CAS CS 565, Data Mining

Course logistics

- Course webpage:
  - http://www.cs.bu.edu/~evimaria/cs565-16.html
- Schedule: Mon – Wed, 4:00-5:30
- Instructor: Evimaria Terzi, evimaria@cs.bu.edu
- Office hours: Mon 5:30-7pm, Wed 9:30pm-11:00am (or by appointment)
- Join the class on piazza to get updates

Topics to be covered (tentative)

- What is data mining?
- Distance functions
- Finding similar entities
- Dimensionality reduction
- Clustering
- Classification
- Link analysis ranking
- Covering problems and submodular function optimization
- Applications: Web advertising, recommendation systems

Course workload

- Two programming assignments (25%)
- Three problem sets (25%)
- Midterm exam (20%)
- Final exam (30%)
- Late assignment policy: 10% per day up to three days; credit will be not given after that
- Incompletes will not be given

Learn what you (don’t)know

The main goal of the class is for you to get to know what you know and what you don’t know (20% rule)

Prerequisites

- Basic algorithms: sorting, set manipulation, hashing
- Analysis of algorithms: O-notation and its variants, perhaps some recursion equations, NP-hardness
- Programming: some programming language, ability to do small experiments reasonably quickly
- Probability: concepts of probability and conditional probability, expectations, binomial and other simple distributions
- Some linear algebra: e.g., eigenvector and eigenvalue computations

Above all

- The goal of the course is to learn and enjoy
- The basic principle is to ask questions when you don't understand
- Say when things are unclear; not everything can be clear from the beginning
- Participate in the class as much as possible
- We will do a lot of thinking together...better to think with company

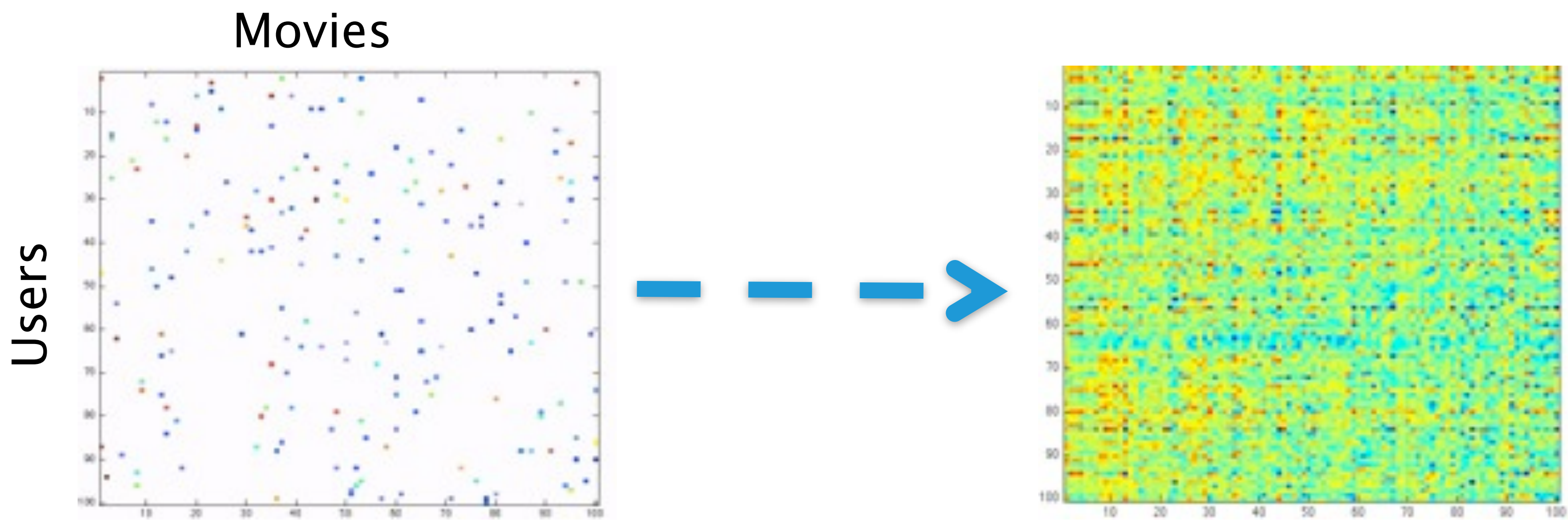
Introduction to data mining

- Why do we need data analysis?
- What is data mining?
- Examples where data mining has been useful
- Data mining and other areas of computer science and statistics
- Some (basic) data-mining tasks

There are lots of data around

- Web
- Online social networks
- Recommendation systems
- Wikipedia
- Genomic sequences: 310^9 nucleotides per individual for 1000 people --> 310^12 nucleotided...+ medical history + census information

Example: Netflix data



Want to predict **all** ratings, but we know only 1% of the entries!

Data complexity

- Multiple types of data: tables, time series, images, graphs, etc
- Spatial and temporal aspects
- Large number of different variables
- Lots of observations → large datasets

What can data-mining methods do?

- Rank web-query results
  - What are the most relevant web-pages to the query: "Student housing BU"?
- Find groups of entities that are similar (clustering)
  - Find groups of facebook users that have similar friends/interests
  - Find groups amazon users that buy similar products
  - Find groups of walmart customers that buy similar products
- Find good recommendations for users
  - Recommend amazon customers new books
  - Recommend facebook users new friends/groups

Goal of this course

- Describe some problems that can be solved using data-mining methods
- Discuss the intuition behind data-mining methods that solve these problems
- Illustrate the theoretical underpinnings of these methods
- Show how these methods can be useful in practice

Data mining when datasets are large

- Time and space complexity are important

- Even for very simple tasks

Some simple data-analysis tasks

- Given a stream or set of numbers (identifiers, etc)
- How many numbers are there?
- How many distinct numbers are there?
- What are the most frequent numbers?
- How many numbers appear at least K times?
- How many numbers appear only once?
- etc

Finding the majority element

- A neat problem
- A stream of identifiers; one of them occurs more than 50% of the time
- How can you find it using no more than a few memory locations?
- Suggestions?

