimes an issue

Using Distributed Algorithms: Advantages

- Integration with other parts of data management
- Can use larger training sets

So Which Approach Should We Take?

- It depends
- Let me try a few examples to illustrate this point

Example: A Multi-class Classification Problem I

- Once I need to train some documents at an Internet company
- From log in data centers we select documents of a time period to one machine
- For each document we generate a feature vector using words in the document (e.g., bigram)
- Data set can fit into one machine ($\geq$ 50G RAM)
- It is easier to run various experiments (e.g., feature engineering) on one machine

Example: A Multi-class Classification Problem II

- So for this application, reducing data to one machine may be more suitable

Example: Features Calculated on Cloud I

- Once I need to train some regression problems
- Features include: "in a time period, the average number of something"
- Values of using a 3-month period differ from those of using a 6-month period
- We need engineers dedicated to extract these features and then copy files to a single machine
- We must maintain two lists of files in distributed and regular file systems

  1. In data center, files of using 3-, 6-, 9-month averages, etc.

Example: Features Calculated on Cloud II

2. In a single computer, subsets of the bigger files

- In this case, running everything in the distributed environment may be more suitable

Resources of Distributes Machine Learning I

- There are many books about Hadoop and MapReduce. I don't list them here.
- For things related to machine learning, a collection of recent works is in the following book

  Scaling Up Machine Learning, edited by Bekkerman, Bilenko, and John Langford, 2011.
- This book covers materials using various parallel environments. Many of them use distributed clusters.

Resources of Distributes Machine Learning II

- Existing tools

  1. Apache Mahout, a machine learning library on Hadoop (http://mahout.apache.org/)

  2. Graphlab (graphlab.org/), a large-scale machine learning library on graph data. Tools include graphical models, clustering, and collaorative filtering

- Subsequently I will show some existing distributed machine learning works
- I won't go through all of them, but these slides can be references for you

Outline

Outline

Support Vector Classification

- Training data $(\mathbf{x}_i, y_i), i = 1, \ldots, l, \mathbf{x}_i \in R^n, y_i = \pm 1$
- Maximizing the margin (Boser et al., 1992; Cortes and Vapnik, 1995)

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l} \max(1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b), 0)$$

- High dimensional (maybe infinite) feature space

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots).$$

- $\mathbf{w}$: maybe infinite variables

Support Vector Classification (Cont'd)

- The dual problem (finite # variables)

$$\begin{array}{ll} \min\limits_{\boldsymbol{\alpha}} & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{subject to} & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T\boldsymbol{\alpha} = 0, \end{array}$$

  where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \ldots, 1]^T$
- At optimum

$$\mathbf{w} = \textstyle\sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x}_i)$$

- Kernel: $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$; closed form

  Example: Gaussian (RBF) kernel: $e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}$

Computational and Memory Bottleneck I

- The square kernel matrix.

$$O(l^2) \text{ memory and } O(l^2 n) \text{ computation}$$

- If $l = 10^6$, then

$$10^{12} \times 8 \text{ bytes } = 8\text{TB}$$

- Distributed implementations include, for example, Chang et al. (2008); Zhu et al. (2009)

  We will look at ideas of these two implementations
- Because the computational cost is high (not linear), the data loading and communication cost is less a concern.

The Approach by Chang et al. (2008) I

- Kernel matrix approximation.
- Original matrix Q with

$$Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

- Consider

$$\bar{Q} = \bar{\Phi}^T \bar{\Phi} \approx Q.$$

- $\bar{\Phi} \equiv [\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_l]$ becomes new training data
- $\bar{\Phi} \in R^{d \times l}, d \ll l$. # features $\ll$ # data
- Testing is an issue, but let's not worry about it here

The Approach by Chang et al. (2008) II

- They follow Fine and Scheinberg (2001) to use incomplete Cholesky factorization
- What is Cholesky factorization?

