CS 422 Data Mining

Lecture 9

October 25, 2018

- Midterm

Midterm

- Decision tree induction algorithms that we discussed use what strategy to grow the tree? Check all that apply. 5 points
  - Global optimization
  - X Greedy strategy
  - X Series of locally optimal decisions about what attribute to use for splitting data

- The decision tree induction algorithm always produces the best possible tree - true or false? 2 points
  - True
  - X False

Midterm

- Consider the following case during your tree induction process:
  - There are 4 records associated with node Dt. There are 2 records from class 1 and 2 records from class 2. We are using the parameter *minimum number of records per leaf node* = 3.
  - Explain how the parameter *minimum number of records per leaf node* is used in the tree induction process. Explain how the tree induction algorithm will proceed in this situation using the steps of the tree induction process.

Decision Tree Induction

- **Algorithm 4.1 Decision tree induction algorithm**
  - E is the set of the training records
  - F is the set of labels
  - TreeGrowth(E,F)
    - If stopping_cond(E,F) = true then
      - Leaf = createNode();
      - Leaf.label = classifyNode(E)
      - Retrun leaf
    - Else
      - Root = createNode()
      - Root.test_cond = find_best_split(E,F)
      - V = { v | v is a possible output of root.test_cond}
      - For each v in V
        - Ev = { e | root.test_cond(e) = v and e in E}
        - Child = TreeGrowth(Ev,F)
        - Add child as descendant of root and lable edge (root->child) as v
      - end for
    - End if

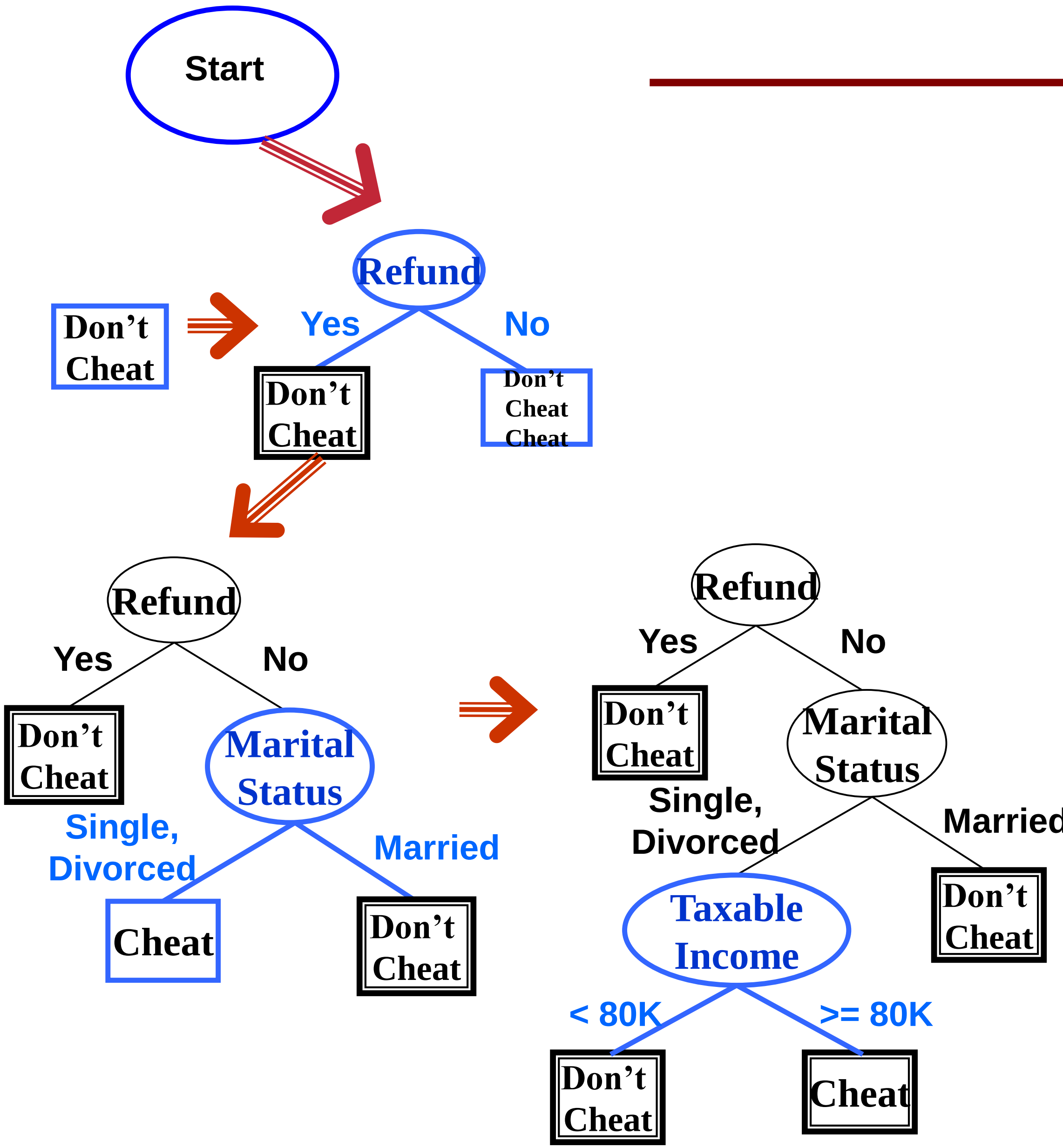
CreateNode() exteds the tree with a new node
The node has either the node.test_cond or a node.leaf.

