Some Thoughts on Large-scale Data Classification

Chih-Jen Lin
National Taiwan University



Talk at Baidu, October 24, 2012

Outline

1. Introduction
2. Different considerations for large-scale classification
3. Possible Directions



Outline

1 Introduction

2 Different considerations for large-scale classification

3 Possible Directions



Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example

  Find a patient's blood pressure, weight, etc.

  After several years, know if he/she recovers

  Build a machine learning model

  New patient: find blood pressure, weight, etc

  Prediction
- Training and testing



Data Classification (Cont'd)

- Representative methods

  Nearest neighbor, naive Bayes

  Decision tree, random forest

  Neural networks, support vector machines
- Data classification has been widely used in many areas

  for example, texts, images, and bioinformatics



Large-scale Classification

- Most existing classification algorithms assume that data can fit in memory
- But what if data is too big for one computer?
- One may say that let's design a distributed classification algorithm
- But there are many considerations different from those in traditional machine learning



Let's Start with An Example

- Using a simple 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

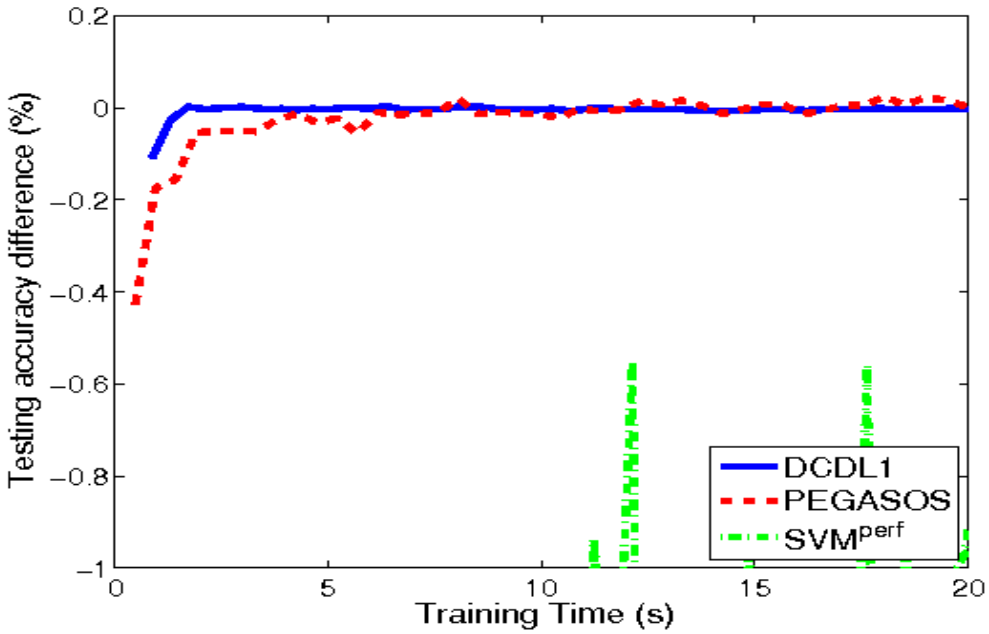
- Therefore,

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

  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 clusters 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 RAM and disk are 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

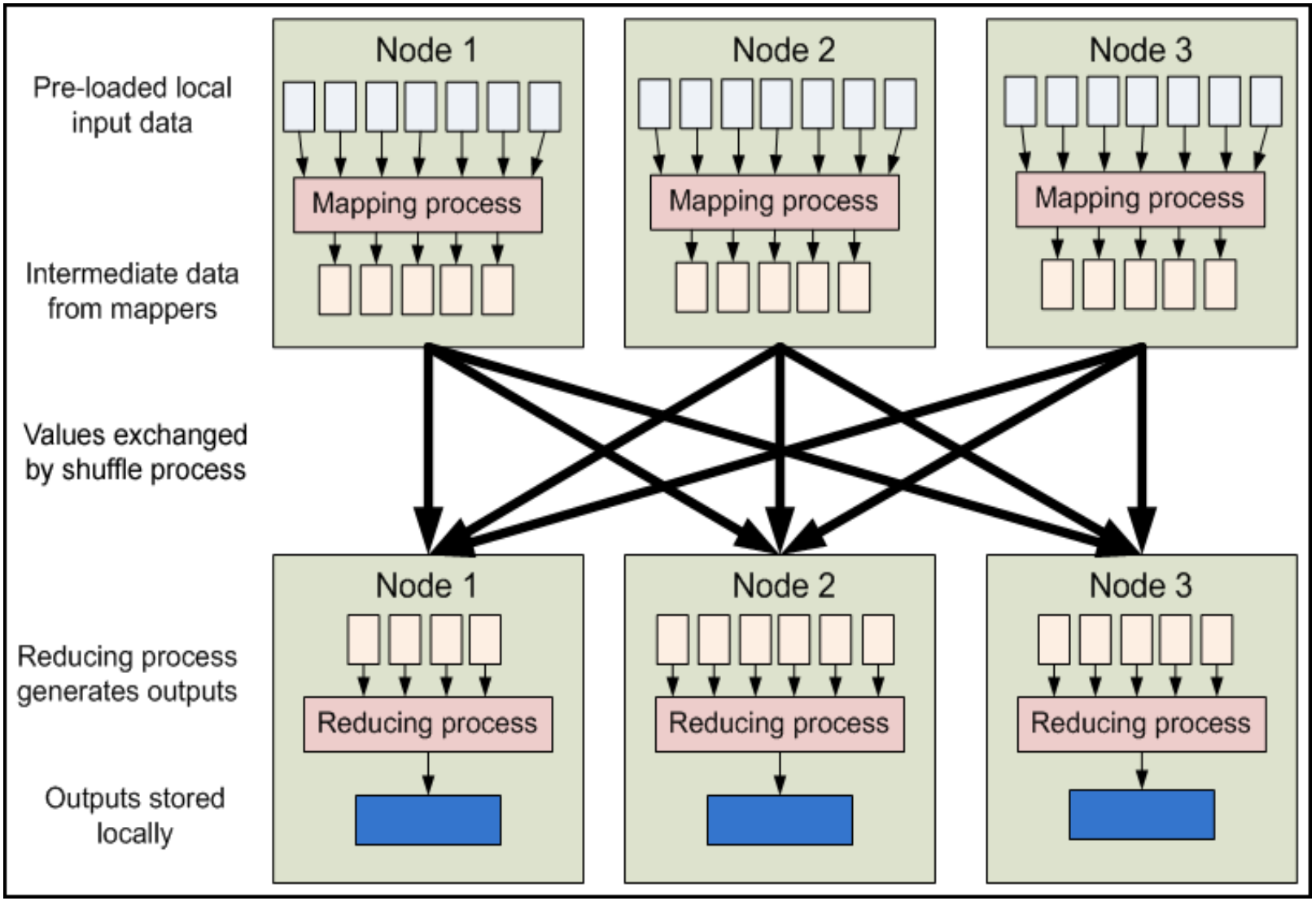
- MapReduce (Dean and Ghemawat, 2008). A framework now commonly used for large-scale data processing
- In MapReduce, every element is a (key, value) pair