  Any symmetric positive definite Q can be factorized as

  $$Q = LL^T,$$

  where $L \in R^{l \times l}$ is lower triangular

The Approach by Chang et al. (2008) III

- There are several ways to do Cholesky factorization. If we do it columnwisely

$$\begin{bmatrix} L_{11} & & \\ L_{21} & & \\ L_{31} & & \\ L_{41} & & \\ L_{51} & & \end{bmatrix} \Rightarrow \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & \\ L_{41} & L_{42} & \\ L_{51} & L_{52} & \end{bmatrix} \Rightarrow \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \\ L_{41} & L_{42} & L_{43} \\ L_{51} & L_{52} & L_{53} \end{bmatrix}$$

  and stop before it's fully done, then we get incomplete Cholesky factorization

The Approach by Chang et al. (2008) IV

- To get one column, we need to use previous columns:

$$\begin{bmatrix} L_{43} \\ L_{53} \end{bmatrix} \text{ needs } \begin{bmatrix} Q_{43} \\ Q_{53} \end{bmatrix} - \begin{bmatrix} L_{41} & L_{42} \\ L_{51} & L_{52} \end{bmatrix} \begin{bmatrix} L_{31} \\ L_{32} \end{bmatrix}$$

- The matrix-vector product is parallelized. Each machine is responsible for several rows
- Using $d = \sqrt{l}$, they report the following training time

The Approach by Chang et al. (2008) V

Nodes	Image (200k)	CoverType (500k)	RCV (800k)
10	1,958	16,818	45,135
200	814	1,655	2,671

- We can see that communication cost is a concern
- The reason they can get speedup is because the complexity of the algorithm is more than linear
- They implemented MPI in Google distributed environments
- If MapReduce is used, scalability will be worse

A Primal Method by Zhu et al. (2009) I

- They consider stochastic gradient descent methods (SGD)
- SGD is popular for linear SVM (i.e., kernels not used).
- At the tth iteration, a training instance $\mathbf{x}_{i_t}$ is chosen and $\mathbf{w}$ is updated by

$$\mathbf{w} \leftarrow \mathbf{w} - \eta_t \nabla^S \Big(\frac{1}{2}\|\mathbf{w}\|_2^2 + C \max(0, 1 - y_{i_t} \mathbf{w}^T \phi(\mathbf{x}_{i_t})) \Big),$$

∇^S: a sub-gradient operator; η: learning rate.

A Primal Method by Zhu et al. (2009) II

- Bias term b omitted here
- The update rule becomes

$$\text{If} \quad 1 - y_{i_t}\mathbf{w}^T\mathbf{x}_{i_t} > 0, \quad \text{then}$$
$$\mathbf{w} \leftarrow (1-\eta_t)\mathbf{w} + \eta_t C y_{i_t}\phi(\mathbf{x}_{i_t}).$$

- For kernel SVM, neither $\phi(\mathbf{x})$ nor $\mathbf{w}$ can be stored. So we need to store all $\eta_1, \ldots, \eta_t$

A Primal Method by Zhu et al. (2009) III

- The calculation of

$$\mathbf{w}^T \mathbf{x}_{i_t}$$

  becomes

$$\sum_{s=1}^{t-1} (\text{some coefficient}) K(\mathbf{x}_{i_s}, \mathbf{x}_{i_t}) \tag{1}$$

- Parallel implementation.

  If $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_t}$ distributedly stored, then (1) can be computed in parallel
- Two challenges

A Primal Method by Zhu et al. (2009) IV

1. $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_t}$ must be evenly distributed to nodes, so (1) can be fast.
2. The communication cost can be high

– Each node must have $\mathbf{x}_{i_t}$

– Results from (1) must be summed up

- Zhu et al. (2009) propose some ways to handle these two problems
- Note that Zhu et al. (2009) use a more sophisticated SGD by Shalev-Shwartz et al. (2011), though concepts are similar.
- MPI rather than MapReduce is used

A Primal Method by Zhu et al. (2009) V

- Again, if they use MapReduce, the communication cost will be a big concern

Discussion: Parallel Kernel SVM

- An attempt to use MapReduce is by Liu (2010)
  As expected, the speedup is not good
- From both Chang et al. (2008); Zhu et al. (2009), we know that algorithms must be carefully designed so that time saved on computation can compensate communication/loading

Outline

Linear Support Vector Machines

- By linear we mean kernels are not used
- For certain problems, accuracy by linear is as good as nonlinear

  But training and testing are much faster
- Especially document classification

  Number of features (bag-of-words model) very large
- Recently there are many papers and software

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Comparison Between Linear and Nonlinear (Training Time & Testing Accuracy)

	Linear		RBF Kernel	
Data set	Time	Accuracy	Time	Accuracy
MNIST38	0.1	96.82	38.1	99.70
ijcnn1	1.6	91.81	26.8	98.69
covtype	1.4	76.37	46,695.8	96.11
news20	1.1	96.95	383.2	96.90
real-sim	0.3	97.44	938.3	97.82
yahoo-japan	3.1	92.63	20,955.2	93.31
webspam	25.7	93.35	15,681.8	99.26

