Clustering and greedy algorithms

Prof. Noah Snavely
CS1114
http://cs1114.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- A6 due tomorrow
  - Please sign up for demo sessions
  - You will also turn in your code this time (turnin due Monday)
- Prelim 3 next Thursday, 4/30 (last lecture)
  - Will be comprehensive, but focus on most recent material
  - Review session Wednesday at 7pm

Administrivia

- Course evals:

  http://www.engineering.cornell.edu/CourseEval/

  Please fill these out – we appreciate your feedback!

  Also, filling out a course evaluation is worth 2 points*

* The College of Engineering will inform the course instructor about who has completed the course evaluation, *but the instructor will **not** see your individual response*. The course instructor will see only the compiled summary of the evaluation *after* grades have been submitted to the registrar when the semester ends.

New problem

- Suppose we are given a bunch of different texts, but we don't know who wrote any of them
- We don't even know how many authors there were

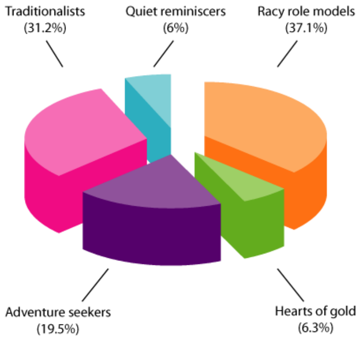
- How can we figure out:
  1. The number of authors
  2. Which author wrote each text

Clustering

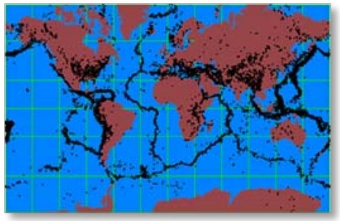
- Idea: Assume the statistics of each author are similar from text to text
- Assign each text to a group such that:
  - The texts in each group are similar
  - The texts in different groups are different

- This problem is known as *clustering*

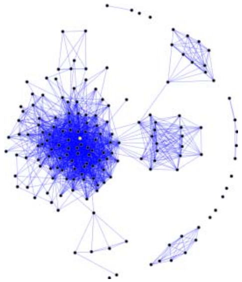
Applications of clustering



- Economics or politics
  - Finding similar-minded or similar behaving groups of people (market segmentation)
  - Find stocks that behave similarly



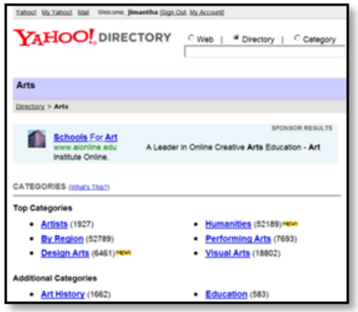
- Spatial clustering
  - Earthquake centers cluster along faults



