When and When Not to Use Distributed Machine Learning

Chih-Jen Lin
Department of Computer Science
National Taiwan University

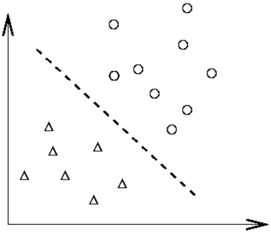
2nd International Winter School on Big Data, February 2016

Outline

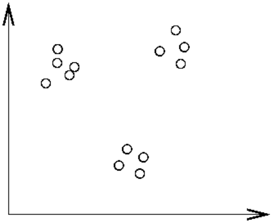
Outline

Machine Learning

- Extract knowledge from data
- Representative tasks:
  Classification, clustering, ranking and others



Classification



Clustering

Today I will focus more on classification

Data Classification

- Given training data in different classes (labels known)

  Predict test data (labels unknown)
- Classic example: medical diagnosis

  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

Traditional Ways of Doing Machine Learning

- Get an integrated tool for data mining and machine learning (e.g., Weka, R, Scikit-learn)

  You can also get an implementation of a particular ML algorithm (e.g., LIBSVM)
- Pre-process the raw data and then run ML algorithms
- Conduct analysis on the results

All these are done in the RAM of one computer

Traditional Ways of Doing Machine Learning (Cont'd)

- But the traditional way may not work if data are too large to store in the RAM of one computer
- Most existing machine learning algorithms are designed by assuming that data can be easily accessed
- Therefore, the same data may be accessed many times
- But random access of data from disk is slow

Outline

Handling Very Large Data

Some possible approaches

- Buy a machine with several TB RAM
  - We can use existing methods/tools
  - But we cannot handle extremely large data
  - Initial data loading can be time consuming
  - We need to subsample and transfer data to one machine
- Disk-level machine learning
  - Can handle bigger data
  - But frequent data access from disk is a big concern

Handling Very Large Data (Cont'd)

- Distributed machine learning
  - Parallel data loading and fault tolerance
  - Communication and synchronization are concerns
  - Programs become more complicated

Handling Very Large Data (Cont'd)

- Currently there are various types of arguments
- Some say that single machines with huge RAM is the way to go. Their arguments are
  - RAM in a machine is getting bigger and bigger
    From KDnuggets News (15:n11), the increase of RAM has been much faster than the increase of the typical data set used
  - There are not so many big data – only Google or Facebook has

Handling Very Large Data (Cont'd)

- More arguments for the single-machine model:
  - Loading data from disk can be fast if for example you store data in binary format
  - You load data once and keep it in memory for analysis (e.g., using MATLAB or R)
  - If proper feature engineering is done, then you don't need lots of data

Handling Very Large Data (Cont'd)

- In contrast, some say that in the future data analytics will be mainly done in a distributed environment
    - Big data is everywhere – a simple health application can easily accumulate lots of information

Handling Very Large Data (Cont'd)

- I think different types of approaches will exist
- However, when to use which is the big issue
- I will discuss issues that need to be considered

Loading time

- Usually on one machine ML people don't care too much about loading time
- However, we will argue that the situation depends on the time spent on computation
- Let's use the following example to illustrate that sometimes loading may be more than computation
- 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

Loading Time (Cont'd)

- On a typical PC: Total time: 50.88 seconds.
  Loading time: 43.51 seconds
- In fact, 2 seconds are enough to get stable test accuracy

$$\text{loading time} \gg \text{running time}$$

- 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 $q \geq 1$.

Loading Time (Cont'd)

- Traditionally running time is larger because of using nonlinear algorithms (i.e., $q > 1$)
- But when l is large, we may use a linear algorithm (i.e., $q = 1$) for efficiency $\Rightarrow$ loading time may dominate
- Parallel data loading:
  - Using 100 machines, each has $1/100$ data in its local disk $\Rightarrow$ $1/100$ loading time
  - But having data ready in these 100 machines is another issue

Fault Tolerance

- Some data are replicated across machines: if one fails, others are still available
- However, having this support isn't easy. MPI has no fault tolerance, but is efficient. MapReduce on Hadoop has it, but is slow.
- In machine learning applications very often training is done off-line
- So if machines fail, you just restart the job. If it does not finish on time, the old model can still be used

Fault Tolerance (Cont'd)

- In this sense, an implementation using MPI (no fault tolerance) may be fine for median-sized problems (e.g., tens or hundreds of nodes)
- However, fault tolerance is needed if you use more machines (e.g., thousands). Restarting a job on thousands of machines is a waste of resources
- This is an interesting example that data size may affect the selection of the programming framework

Communication and Synchronization

- Communication and synchronization are often the bottleneck to cause lengthy running time of distributed machine learning algorithms
- Consider matrix-vector multiplication as an example.

$$X^T \boldsymbol{s}$$

- This operation is common in distributed machine learning

Communication and Synchronization (Cont'd)