Size reasonably large: e.g., yahoo-japan: 140k instances and 830k features

Parallel Linear SVM I

- Training linear SVM is faster than kernel SVM because **w** can be maintained
- Recall that SGD's update rule is

$$\begin{aligned} &\text{If} \quad 1 - y_{i_t}\mathbf{w}^T\mathbf{x}_{i_t} > 0, \quad \text{then} \\ &\quad \mathbf{w} \leftarrow (1-\eta_t)\mathbf{w} + \eta_t C y_{i_t}\mathbf{x}_{i_t}. \end{aligned} \tag{2}$$

Parallel Linear SVM II

- For linear, we directly calculate

$$\mathbf{w}^T \mathbf{x}_{i_t}$$

For kernel, $\mathbf{w}$ cannot be stored. So we need to store all $\eta_1, \ldots, \eta_{t-1}$

$$\sum_{s=1}^{t-1} (\text{some coefficient}) K(\mathbf{x}_{i_s}, \mathbf{x}_{i_t})$$

- For linear SVM, each iteration is cheap.
- It is difficult to parallelize the code

Parallel Linear SVM III

- Issues for parallelization

  - Many methods (e.g., stochastic gradient descent or coordinate descent) are inherently sequential

  - Communication cost is a concern

Simple Distributed Linear Classification I

- Bagging: train several subsets and ensemble results
  - Useful in distributed environments; each node ⇒ a subset
  - Example: Zinkevich et al. (2010)
- Some results by averaging models

	yahoo-korea	kddcup10	webspam	epsilson
Using all	87.29	89.89	99.51	89.78
Avg. models	86.08	89.64	98.40	88.83

- Using all: solves a single linear SVM

Simple Distributed Linear Classification II

- Avg. models: each node solves a linear SVM on a subset
- Slightly worse but in general OK

ADMM by Boyd et al. (2011) I

- Recall the SVM problem (bias term b omitted)

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

- An equivalent optimization problem

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_m,\mathbf{z}} \quad \frac{1}{2}\mathbf{z}^T\mathbf{z} + C\sum_{j=1}^{m}\sum_{i\in B_j}\max(0, 1 - y_i\mathbf{w}_j^T\mathbf{x}_i) +$$

$$\frac{\rho}{2}\sum_{j=1}^{m}\|\mathbf{w}_j - \mathbf{z}\|^2$$

$$\text{subject to} \quad \mathbf{w}_j - \mathbf{z} = \mathbf{0}, \forall j$$

ADMM by Boyd et al. (2011) II

- The key is that

$$\mathbf{z} = \mathbf{w}_1 = \cdots = \mathbf{w}_m$$

  are all optimal $\mathbf{w}$
- This optimization problem was proposed in 1970s, but is now applied to distributed machine learning
- Each node has a subset B_j and updates $\mathbf{w}_j$
- Only $\mathbf{w}_1, \ldots, \mathbf{w}_m$ must be collected

  Data are not moved; less communication cost
- Still, we cannot afford too many iterations because of communication cost

ADMM by Boyd et al. (2011) III

- An MPI implementation is by Zhang et al. (2012)
- I am not aware of any MapReduce implementation yet

Vowpal_Wabbit (Langford et al., 2007) I

- It started as a linear classification package on a single computer
- It actually solves logistic regression rather than SVM.
- After version 6.0, Hadoop support has been provided
- A hybrid approach: parallel SGD initially and switch to LBFGS (quasi Newton)
- They argue that AllReduce is a more suitable operation than MapReduce
- What is AllReduce?