Find_best_split() determines which attribute to select as the test condition (Gini index)

Classify() assigns the class label

Stopping_cond is used to terminate the node creation

Hunt's Algorithm



Hunt's Algorithm

– If D_t contains records that belong to the same class y_t, then t is a leaf node labeled as y_t

– If D_t is an empty set,

then t is a leaf node labeled by the default class, y_d

– If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets.

- Recursively apply the procedure

to each subset.

- Is it always the goal to build a tree that has zero training error? Explain why in a few sentences.
    - No. Overfitting

Midterm

- Use the data from the table below. There are 8 records, 3 attributes (A, B, C) and two class labels (+,-). Use IG as the impurity measure. Discuss the contingency tables below and how even from them we can see that is the best split attribute.

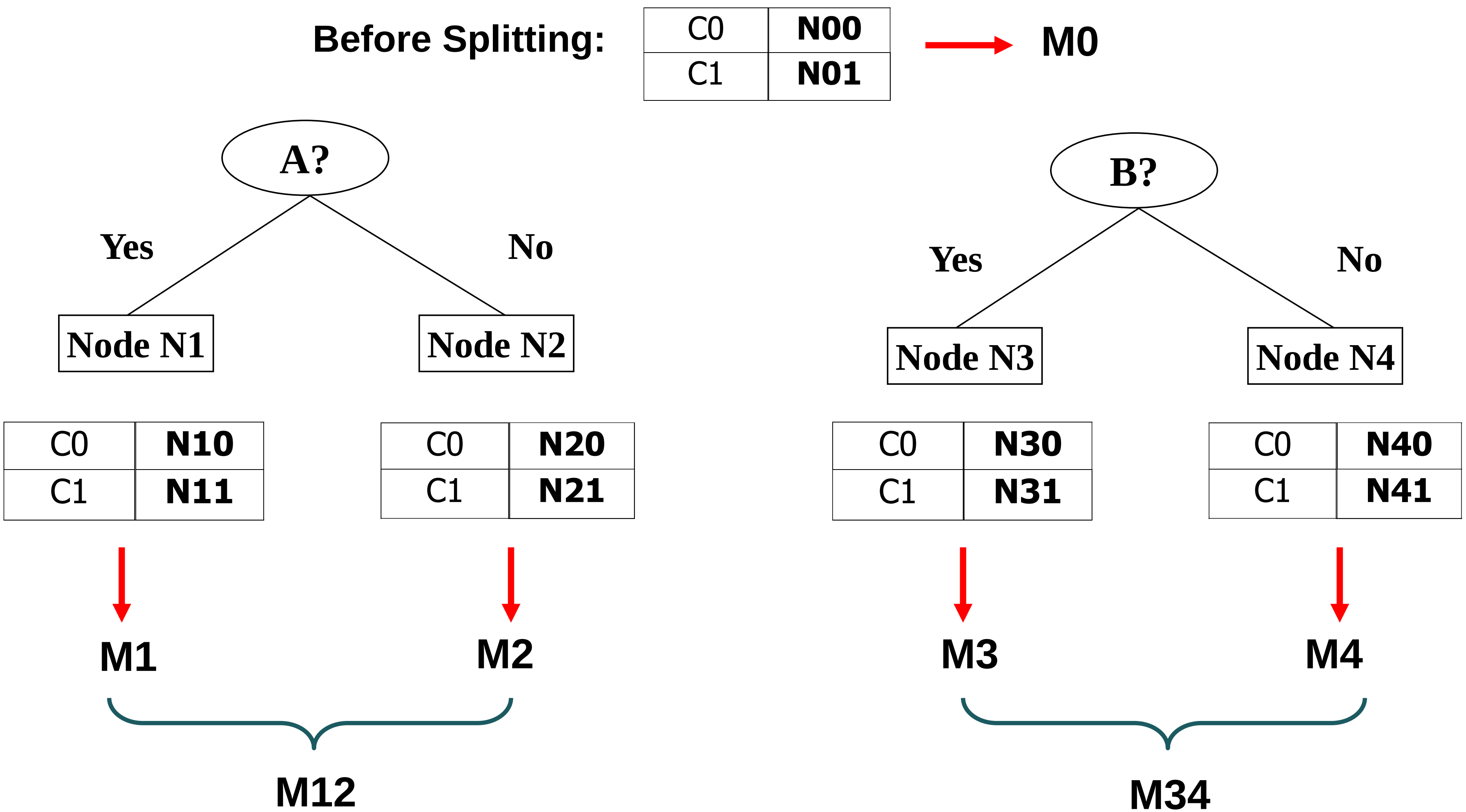
	$A = T$	$A = F$
+	25	25
−	0	50

	$B = T$	$B = F$
+	30	20
−	20	30

	$C = T$	$C = F$
+	25	25
−	25	25

- \- Write the pseudocode and explain briefly how you will apply every step of the decision tree induction process to build the **FIRST** level of a decision tree based on the provided information above. At what step and how will you use the IG as impurity measure?