  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





An Introduction of Distributed Systems V

- 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 VI

- 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

  For example, shared memory (Talbot et al., 2011) and distributed clusters (e.g., Hadoop)



An Introduction of Distributed Systems VII

- 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
- These algorithms include linear regression, k-means, logistic regression, naive Bayes, SVM, ICA, PCA, EM, Neural networks, etc



An Introduction of Distributed Systems VIII

- 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



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



Outline

1 Introduction

2 Different considerations for large-scale classification

3 Possible Directions



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
- This is different from clustering, where less frequently we use a subset to produce a model



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



Outline

1 Introduction

2 Different considerations for large-scale classification

3 Possible Directions



Integration with the Whole Workflow I

- This is possible if training is a "small part" of the whole workflow
- Example: the approach at Twitter (Lin and Kolcz, 2012)

Hadoop
↓
Pig
↓
Classification

- You write scripts for such tasks

  It's just like you write Matlab code



Integration with the Whole Workflow II

- 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;

store training into $OUTPUT using TextLRClassifierBuilder
```



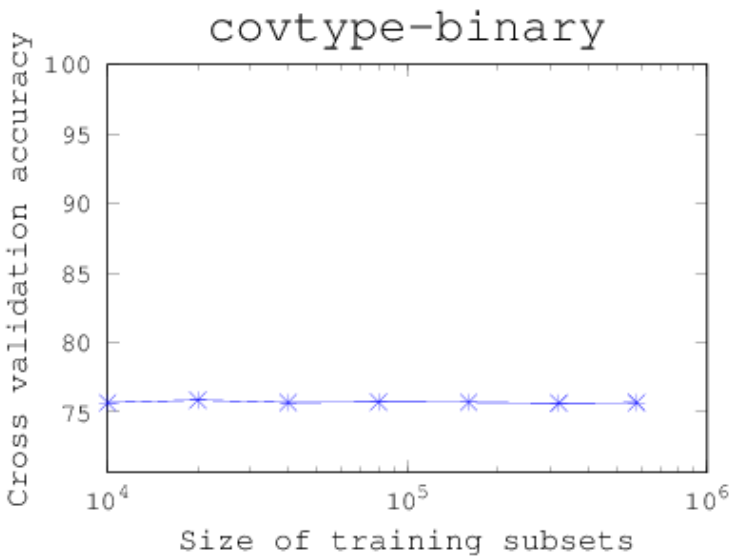
Integration with the Whole Workflow III

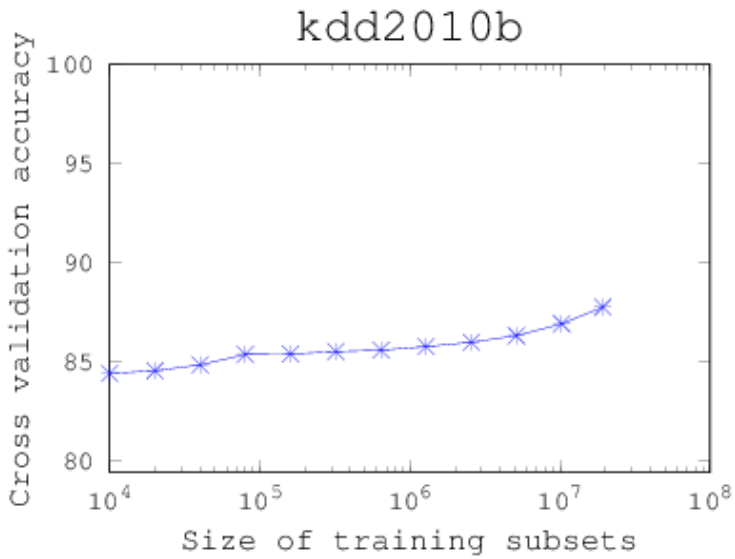
- They use stochastic gradient descent methods
- You may question that this is a sequential algorithm
- But according to the authors, they go through all data only once
- Therefore, training is only a small part of the whole procedure
- Software engineering issues to put things together become the main 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

  We mentioned that many new systems aiming to replace Hadoop are still being developed
- An important fact is that existing distributed systems or parallel frameworks are not particularly designed for machine learning algorithms
- 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



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

- Large-scale classification 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 the design of new systems
  - design new algorithms for existing systems

