Clustering

Prof. Ramin Zabih

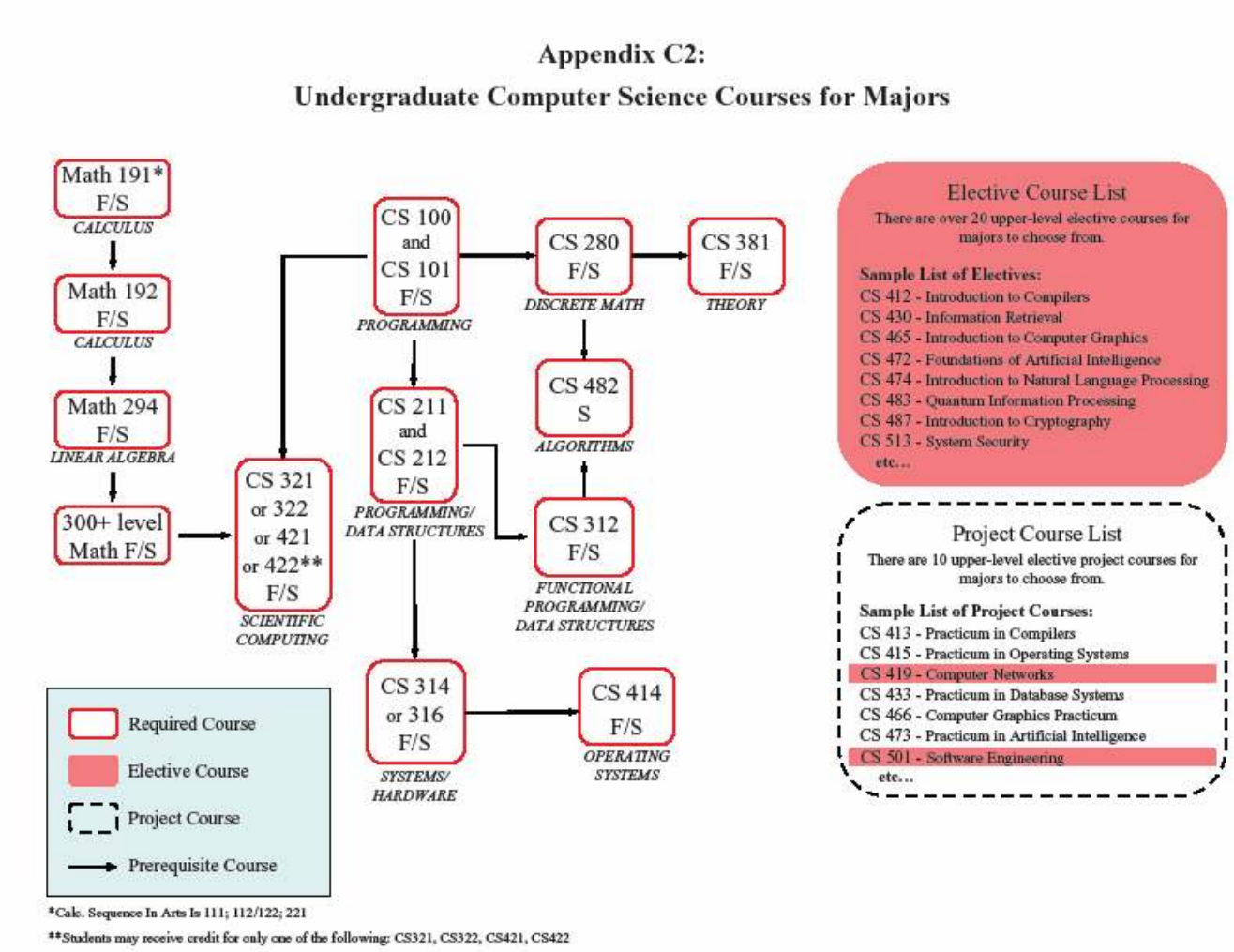
http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 5 is due Wednesday
- A6: rotation, overhead robot tracking
- Professor Halpern will give an optional lecture tomorrow (Wed) at 6PM in the RPC auditorium
  - "Knowledge and common knowledge in multi-agent systems"
- Final project proposals will be due Friday
  - Not graded, but required
  - We will post some suggestions

Life beyond CS100R?