How to Find the Best Split



Gain = M0 – M12 vs M0 – M34

Examples for computing Entropy

$$Entropy(t) = -\sum_j p(j|t)\log_2 p(j|t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6   P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Splitting Based on Information Theory

- Information Gain:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node, p is split into k partitions;

n_i is number of records in partition i

- Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
- Used in ID3 and C4.5
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

$$E_{orig} = 1 - \max(\frac{50}{100}, \frac{50}{100}) = \frac{50}{100}.$$

After splitting on attribute *A*, the gain in error rate is:

	$A = T$	$A = F$
$+$	25	25
$-$	0	50

$$E_{A=T} = 1 - \max(\frac{25}{25}, \frac{0}{25}) = \frac{0}{25} = 0$$

$$E_{A=F} = 1 - \max(\frac{25}{75}, \frac{50}{75}) = \frac{25}{75}$$

$$\Delta_A = E_{orig} - \frac{25}{100}E_{A=T} - \frac{75}{100}E_{A=F} = \frac{25}{100}$$

	$B = T$	$B = F$
$+$	30	20
$-$	20	30

$$E_{B=T} = \frac{20}{50}$$

$$E_{B=F} = \frac{20}{50}$$

$$\Delta_B = E_{orig} - \frac{50}{100}E_{B=T} - \frac{50}{100}E_{B=F} = \frac{10}{100}$$