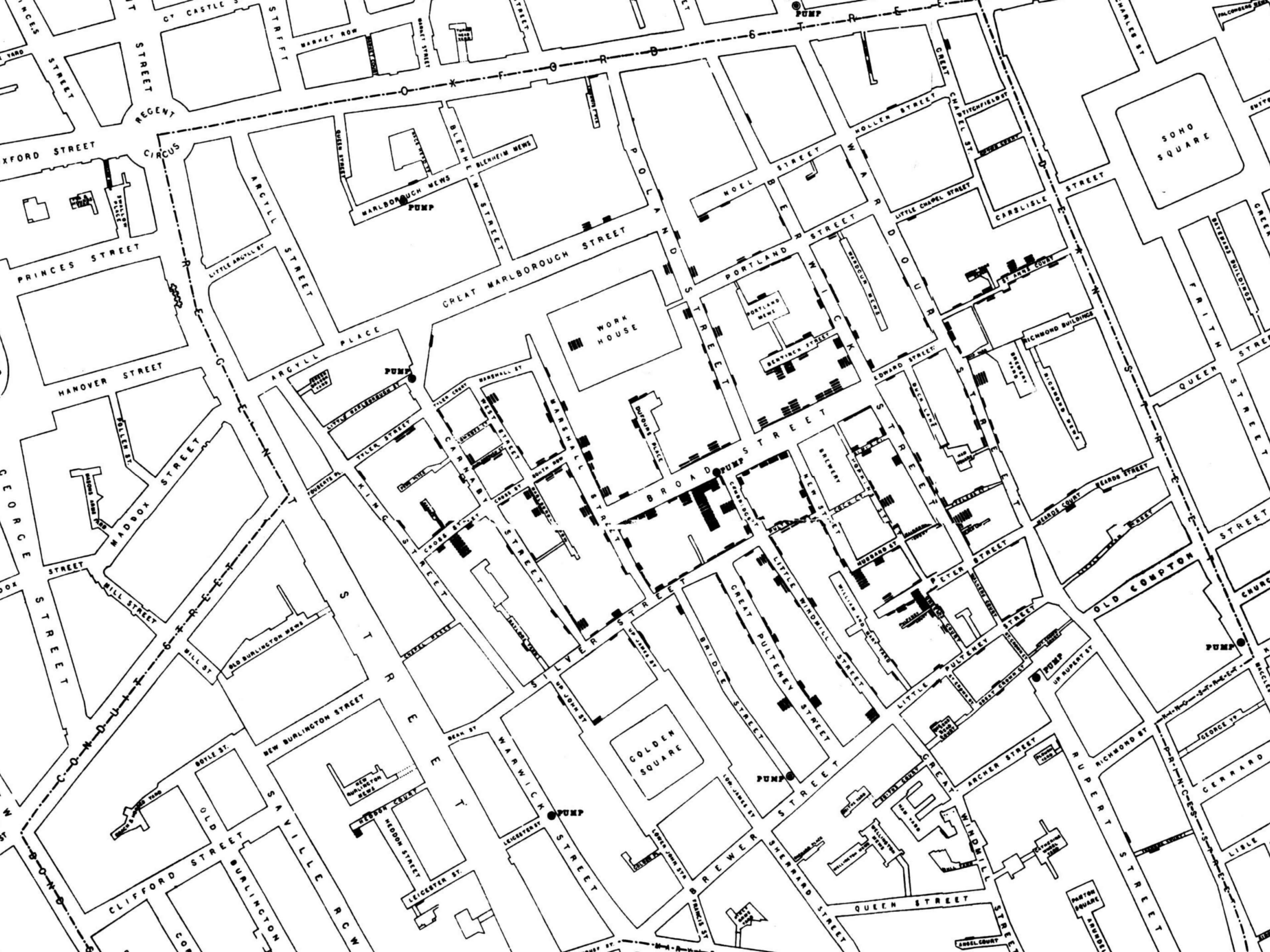
- Social network analysis
  - Recognize communities of similar people

Applications of clustering

- Image segmentation
  - Divide an image into components representing the same object (thresholding + connected components is a very simple version of this)



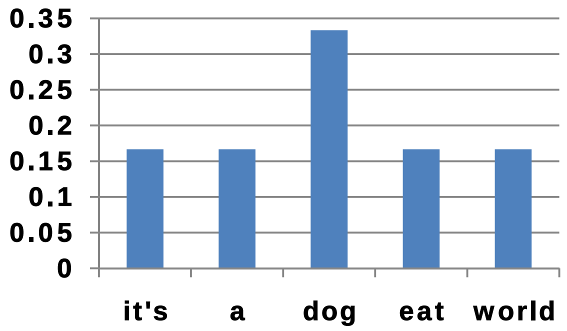
- Classify documents for web search
  - Automatic directory construction (like Yahoo!)


OXFORD STREET
REGENT CIRCUS
PRINCES STREET
HANOVER STREET
ARGYLL STREET
LITTLE ARGYLL ST
ARGYLL PLACE
MARLBOROUGH MEWS
BLENHEIM MEWS
BLENHEIM STREET
GREAT MARLBOROUGH STREET
POLAND STREET
NOEL STREET
PORTLAND STREET
PORTLAND MEWS
BENTINCK STREET
BERWICK STREET
WARDOUR STREET
HOLLEN STREET
LITTLE CHAPEL STREET
CARLISLE STREET
ST ANNE COURT
RICHMOND BUILDINGS
RICHMOND MEWS
DEAN STREET
SOHO SQUARE
FRITH STREET
QUEEN STREET
WORK HOUSE
BROAD STREET
MARSHALL STREET
CARNABY STREET
KING STREET
TYLER STREET
CROSS STREET
EDWARD STREET
PETER STREET
OLD COMPTON STREET
SILVER STREET
GREAT PULTENEY STREET
LITTLE PULTENEY STREET
LITTLE WINDMILL STREET
GREAT WINDMILL STREET
BRIDLE STREET
GOLDEN SQUARE
BREWER STREET
WARWICK STREET
REGENT STREET
CONDUIT STREET
MADDOX STREET
GEORGE STREET
MILL STREET
OLD BURLINGTON MEWS
NEW BURLINGTON STREET
NEW BURLINGTON MEWS
BOYLE ST.
OLD BURLINGTON STREET
SAVILLE ROW
CLIFFORD STREET
HEDDON COURT
HEDDON STREET
LEICESTER ST.
QUEEN STREET
SHERRARD STREET
ARCHER STREET
RUPERT STREET
RICHMOND ST
PRINCES STREET
KING STREET
GERRARD
LISLE
PUMP