Vowpal_Wabbit (Langford et al., 2007) II

Every node starts with a value and ends up with the sum at all nodes

- In Agarwal et al. (2012), the authors argue that many machine learning algorithms can be implemented using AllReduce

  LBFGS is an example

- In the following talk

  Scaling Up Machine Learning

  the authors train 17B samples with 16M features on 1K nodes ⇒ 70 minutes

Outline

1. Why distributed machine learning?
2. Distributed classification algorithms
   - Kernel support vector machines
   - Linear support vector machines
   - Parallel tree learning
3. Distributed clustering algorithms
   - k-means
   - Spectral clustering
   - Topic models
4. Discussion and conclusions

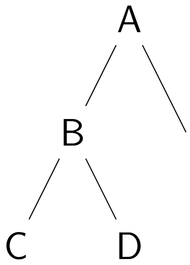
Parallel Tree Learning I

- We describe the work by Panda et al. (2009)
- It considers two parallel tasks

  \- single tree generation

  \- tree ensembles
- The main procedure of constructing a tree is to decide how to split a node
- This becomes difficult if data are larger than a machine's memory
- Basic idea:

Parallel Tree Learning II



If A and B are finished, then we can generate C and D in parallel

- But a more careful design is needed. If data for C can fit in memory, we should generate all subsequent nodes on a machine

Parallel Tree Learning III

- That is, when we are close to leaf nodes, no need to use parallel programs

  If you have only few samples, a parallel implementation is slower than one single machine
- The concept looks simple, but generating a useful code is not easy
- The authors mentioned that they face some challenges

  - "MapReduce was not intended ... for highly iterative process .., MapReduce start and tear down costs were primary bottlenecks"

Parallel Tree Learning IV

- "cost ... in determining split points ... higher than expected"

  - "... though MapReduce offers graceful handling of failures within a specific MapReduce ..., since our computation spans multiple MapReduce ..."

- The authors address these issues using engineering techniques.
- In some places they even need RPCs (Remote Procedure Calls) rather than standard MapReduce
- For 314 million instances ($>$ 50G storage), in 2009 they report

Parallel Tree Learning V

nodes	time (s)
25	≈ 400
200	$\approx 1{,}350$

- This is good in 2009. At least they trained a set where one single machine cannot handle at that time
- The running time does not decrease from 200 to 400 nodes
- This study shows that

Parallel Tree Learning VI

- Implementing a distributed learning algorithm is not easy. You may need to solve certain engineering issues

- But sometimes you must do it because of handling huge data

Outline

1. Why distributed machine learning?
2. Distributed classification algorithms
   - Kernel support vector machines
   - Linear support vector machines
   - Parallel tree learning
3. Distributed clustering algorithms
   - k-means
   - Spectral clustering
   - Topic models
4. Discussion and conclusions

Outline

k-means I

- One of the most basic and widely used clustering algorithms
- The idea is very simple.
- Finding k cluster centers and assign each data to the cluster of its closest center

k-means II

Algorithm 1 k-means procedure

1. Find initial k centers
2. While not converge
   - Find each point's closest center
   - Update centers by averaging all its members

We discuss difference between MPI and MapReduce implementations of k-means

k-means: MPI Implementation I

- Broadcast initial centers to all machines
- While not converged

  Each node assigns its data to k clusters and compute local sum of each cluster

  An MPI_AllReduce operation obtains sum of all k clusters to find new centers
- Communication versus computation:
- If $\mathbf{x} \in R^n$, then each node transfer

  kn elements (local sum) after $kn \times l/p$ operations,

  l: total number of data and p: number of nodes.

k-means: MapReduce implementation I

- We describe one implementation by Thomas Jungblut

  `http://codingwiththomas.blogspot.com/2011/05/k-means-clustering-with-mapreduce.html`
- You don't specifically assign data to nodes

  That is, data has been stored somewhere at HDFS
- Each instance: a (key, value) pair

  key: its associated cluster center

  value: the instance

k-means: MapReduce implementation II

- Map:

  Each (key, value) pair find the closest center and update the key (after loading all data centers)

- Reduce:

  For instances with the same key (cluster), calculate the new cluster center (and save data centers)

- As we said earlier, you don't control where data points are

  Therefore, it's unclear how expensive loading and communication is

Outline

Spectral Clustering I

Input: Data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$; k: number of desired clusters.

1. Construct similarity matrix $S \in R^{n \times n}$.
2. Modify S to be a sparse matrix.
3. Compute the Laplacian matrix L by

$$L = I - D^{-1/2} S D^{-1/2},$$

4. Compute the first k eigenvectors of L; and construct $V \in R^{n \times k}$, whose columns are the k eigenvectors.

Spectral Clustering II

5. Compute the normalized matrix U of V by

$$U_{ij} = \frac{V_{ij}}{\sqrt{\sum_{r=1}^{k} V_{ir}^2}}, i = 1, \ldots, n, j = 1, \ldots, k.$$