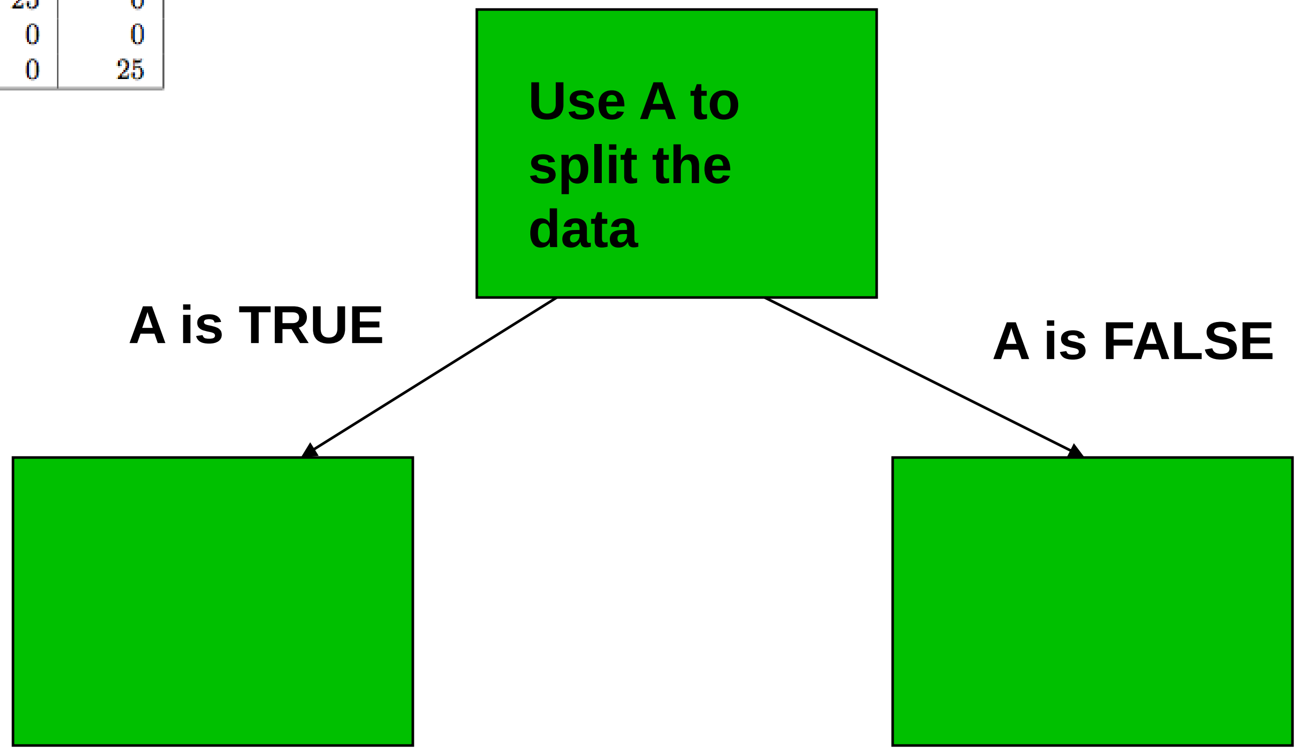
	$C = T$	$C = F$
$+$	25	25
$-$	25	25

$$E_{C=T} = \frac{25}{50}$$

$$E_{C=F} = \frac{25}{50}$$

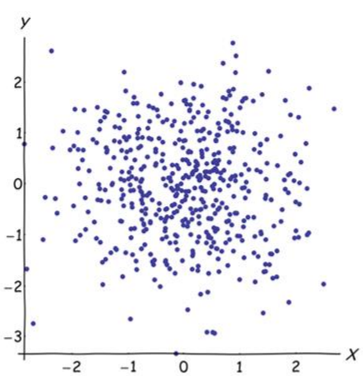
$$\Delta_C = E_{orig} - \frac{50}{100}E_{C=T} - \frac{50}{100}E_{C=F} = \frac{0}{100} = 0$$