Clustering

- We will assume that each item to be clustered is represented as a (possibly long) vector
- For documents, it might be a frequency distribution of words (a histogram)

"it's a dog eat dog world" →



- For earthquakes, the vector could be the latitude/longitude of the quake

Clustering

- The distance between two items is the distance between their vectors

- We'll also assume for now that we know the number of clusters

Example

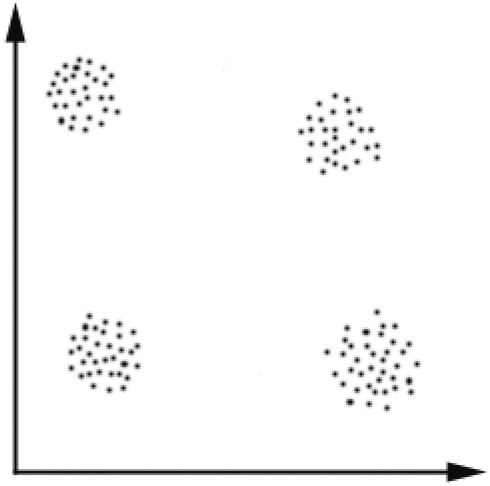


Figure from Johan Everts

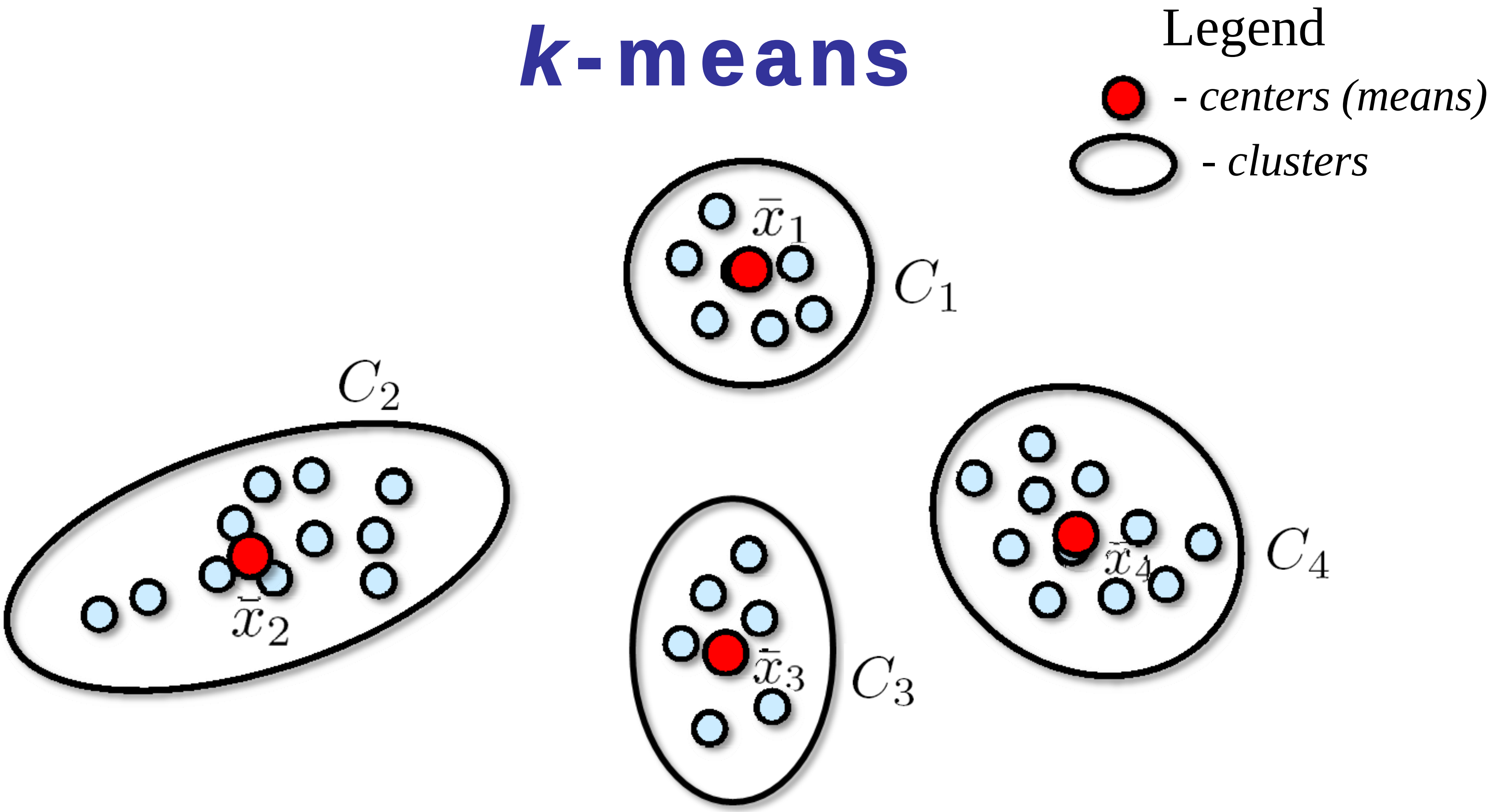
Clustering algorithms

- There are many different approaches
- How is a cluster is represented?
  - Is a cluster represented by a data point, or by a point in the middle of the cluster?
- What algorithm do we use?
  - An interesting class of methods uses graph partitioning
  - Edge weights are distances

One approach: *k*-means

- Suppose we are given *n* points, and want to find *k* clusters
- We will find *k* cluster centers (or means), and assign each of the *n* points to the nearest cluster center $\bar{x}_j$
  - A *cluster* is a subset of the *n* points, called C_j
  - We'll call each cluster center a *mean*

k-means



- How do we define the best *k* means?

k-means

- Idea: find the centers that minimize the sum of squared distances to the points

- Objective:

Given input points $x_1, x_2, x_3, \ldots, x_n$, find the clusters $C_1, C_2, \ldots C_k$ and the cluster centers $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_k$ that minimize

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \bar{x}_j|^2$$

Optimizing *k*-means

Given input points $x_1, x_2, x_3, \ldots, x_n$, find the clusters $C_1, C_2, \ldots C_k$ and the cluster centers $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_k$ that minimize

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \bar{x}_j|^2$$

- How do we minimize this objective?
- At first, this looks similar to a least squares problem
  – but we're solving for cluster membership along with cluster centers

Optimizing *k*-means

Given input points $x_1, x_2, x_3, \ldots, x_n$, find the clusters $C_1, C_2, \ldots C_k$ and the cluster centers $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_k$ that minimize