- Data matrix X is now distributedly stored

node 1 $\longrightarrow$	X_1
node 2 $\longrightarrow$	X_2
	$\cdots$
node p $\longrightarrow$	X_p

$$X^T \boldsymbol{s} = X_1^T \boldsymbol{s}_1 + \cdots + X_p^T \boldsymbol{s}_p$$

- Synchronization:

$$X_1^T \boldsymbol{s}_1, \ldots, X_p^T \boldsymbol{s}_p$$

  may not finish at the same time

Communication and Synchronization (Cont'd)

- Communication:

$$X_1^T \boldsymbol{s}_1, \ldots, X_p^T \boldsymbol{s}_p$$

  are transferred to a master node for the sum
- Which one is more serious depends on the system configuration
- For example, if your machines are the same, probably the synchronization cost is low

Workflow

- If data are already distributedly stored, it's not convenient to reduce some to one machine for analysis $\Rightarrow$ workflow interrupted
- This is particularly a problem if you must frequently re-train models

Programming Framework

- Unfortunately writing and running a distributed program is a bit complicated
- Further, platforms are still being actively developed (Hadoop, Spark, Reef, etc.)
- Developing distributed machine learning packages becomes difficult because of platform dependency

Going Distributed or Not Isn't Easy to Decide

- Quote from Yann LeCun (KDnuggets News 14:n05)
  "I have seen people insisting on using Hadoop for datasets that could easily fit on a flash drive and could easily be processed on a laptop."
- The decision isn't easy because we have discussed many considerations

Going Distributed or Not Isn't Easy to Decide (Cont'd)

Quote from Xavier Amatriain "10 more lessons learned from building Machine Learning systems":

- You don't need to distribute your ML algorithm.
- Most of what people do in practice can fit into a multi-core machine: smart data sampling, offline schemes and efficient parallel code.
- Example:
  - Spark implementation: 6 hours, 15 machines.
  - Developer time: 4 days
  - Same model on 1 machine within 10 minutes

Scenarios that Need Distributed Linear Classification

- Example: computational advertising (in particular, click-through rate prediction) is an area that heavily uses distributed machine learning
- This application has the following characteristics
  - Frequent re-training so workflow shouldn't be interrupted
  - Data are big, so parallel loading is important

Example: CTR Prediction

- Definition of CTR:

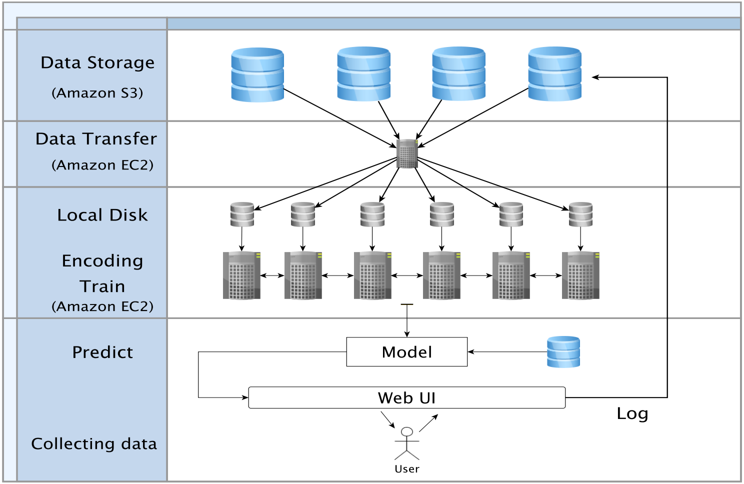
$$\text{CTR} = \frac{\#\ \text{clicks}}{\#\ \text{impressions}}.$$

- A sequence of events

Not clicked	Features of user
Clicked	Features of user
Not clicked	Features of user
...	...

- A binary classification problem.

Example: CTR Prediction (Cont'd)



Outline

Algorithms for Distributed Machine Learning

This is an on-going research topic.

Roughly there are two types of approaches

1. Parallelize existing (single-machine) algorithms

   An advantage is that things such convergence properties still hold

2. Design new algorithms particularly for distributed settings

Of course there are things in between

Algorithms on Single Machines

- Efficient algorithms on single machines are still important
- They can be useful components for distributed machine learning
- Multi-core machine learning is an important research topic
- For example, in the past 2 years we have multi-core and distributed extensions of LIBLINEAR for large-scale linear classification
- So far the multi-core code has more users!

Distributed Machine Learning Frameworks

- Earlier frameworks include, for example,
  - Apache Mahout
  - Spark MLlib
  - MADlib
- There are many ongoing efforts.
- One possible goal is to have a framework that is independent of the distributed programming platforms

Conclusions

- Big-data machine learning is in its infancy. Algorithms and tools in distributed environments are still being actively developed
- We think various settings (e.g., single-machine, distributed, etc.) will exist.
- One must be careful in deciding when to use which