Data Mining, Parallelism, and Grids

David Skillicorn
Queen's University, Kingston
skill@cs.queensu.ca

Data mining builds models from data – in the hope that these models reveal some knowledge about the underlying data.

Think of data as a matrix:

Models of data are used for:

-- prediction (supervised learning, one of the attributes is the target attribute)

- neural networks
- decision trees
- support vector machines

-- understanding (unsupervised learning, learn relationships among objects via their attributes)

- clustering
- neural networks

There are typically three phases to the data mining process:

1. A model is built on a training dataset
2. The model is tested on a test dataset
3. The model is deployed on new data

The quality of a model is measured by the prediction error rate on the test dataset (supervised), or some measure of the consistency and tightness of the relationships (unsupervised).

Applications exist in:

- commercial (customer relationship management);
- industrial (component maintenance prediction);
- scientific (unusual particle interactions);
- engineering (turbulence in fluid flow);

but the commercial sector is by far the largest, and there's still lots of runway.

(The major limitation on further growth is the shortage of skilled people.)

Parallel Data Mining

Since data mining algorithms are both compute-bound and data-access bound, it's natural to use parallelism.

Multiple processors help with the compute part;

Parallel computers have flatter memory hierarchies (more memory is closer to a processor) which helps with data access.

Most DM techniques are approximating in this sense:

The quality of the model improves as more examples are seen.

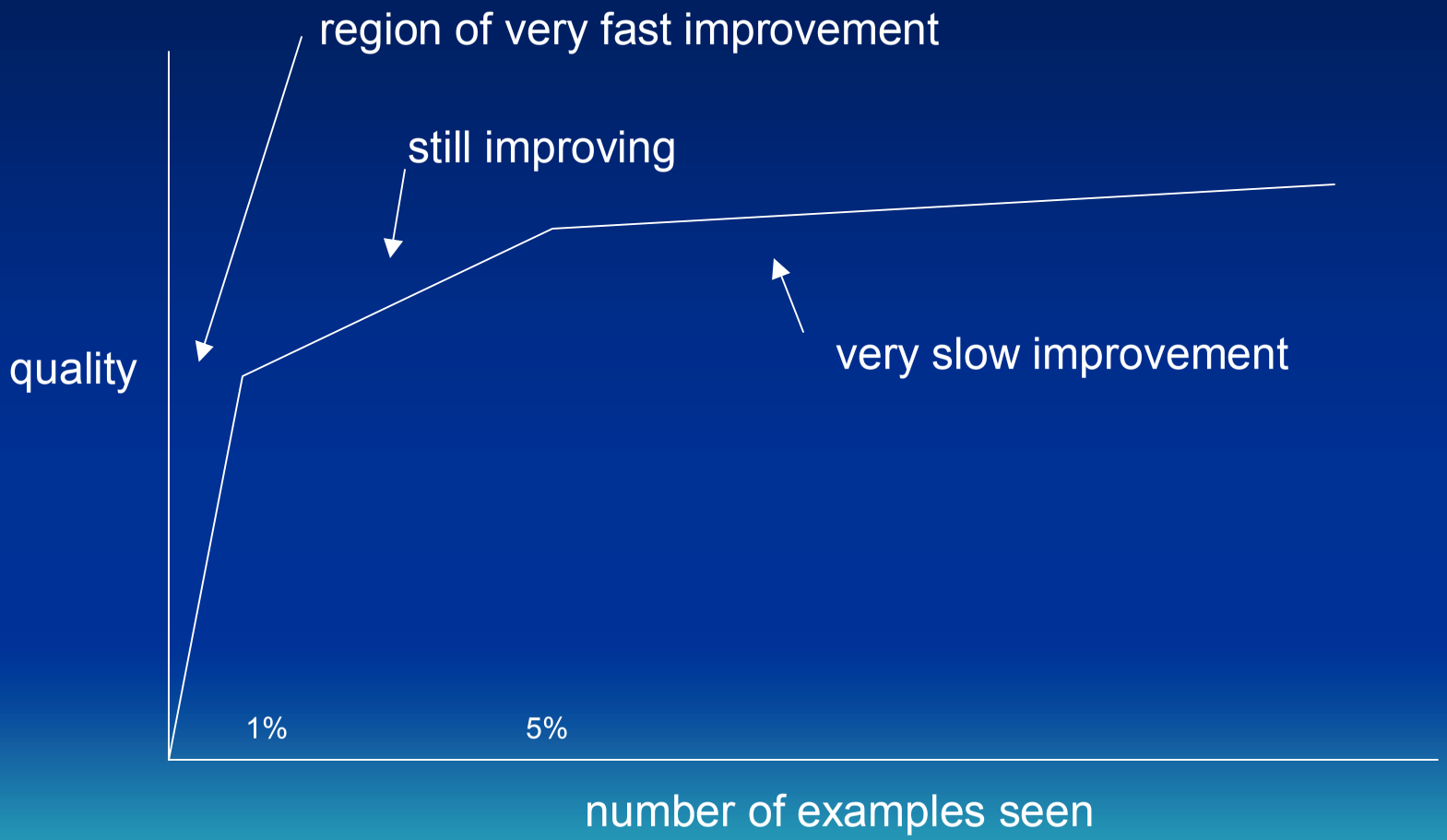
This creates some interesting possibilities:

-- quick and dirty modelling based on small samples

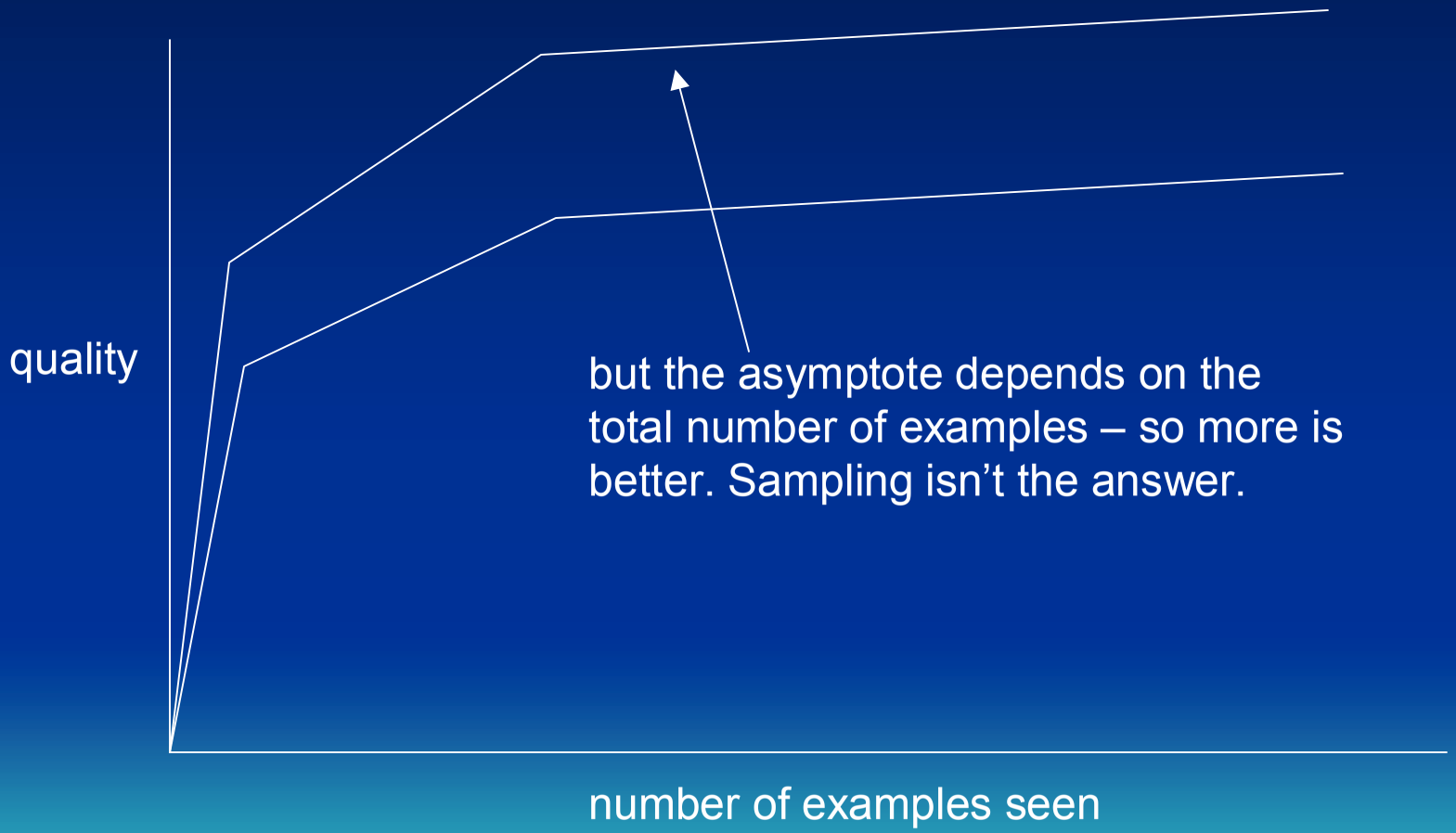
-- steerable modelling where early feedback helps a user select interesting questions