6. Use k-means algorithm to cluster n rows of U into k groups.

Early studies of this method were by, for example, Shi and Malik (2000); Ng et al. (2001)

We discuss the parallel implementation by Chen et al. (2011)

MPI and MapReduce

Similarity matrix

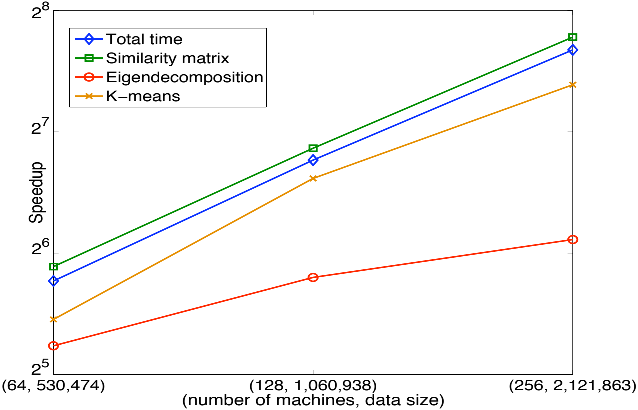
- Only done once: suitable for MapReduce
- But size grows in $O(n^2)$

First k Eigenvectors

- An iterative algorithm called implicitly restarted Arnoldi
- Iterative: not suitable for MapReduce
- MPI is used but no fault tolerance

Sample Results I

2,121,863 points and 1,000 classes



Sample Results II

We can see that scalability of eigen decomposition is not good

Nodes	Similarity	Eigen	kmeans	Total	Speedup
16	752542s	25049s	18223s	795814s	16.00
32	377001s	12772s	9337s	399110s	31.90
64	192029s	8751s	4591s	205371s	62.00
128	101260s	6641s	2944s	110845s	114.87
256	54726s	5797s	1740s	62263s	204.50

How to Scale Up?

- We can see two bottlenecks
  - computation: $O(n^2)$ similarity matrix
  - communication: finding eigenvectors
- To handle even larger sets we may need to use non-iterative algorithms (e.g., Nyström approximation)

  Slightly worse performance, but may scale up better

Outline

1. Why distributed machine learning?
2. Distributed classification algorithms
   - Kernel support vector machines
   - Linear support vector machines
   - Parallel tree learning
3. Distributed clustering algorithms
   - k-means
   - Spectral clustering
   - Topic models
4. Discussion and conclusions

Latent Dirichlet Allocation I

- LDA (Blei et al., 2003) detects topics from documents
- Finding the hidden structure of texts
- For example, Figure 2 of Blei (2012) shows that 100-topic LDA to science paper gives frequent words like

"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
⋮	⋮	⋮	⋮

Latent Dirichlet Allocation II

- The LDA model

$$p(\mathbf{w}, \mathbf{z}, \boldsymbol{\Theta}, \boldsymbol{\Phi} | \boldsymbol{\alpha}, \boldsymbol{\beta}) =$$

$$\left[\prod_{i=1}^{m}\prod_{j=1}^{m_i} p(w_{ij}|z_{ij}, \boldsymbol{\Phi}) p(z_{ij}|\boldsymbol{\theta}_i)\right]\left[\prod_{i=1}^{m} p(\boldsymbol{\theta}_i|\boldsymbol{\alpha})\right]\left[\prod_{j=1}^{k} p(\boldsymbol{\phi}_j|\boldsymbol{\beta})\right]$$

- w_{ij}: jth word from ith document
  z_{ij}: the topic
- $p(w_{ij}|z_{ij}, \boldsymbol{\Phi})$ and $p(z_{ij}|\boldsymbol{\theta}_i)$: multinomial distributions
  That is, w_{ij} is drawn from $z_{ij}, \boldsymbol{\Phi}$ and z_{ij} is drawn from $\boldsymbol{\theta}_i$

Latent Dirichlet Allocation III

- $\mathbf{\Phi}$: distribution over vocabulary

  $\boldsymbol{\theta}_i$: topic proportion for the ith document
- $p(\boldsymbol{\theta}_i|\boldsymbol{\alpha}), p(\phi_j|\boldsymbol{\beta})$: Dirichlet distributions

  $\boldsymbol{\alpha}, \boldsymbol{\beta}$: prior of $\mathbf{\Theta}, \mathbf{\Phi}$, respectively
- Maximizing the likelihood is not easy, so Griffiths and Steyvers (2004) propose using Gipps sampling to iteratively estimate the posterior $p(\mathbf{z}|\mathbf{w})$