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \bar{x}_j|^2$$

- The bad news: this is not a convex optimization
- The worse news: it is practically impossible to find the global minimum of this objective function
  - no one has ever come up with an algorithm that is faster than exponential time (and probably never will)
- There are many problems like this (called *NP-hard*)

Optimizing *k*-means

- It's possible to prove that this is a hard problem (you'll learn how in future CS courses – it involves reductions)
- What now?
- We shouldn't give up… it might still be possible to get a "pretty good" solution

Possible algorithms

1. Guess an answer until you get it right
2. Guess an answer, improve it until you get it right
3. Magically compute the right answer

- Sometimes we can tell when we have the right answer (e.g., sorting, minimizing convex functions)
- Sometimes we *can't* tell without checking every single possibility (e.g., *k*-means)

Possible algorithms

1. Guess an answer until you get it right
2. Guess an answer, improve it until you get it right
3. Magically compute the right answer

- For *k*-means, none of these algorithms work (we can't check if we're right, no magic formula for the right answer)
- What do we do?

Possible algorithms

- We can adapt algorithms 1 and 2:

1. Randomly select *k* means many times, choose the one with the lowest cost
2. Randomly select *k* means, improve them locally until the cost stops getting lower (Lloyd's algorithm – we'll come back to this)

- Can't really prove how good the answer is
- We will look at another possibility:
  - Build the solution one mean at a time

Greedy algorithms

- Many CS problems can be solved by repeatedly doing whatever seems best at the moment
  - I.e., without needing a long-term plan
- These are called greedy algorithms
- Example: hill climbing for convex function minimization
- Example: sorting by swapping out-of-order pairs (e.g., bubble sort)

Making change

- For US currency (quarters, dimes, nickels, pennies) we can make change with a greedy algorithm:
  1. While remaining change is > 0
  2. Give the highest denomination coin whose value is >= remaining change