-- parallel and distributed modelling where each processor models its part of a larger dataset

Surprise 1: The rate at which model `quality' improves is much greater than known error bounds suggest.



Often 95% of the objects provide the final 2% of model improvement.



What's going on?

There's a lot of repetition in typical data mining datasets.

Every early example reveals a new aspect of the model. After a while, new examples repeat much of the `knowledge' from examples seen earlier.

A big sample contains more examples that differ from one another. They force the model to consider richer representations.

Strategy for parallelism:

1. Partition the dataset `by rows' and allocate a partition to each processor;
2. Learn a model locally at each processor;
3. (Somehow) merge the local models into a single, global model that would have been produced by a sequential data mining learner.

There's no magic bullet – a new merging technique has to be discovered for each data mining technique.

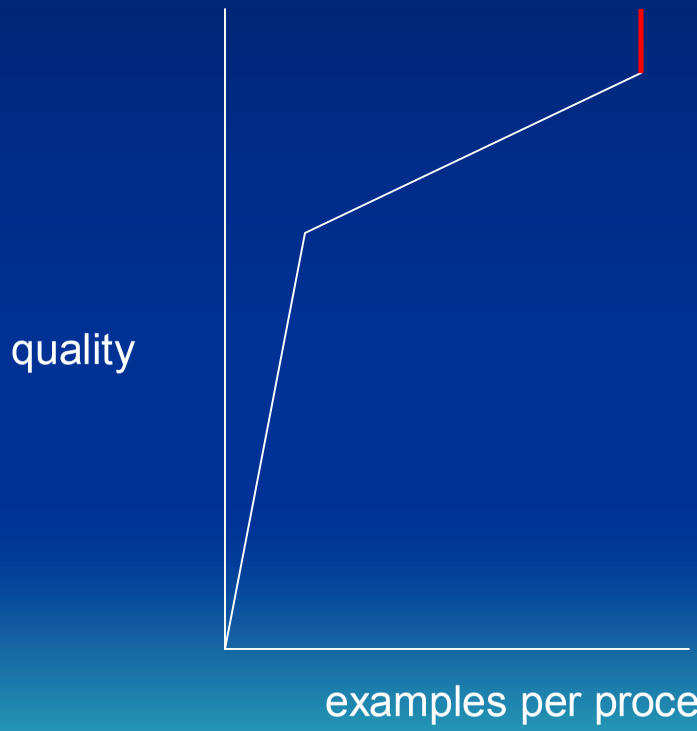
Merging techniques are known for:

1. Neural networks (supervised and unsupervised)
2. Inductive logic programming
3. Frequent sets (and so association rules)
4. Bagging
5. Boosting/arcing

and probably for other techniques too.

Using *p* processors gives an immediate speedup of almost *p* (less merging overhead).
(Speedup Factor 1)

But… each processor is working with early examples. The merge step improves model quality without seeing fresh examples.