Latent Dirichlet Allocation IV

- While the model looks complicated, $\mathbf{\Theta}$ and $\mathbf{\Phi}$ can be integrated out to

$$p(\mathbf{w}, \mathbf{z} | \boldsymbol{\alpha}, \boldsymbol{\beta})$$

  Then at each iteration only a counting procedure is needed

- We omit details but essentially the algorithm is

Latent Dirichlet Allocation V

Algorithm 2 LDA Algorithm

For each iteration

For each document i

For each word j in document i

Sampling and counting

- Distributed learning seems straightforward
  - Divide data to several nodes
  - Each node counts local data
  - Models are summed up

Latent Dirichlet Allocation VI

- However, an efficient implementation is not that simple
- Some existing implementations

  Wang et al. (2009): both MPI and MapReduce

  Newman et al. (2009): MPI

  Smola and Narayanamurthy (2010): Something else
- Smola and Narayanamurthy (2010) claim higher throughputs.

  These works all use same algorithm, but implementations are different

Latent Dirichlet Allocation VII

- A direct MapReduce implementation may not be efficient due to I/O at each iteration
- Smola and Narayanamurthy (2010) use quite sophisticated techniques to get high throughputs

  \- They don't partition documents to several machines. Otherwise machines need to wait for synchronization

  \- Instead, they consider several samplers and synchronize between them

  \- They use memcached so data stored in memory rather than disk

Latent Dirichlet Allocation VIII

- They use Hadoop streaming so C++ rather than Java is used
- And some other techniques

- We can see that a efficient implementation is not easy

Outline

Integration with the Whole Workflow I

- We mentioned before that sometimes copy data from distributed systems to a single machine isn't convenient

  Workflow is broken
- Training is sometimes only a "small part" of the whole data management workflow
- Example: the approach at Twitter (Lin and Kolcz, 2012)
- They write Pig scripts for data management tasks (including classification)

Integration with the Whole Workflow II

- It's just like you write Matlab code
- Sample code in Lin and Kolcz (2012)

```
-- Filter for positive examples
positive = filter status by ContainsPositiveEmoticon(text
positive = foreach positive generate 1 as label, RemovePo
positive = order positive by random; -- Randomize orderin
positive = limit positive $N; -- Take N positive examples
-- Filter for negative examples
...
-- Randomize order of positive and negative examples
training = foreach training generate $0 as label, $1 as t
training = order training by random parallel $PARTITIONS;
training = foreach training generate label, text;
```

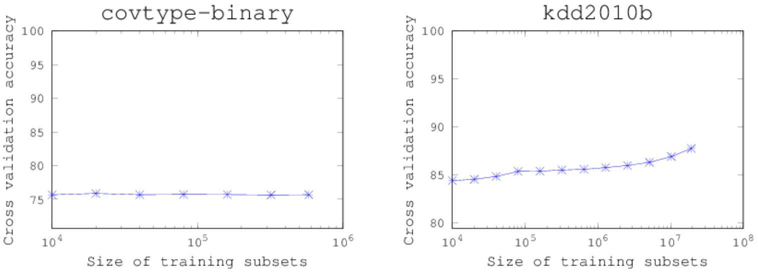
Integration with the Whole Workflow III

```
store training into $OUTPUT using TextLRClassifierBuilder
```

- They use stochastic gradient descent methods
- You may question that this is a sequential algorithm
- But according to the authors, they go through all data only once
- But that's enough for their application
- Software engineering issues to put things together become the main issues rather than machine learning algorithms

Training Size versus Accuracy I

- More training data may be helpful for some problems, but not others
- See the following two figures of cross-validation accuracy versus training size



Training Size versus Accuracy II

- If more data points don't help, probably there is no need to run a distributed algorithm
- Can we easily know how many data points are enough?

  Could machine learning people provide some guidelines?

System Issues I

- Systems related to distributed data management are still being rapidly developed
- An important fact is that existing distributed systems or parallel frameworks are not particularly designed for machine learning algorithms

  For example, Hadoop is slow for iterative algorithms due to heavy disk I/O
- I will illustrate this point by the following example for a bagging implementation
- Assume data is large, say 1TB. You have 10 machines with 100GB RAM each.