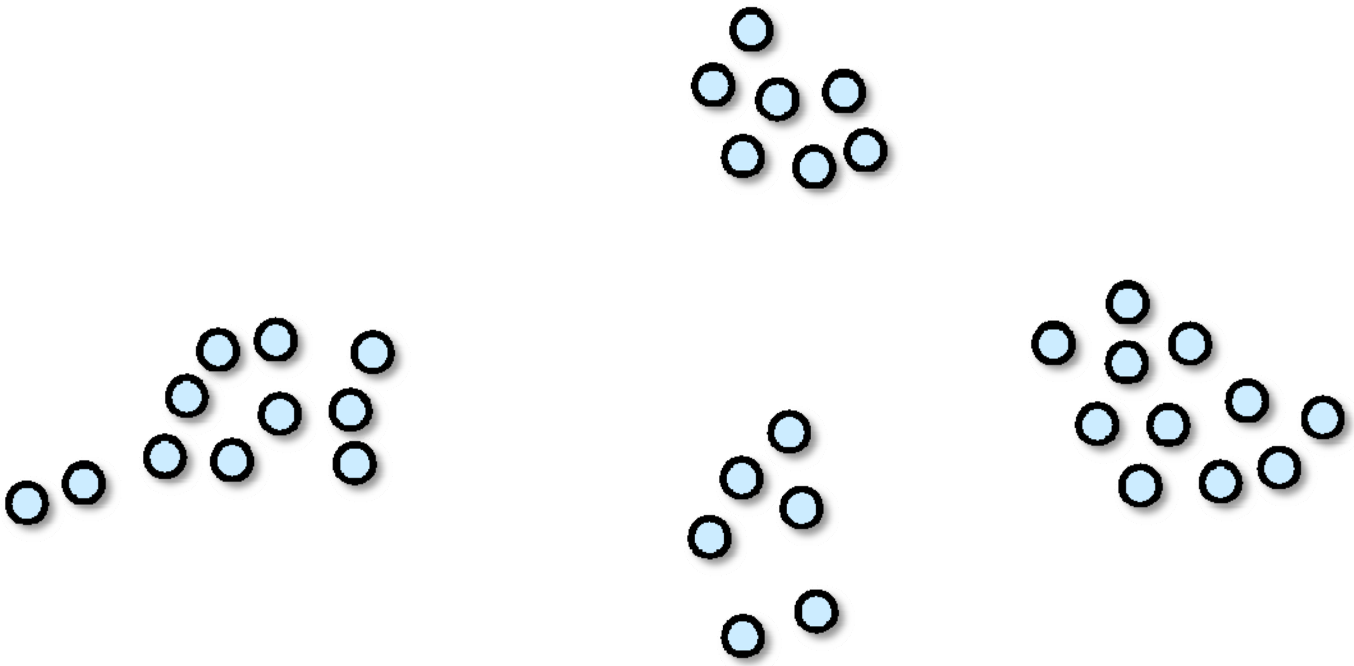
41 cents: 

- What if our denominations are 50, 49, and 1?
  – How should we give back 98 cents in change?
  – Greedy algorithms don't always work…
  – (This problem requires more advanced techniques)

A greedy method for *k*-means

- Pick a random point to start with, this is your first cluster center
- Find the farthest point from the cluster center, this is a new cluster center
- Find the farthest point from any cluster center and add it
- Repeat until we have *k* centers

A greedy method for *k*-means



A greedy method for *k*-means

- Unfortunately, this doesn't work that well
- The answer we get could be **much** worse than the optimum

The *k*-centers problem

- Let's look at a related problem: *k*-centers
- Find *k* cluster centers that minimize the *maximum* distance between any point and its nearest center
  - We want the worst point in the worst cluster to still be good (i.e., close to its center)
  - Concrete example: place *k* hospitals in a city so as to minimize the maximum distance from a hospital to a house

k-centers

- What objective function does this correspond to?

Given input points $x_1, x_2, x_3, \ldots, x_n$, find the clusters $C_1, C_2, \ldots C_k$ and the cluster centers $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_k$ that minimize

$$\max_{j=1}^{k} \max_{x_i \in C_j} \left| x_i - \bar{x}_j \right|^2$$

- We can use the same greedy algorithm

An amazing property

- This algorithm gives you a solution that is no worse than twice the optimum
- Such results are sometimes difficult to achieve, and the subject of much research
  - Mostly in CS6810, a bit in CS4820
  - You can’t find the optimum, yet you can prove something about it!
- Sometimes related problems (e.g. *k*-means vs. *k*-centers) have very different guarantees

Next time

- More on clustering