Last topic: robot Pepsi challenge

- How can a robot tell the difference between a Coke can and a Pepsi can?
- This is actually a fundamental problem that no one knows how to solve in general
  - How does Google image search work?
    - It uses the text near to pictures
- As usual for 100R, we will simplify the problem and use it for a tour of CS

What information to use?

- Lots of things let humans tell Coke cans from Pepsi cans
  - For instance, the words
  - Also, the lettering (font)
- We'll focus on color
  - Easier to analyze

Recall: color space

- In a color image, each pixel contains 3 numbers (R, G, B)
  - Can view color images as 3 non-color images
    - A red image, green image, blue image
    - This is what the 100R software supports
- Each pixel is a point in this 3D space
  - Gray values have R=G=B

Our strategy

- We will start off with an idea of what color a Coke/Pepsi can is
  - Eventually, we will figure this out ourselves
  - By taking picture of Coke/Pepsi cans
- We'll also start off with an idea of what color the "background" is
  - Obviously, if we are trying to recognize objects using color, the background needs to be simple
    - Can't recognize a Pepsi can against a backdrop of Coke cans, in general

Background assumptions

- We will simplify life by only worrying about situations with a simple background
  - We'll put a piece of paper behind the can
  - We need the background to be relatively uniform (i.e., mostly a single color)
- We start with 3 point sets in RGB space
  - **C**, for Coke can (a few red pixels)
  - **P**, for Pepsi can (a few blue pixels)
  - **B**, for background (a few gray pixels)
- We will call these *model pixels*

Clustering issues

- Query: given some model pixels and a query pixel, classify the query pixel
- Cluster construction: divide the model pixels into groups
- Both of these are fascinating problems with a huge number of applications
  - "Data mining"
  - Recommendation systems
  - Web search

Classifying pixels

- Given an image, figure out if it's a picture of a Coke can or a Pepsi can.
  - Every pixel occupies a point in RGB space
  - Coke can pixels will tend to be close to **C**
  - Pepsi can pixels will tend to be close to **P**
  - Background pixels will tend to be close to **B**
- So we can classify the pixels!
  - Ideally, in a Coke image we will get a big "blob" of pixels that look like **C**, surrounded by pixels that look like **B**

1-nearest neighbor

- What is the computational problem?
- Can figure out, for a given pixel P = (r,g,b), which of **C**,**P**,**B** it is closest to
  - Is the nearest model pixel in **C**, **P** or **B**?
  - This algorithm is called 1-nearest neighbor
  - Figure out what you are closest to
    - Similar to nearest-neighbor interpolation
- But we have to do this many times
  - About 100,000 times or so

1-NN for color recognition

- A few model pixels for Coke cans (red), Pepsi cans (blue), background (gray)
  - Denote these sets as **C**, **P**, **B**
- Algorithm: for each pixel P in the new image, find the closest model pixel
  - Measure distance in 3D (RGB space)
  - This gives us the class of P
  - Make a decision based on this classification of each pixel
- Problem: find the closest model pixel for about 100,000 different choices of P

Some really bad solutions

- Idea: sort all the model pixels, by distance from our input pixel P
  - We don't need to sort, we just need the min
  - But we still have to do this 100,000 times
- Idea: move from P "radially" in RGB space
  - See if there is a model pixel at distance 1 from P, then distance 2, etc.
- Which one of these bad solution is worse?
  - Answer: it depends, but good answers do variants of bad idea #1

Solution: pre-processing

- Recall: finding when a particular change was made to a document
  - Many queries sharing same data
  - Solution: sort the versions by modification date, then use binary search
- Can we figure out how to process the data so that we don't need to find the distance to every model pixel?
  - Even if this processing is expensive, it can pay off, since we will do tons of queries
  - Sometimes call on-line vs. off-line processing

How to pre-process?

- There are many ways to do this
  - We want to not bother to compute the distance to model pixels that are very far away
  - The "right" answer involves Voronoi diagrams, which are (probably) too complex for 100R
- We can do something simpler that works pretty well, though
  - We'll do our examples in 2D colorspace
  - Suppose we break our space into big squares and keep track of what model pixels are in each square. Can this help us?