System Issues II

- One way to train this large data is a bagging approach

machine	1	trains	1/10	data
	2		1/10	
	⋮		⋮	
	10		1/10	

- Then use 10 models for prediction and combine results
- Reasons of doing so is obvious: parallel data loading and parallel computation

System Issues III

- But it is not that simple if using MapReduce and Hadoop.
- Hadoop file system is not designed so we can easily copy a subset of data to a node

  That is, you cannot say: block 10 goes to node 75
- A possible way is

  1. Copy all data to HDFS

  2. Let each n/p points to have the same key (assume p is # of nodes). The reduce phase collects n/p points to a node. Then we can do the parallel training

System Issues IV

- As a result, we may not get 1/10 loading time
- In Hadoop, data are transparent to users

  We don't know details of data locality and communication

Here is an interesting communication between me and a friend (called D here)

- Me: If I have data in several blocks and would like to copy them to HDFS, it's not easy to specifically assign them to different machines
- D: yes, that's right.

System Issues V

- Me: So probably using a poor-man's approach is easier. I use USB to copy block/code to 10 machines and hit return 10 times
- D: Yes, but you can do better by `scp` and `ssh`. Indeed that's usually how I do "parallel programming"

This example is a bit extreme, but it clearly shows that large-scale machine learning is strongly related to many system issues

Also, some (Lin, 2012) argue that instead of developing new systems to replace Hadoop, we should modify machine learning algorithms to "fit" Hadoop

Easy of Use I

- Distributed programs and systems are complicated
- Simplicity and easy of use are very important in designing such tools
- From a Google research blog by Simon Tong on their classification tool SETI:

  "It is perhaps less academically interesting to design an algorithm that is slightly worse in accuracy, but that has greater ease of use and system reliability. However, in our experience, it is very valuable in practice."

Easy of Use II

- Title of the last slide of another Google tool Sibyl at MLSS Santa Cruz 2012:

  "Lesson learned (future direction): Focus on easy of use"
- Also from Simon Tong's blog: it is recommended to "start with a few specific applications in mind"
- That is, let problems drive the tools (Lin and Kolcz, 2012)

Conclusions

- Distributed machine learning is still an active research topic
- It is related to both machine learning and systems
- An important fact is that existing distributed systems or parallel frameworks are not particularly designed for machine learning algorithms
- Machine learning people can

  - help to affect how systems are designed

  - design new algorithms for existing systems

Acknowledgments

I thank comments from

- Wen-Yen Chen
- Dennis DeCoste
- Alex Smola
- Chien-Chih Wang
- Xiaoyun Wu
- Rong Yen

References I

A. Agarwal, O. Chapelle, and M. D. J. Langford. A reliable effective terascale linear learning system. 2012. Submitted to KDD 2012.

D. M. Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. MIT Press, Cambridge, MA, 2008.

W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33 (3):568–586, 2011.

References II

C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA, 2007.

C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

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

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101:5228–5235, 2004.

W. Gropp, E. Lusk, and A. Skjellum. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press,, 1999.

References III

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf.

J. Langford, L. Li, and A. Strehl. Vowpal Wabbit, 2007. https://github.com/JohnLangford/vowpal_wabbit/wiki.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

J. Lin. MapReduce is good enough? if all you have is a hammer, throw away everything that's not a nail!, 2012. arXiv preprint arXiv:1209.2191.

J. Lin and A. Kolcz. Large-scale machine learning at Twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD 2012)*, pages 793–804, 2012.

S. Liu. Upscaling key machine learning algorithms. Master's thesis, University of Bristol, 2010.

D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10:1801–1828, 2009.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of NIPS*, pages 849–856, 2001.

References IV

B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. PLANET: massively parallel learning of tree ensembles with mapreduce. *Proceedings of VLDB*, 2(2):1426–1437, 2009.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

A. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *Proceedings of the VLDB Endowment*, volume 3, pages 703–710, 2010.

M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.

J. Talbot, R. M. Yoo, and C. Kozyrakis. Phoenix++: Modular mapreduce for shared-memory systems. In *Second International Workshop on MapReduce and its Applications*, June 2011.

Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang. PLDA: Parallel latent Dirichlet allocation for large-scale applications. In *International Conference on Algorithmic Aspects in Information and Management*, 2009.

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

References V

C. Zhang, H. Lee, and K. G. Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 2012.

Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen. P-packSVM: Parallel primal gradient descent kernel SVM. In *Proceedings of the IEEE International Conference on Data Mining*, 2009.

M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603. 2010.