Finding the majority element

- A = first item you see; count = 1
- **for** each subsequent item B

```
    if (A==B) count = count + 1
    else
        count = count – 1
        if (count == 0)  A=B; count = 1
  endfor

  return A
```

- Why does this work correctly?

Finding the majority element

- A = first item you see;
- count = 1
- **for** each subsequent item B

```
    if (A==B)
        count = count + 1
    else
        count = count - 1
        if (count == 0)
            A=B;
            count = 1
endfor
return A
```

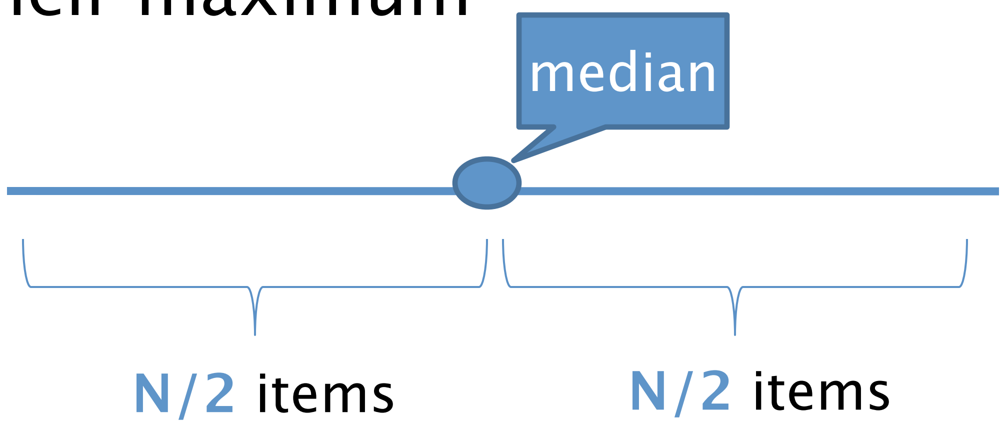
- **Basic observation:** Whenever we discard element **u** we also discard a unique element **v** different from **u**

Finding a number in the top half

- Given a set of N numbers (N is very large)
- Find a number x such that x is *likely* to be larger than the median of the numbers
- Simple solution
  - Sort the numbers and store them in sorted array A
  - Any value larger than A[N/2] is a solution
- Other solutions?

Finding a number in the top half **efficiently**

- A solution that uses small number of operations
  - Randomly sample **K** numbers from the file
  - Output their maximum



- Failure probability **(1/2)^K**

Sampling a sequence of items

- **Problem:** Given a sequence of items P of size N form a **random sample** S of P that has size n ($n<N$) → sampling without replacement

- What does random sample mean?
  - Every element in P appears in S with probability n/N
  - Equivalent as if you generate a **random permutation** of the N elements and take the **first** n elements of the permutation

Sampling algorithm v.0.

- R = {} // empty set
- **for** i=1 **to** n

```
        rnd = Random([1…N])
        while (rnd in R)
                rnd = Random([1…N])
        endwhile
        R = R U {rnd}
        S[i] = P[rnd]
endfor
return S
```

- Running time?
- The algorithm assumes that S and its size are known in advance!

Sampling algorithm v.1.

- **Step 1:** Create a random permutation π of the elem

Can you do **Step 1** in linear time?

- **Step 2:** Return the first n elements of the p rmutation, $S[i] = \pi[i]$, for $(1 \leq i \leq n)$

You can do **Step 2** in linear time☺

Creating a random permutation in linear time

- **for** i=1…N **do**

  j = Random([1…i-1])

  swap P[i] with P[j]

  endfor
- Is this really a random permutation? (see CLR for the proof)
- It runs in linear time

Sampling algorithm v.1.

- **Step 1:** Create a **random permutation** π of the elements in P
- **Step 2:** Return the first n elements of the permutation, $S[i] = \pi[i]$, for $(1 \leq i \leq n)$

- The algorithm works in **linear time** **$O(N)$**
- The algorithm assumes that P **is known in advance**
- The algorithm makes **2 passes** over the data

Sampling algorithm v.2.

- **for** i = 1 to n

  S[i] = P[i]

  endfor

- t = n+1

- **while** P has more elements

  rnd = Random([1…t])

  if (rnd <= n)

  {S[rnd] = P[t]}

  t = t + 1

  endwhile

Correctness proof

- At iteration **t+1** a **new** item is included in the sample with probability **n/(t+1)**
- At iteration **(t+1)** an **old** item is kept in the sample with probability **n/(t+1)**
  - **Inductive argument:** at iteration **t** the old item was in the sample with probability **n/t**
  - **Pr(old item in sample at t+1) = Pr(old item was in sample at t) x (Pr(rnd >n) + Pr(rnd<=n) x Pr(old item was not chosen for eviction))**

= n/t((t+1-n)/(t+1)+n/(t+1)x(1-1/n))

= n/(t+1)

Sampling algorithm v.2.

- **for** i = 1 to n

  S[i] = P[i]

  endfor

- t = n+1

- **while** P has more elements {

  rnd = Random([1…t])

  if (rnd <= n)

  {S[rnd] = P[t]}

  t = t + 1

  endwhile

Advantages

- Linear time
- **Single pass** over the data
- **Any time;** the length of the sequence need not be known in advance