(Speedup Factor 2)

So there's an extra speedup (real superlinear speedup) because every cycle is being spent learning in a productive range of the dataset – convergence happens more quickly.

[Of course, this means that sequential implementations should use a sequentialisation of this parallel strategy – a bitewise strategy. This is one of the few examples of how a parallel mindset leads to new sequential algorithms.]

Surprise 2: Exchanging local models with other processors tends to create even faster convergence – a third source of speedup. (Speedup Factor 3)

The mechanism of this `extra' speedup depends on the underlying data mining technique.

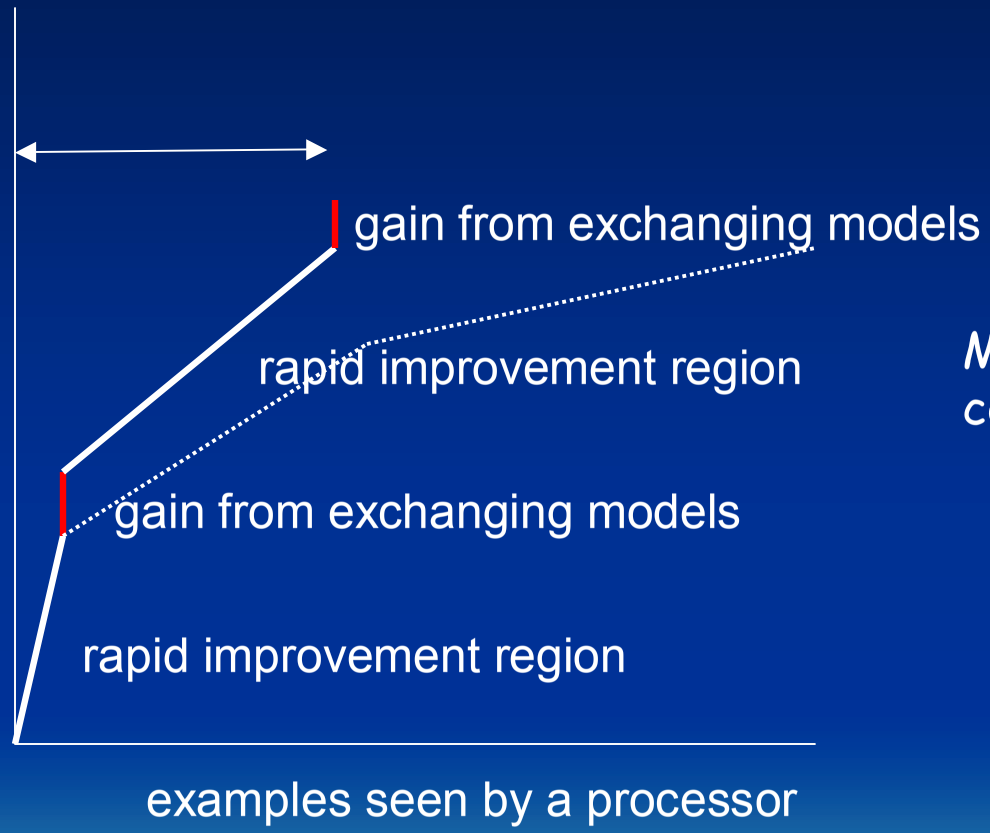
For some datasets, it's because of the shape of the quality-example curve.

If the dataset is big enough, each processor gets enough data in its partition that it moves beyond the early examples, where learning improves steeply, and starts to spend time in the next region.

So exchange models at the end of the steepest region.

Example: Neural networks – choosing the correct batch size is critical.

Quality improvement for each processor



Much faster convergence overall

For other datasets, the `extra' speedup comes because some objects can be ignored once they are accounted for by the model.

Example: Inductive logic programming – find a disjunction of concepts that explains all of the objects.

Once an object is accounted for, it does not need to be considered further. Getting *p* concepts in each round reduces the remaining examples quickly.