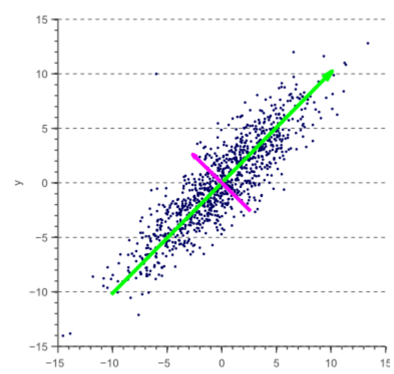
A	B	C	Number of Instances +	Number of Instances −
T	T	T	5	0
F	T	T	0	20
T	F	T	20	0
F	F	T	0	5
T	T	F	0	0
F	T	F	25	0
T	F	F	0	0
F	F	F	0	25



PCA

- Explain what you think is the actual (intrinsic) dimensionality of the data in the plot 1 and 2 below and if you think reducing the dimension from 2 to 1 will keep most of the information (discuss the correlation between the dimensions). 5 points





Midterm

- Give a definition of PCA in your own words. Explain how the eigenvectors are used to compute the low dimensional representation of the data with PCA.

PCA is "an orthogonal linear transformation that transfers the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (*first principal component*), the second greatest variance lies on the second coordinate (*second principal component*), and so on."

- each eigenvector consists of p values which represent the "contribution" of each variable to the principal component axis
- eigenvectors are uncorrelated (orthogonal)
  - their cross-products are zero.

Eigenvectors

	u_1	u_2
X_1	0.7291	-0.6844
X_2	0.6844	0.7291

$$0.7291*(-0.6844) + 0.6844*0.7291 = 0$$

The Algebra of PCA

- coordinates of each object i on the kth principal axis, known as the scores on PC k, are computed as

