ISyE 6416: Computational Statistics

Lecture 1: Introduction

Prof. Yao Xie

H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

What this course is about

- Interface between **statistics** and **computer science**
- Closely related to **data mining**, **machine learning** and **data analytics**
- Aim at the design of algorithm for implementing statistical methods on computers
- computationally intensive statistical methods including resampling methods, Markov chain Monte Carlo methods, local regression, kernel density estimation, artificial neural networks and generalized additive models.

Statistics

data: images, video, audio, text, etc. sensor networks, social networks, internet, genome.

statistics provide tools to

- **model** data
  e.g. distributions, Gaussian mixture models, hidden Markov models
- **formulate** problems or ask questions
  e.g. maximum likelihood, Bayesian methods, point estimators, hypothesis tests, how to design experiments

Computing

- how do we solve these problems?
- computing: find efficient algorithms to solve them
  e.g. maximum likelihood requires finding maximum of a cost function
- "Before there were computers, there were **algorithms**. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing. "

- an algorithm: a tool for solving a well-specified computational problem
- examples:
  - The Internet enables people all around the world to quickly access and retrieve large amounts of information. With the aid of clever algorithms, sites on the Internet are able to manage and manipulate this large volume of data.
  - The Human Genome Project has made great progress toward the goals of identifying all the 100,000 genes in human DNA, determining the sequences of the 3 billion chemical base pairs that make up human DNA, storing this information in databases, and developing tools for data analysis.
  - mining the social networks

A primer of algorithms

- efficient algorithms: usual measure of efficiency is speed, i.e., how long an algorithm takes to produce its result
- complexity of algorithm:
  - The running time of an algorithm on a particular input is the number of primitive operations or "steps" executed
  - worst-case running time: the longest running time for any input of size n
  - order of growth of the running time that really interests us, e.g. $\mathcal{O}(n^3)$
  - also care of "memory" complexity

P vs. NP

- polynomial-time algorithms: on inputs of size n, the worst-case running time is $\mathcal{O}(n^k)$ for some constant k: e.g., sorting
- there are also problems that can be solved but not in time $\mathcal{O}(n^k)$ for any constant k
- we think of problems that are solvable by polynomial-time algorithms as being tractable, or easy, and problems that require superpolynomial time as being intractable, or hard

NP complete

- an interesting class of problems, called the **NP-complete** problems, whose status is unknown: no polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them
- NP-complete problems: finding clique, travel salesman

The age of big data

Machine-Learning Maestro Michael Jordan on the Delusions of Big Data and Other Huge Engineering Efforts

Big-data boondoggles and brain-inspired chips are just two of the things we're really getting wrong

By Lee Gomes
Posted 20 Oct 2014 | 19:37 GMT

Share | Email | Print

Photo-Illustration: Randi Klett

"danger" of big data

Michael Jordan: I like to use the analogy of building bridges. If I have no principles, and I build thousands of bridges without any actual science, lots of them will fall down, and great disasters will occur.

Similarly here, if people use data and inferences they can make with the data without any concern about error bars, about heterogeneity, about noisy data, about the sampling pattern, about all the kinds of things that you have to be serious about if you're an engineer and a statistician—then you will make lots of predictions, and there's a good chance that you will occasionally solve some rea interesting problems. But you will occasionally have some disastrously bad decisions. And you won't know the difference a priori. You will just produce these outputs and hope for the best.

computing needs statistics

***Spectrum*:** Do we currently have the tools to provide those error bars?

Michael Jordan: We are just getting this engineering science assembled. We have many ideas that come from hundreds of years of statistics and computer science. And we're working on putting them together, making them scalable. A lot of the ideas for controlling what are called familywise errors, where I have many hypotheses and want to know my error rate, have emerged over the last 30 years. But many of them haven't been studied computationally. It's hard mathematics and engineering to work all this out, and it will take time.

It's not a year or two. It will take decades to get right. We are still learning how to do big data well.

What He Cares About More Than Whether P = NP

***Spectrum*:** Do you have a guess about whether P = NP? Do you care?

Michael Jordan: I tend to be not so worried about the difference between polynomial and exponential. I'm more interested in low-degree polynomial—linear time, linear space. P versus NP has to do with categorization of algorithms as being polynomial, which means they are tractable and exponential, which means they're not.

I think most people would agree that probably P is not equal to NP. As a piece of mathematics, it's very interesting to know. But it's not a hard and sharp distinction. There are many exponential time algorithms that, partly because of the growth of modern computers, are still viable in certain circumscribed domains. And moreover, for the largest problems, polynomial is not enough. Polynomial just means that it grows at a certain superlinear rate, like quadric or cubic. But it really needs to grow linearly. So if you get five more data points, you need five more amounts of processing. Or even sublinearly, like logarithmic. As I get 100 new data points, it grows by two; if I get 1,000, it grows by three.

That's the ideal. Those are the kinds of algorithms we have to focus on. And that is very far away from the P versus NP issue. It's a very important and interesting intellectual question, but it doesn't inform that much about what we work on.

Example: convex relaxation

Solve $y = Ax$ where $x \in \mathbb{R}^n$ is k-sparse.

Exponential complexity

$$\begin{aligned} \text{minimize} \quad & \|x\|_0 \\ \text{subject to} \quad & y = Ax \end{aligned}$$

Polynomial complexity

$$\begin{aligned} \text{minimize} \quad & \|x\|_1 \\ \text{subject to} \quad & y = Ax \end{aligned}$$