The overall program structure is:

Partition the dataset across p processors
Forall processors (in parallel)
  Set base model to be empty
  For q rounds
    Improve the base model using n/pq new data
      (choose n/pq to get optimal speedup behaviour)
    Total exchange of models among processors
     Produce a new base model merging models received

N.B. fit with BSP!

N.B. the structure of these algorithms goes well beyond the reductive structure assumed by other data-intensive approaches, e.g. DataCutter.

There remain interesting problems around storage management – e.g. extracting a sample without fetching every page to memory.

Distributed Data Mining

Distributed data mining is also becoming important.

Here the attributes of an object are located in different places. Perhaps they were collected via different touchpoints (store, 800 number, web site), or different channels (roaming cell phone use).

This corresponds to partitioning the dataset by columns.

It is often not possible to collect the attributes in one place because the dataset is too big; or there are jurisdictional boundaries.

Solutions require learning useful information locally in such a way that it can be combined to give a globally accurate model.

For example, a customer may seem to fit the profile of a good customer by her attributes at 1 site, but not at the others. How can we tell the true state of affairs (i.e. what the sequential algorithm would have said)?

There's only very preliminary work – e.g. Kargupta (Fourier bases, wavelets), my group (SVD).

Distributed DM is the first obvious example of an increasingly important class of applications: those that use large, immovable datasets and large computations on them.

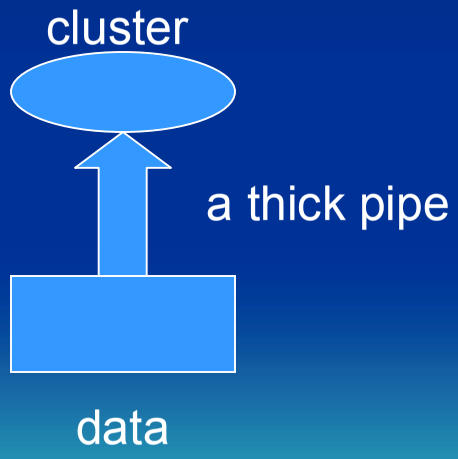
Other examples include: on-the-fly application construction from components (`cloud computing'); and mobile agent applications.

Data has inertia: it's easy to keep it in a fixed place; and it's easy to move it around: but transitions between these two states are complex, messy, and slow.

Data grids don't seem very scalable. Moving a petabyte of data is problematic, no matter what your beliefs about network cost and bandwidth. Finding a petabyte of temporary disk space for every application running on a compute server seems unrealistic. And yet petabyte datasets are very close.

Increasingly, moving data to computations is the wrong thing to do; better to move computations to data. This is the premise of the datacentric grid project.

The main new architectural requirement is that data repositories need to be fronted by large compute servers to process their data.



Find the average value of galaxy brightness in the X-ray spectrum.

There are 100 gigagalaxies known; number increasing rapidly (Hubble). Partially overlapped data about them is kept in ~30 big repositories.

Galaxies have about a kiloattribute: each repository holds some; but often scaled differently (e.g. to account for red shift, or not).

Some datasets can be downloaded; some have sql interfaces; some have home grown query interfaces.

The same object has different names (30+)

Today's solution:

Huge amount of figuring out dataset contents and properties up front.

Messy combination of downloading; generating queries; and postprocessing.

Poor solutions, and a lot of work to get any useful results (about 4 grad-student-months per result).

The requirements for datacentric grids are quite different from those of computational grids. Some of the interesting issues are:

* infrastructure for application description
* building programs (perhaps from queries)
* execution planning esp. as deltas are common
* keeping results for reuse
* describing the contents of repositories (contents and types (*cf* constructor calculus))

Summary

1. Data mining is a major application area, with huge demands for resources, and a large potential pool of users.
2. Parallelism and data mining fit well together because the local computation requirements are large, and the global communication requirements are small.
3. Distributed, grid-scale computing and data mining fit well together but moving large datasets is too expensive; so a new datacentric approach is needed.

Credits:

Sabine McConnell
Freeman Huang
Owen Rogers
Ali Roumani
Ricky Wang
Carol Yu

www.cs.queensu.ca/home/skill

ز