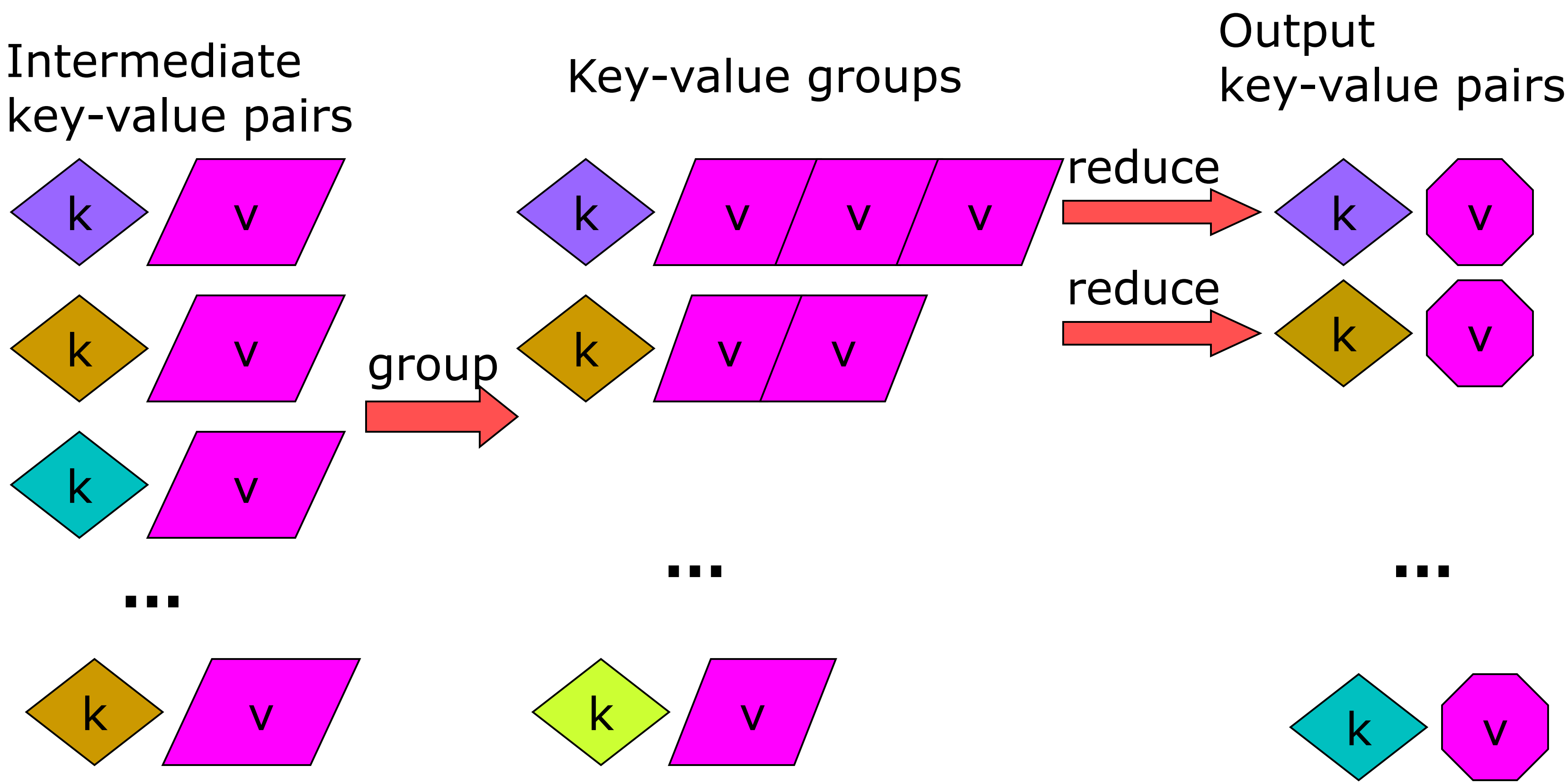
$$z_{ki} = u_{1k}x_{1i} + u_{2k}x_{2i} + \cdots + u_{pk}x_{pi}$$

- where Z is the n x k matrix of PC scores, X is the n x p centered data matrix and U is the p x k matrix of eigenvectors.

Midterm

- You are given a large number of files containing positive integers. Design the MapReduce process to compute the number of even integers in all files.

MapReduce: The Reduce Step



Midterm

- MapReduce Job

- Mapper // Compute the count per line
  - Input: key = fileID_lineN, value = line
  - Output: key = fileID, value = sum per line

```
Input< fileID_lineNumber,line>
        Int sum = 0;
        String[] tokens = line.split("\n"); // Tokenize the line
        For( String token: tokens){
                Int value = Integer.parseString(token);
                If(value % 2 == 0){// Compute the count for the line
                        Sum++;
                }}
        Emit(fileID, sum);
```

- Combiner 1 // Compute the sum per file
  - Input: key = fileID, value = array of sum values per line
  - Output: key = 1, value = sum per file

```
Input<fileID, Integer[] values>
        Int sum = 0;
        For(Integer value : values){
        // Compute the sum for the file
                Sum+=value;
                }
        Emit(1,sum);
```

- MapReduce Job
- Combiner 2
  - Input: key = 1, value = array of sum values per file
  - Output: key = 1, value = sum per file

```
Input<1, Integer[] values>
        Int sum = 0;
        For(Integer value : values){
        // Compute the sum for the file
                Sum+=value;
                }
        Emit(1,sum);
```

- Reducer // Compute the sum per file
  - Input: key = 1, value = array of sum values per line
  - Output: key = 1, value = sum per file

```
Input<1, Integer[] values>
        Int sum = 0;
        For(Integer value : values){
        // Compute the sum for the file
                Sum+=value;
                }
        Emit(1,sum);
```

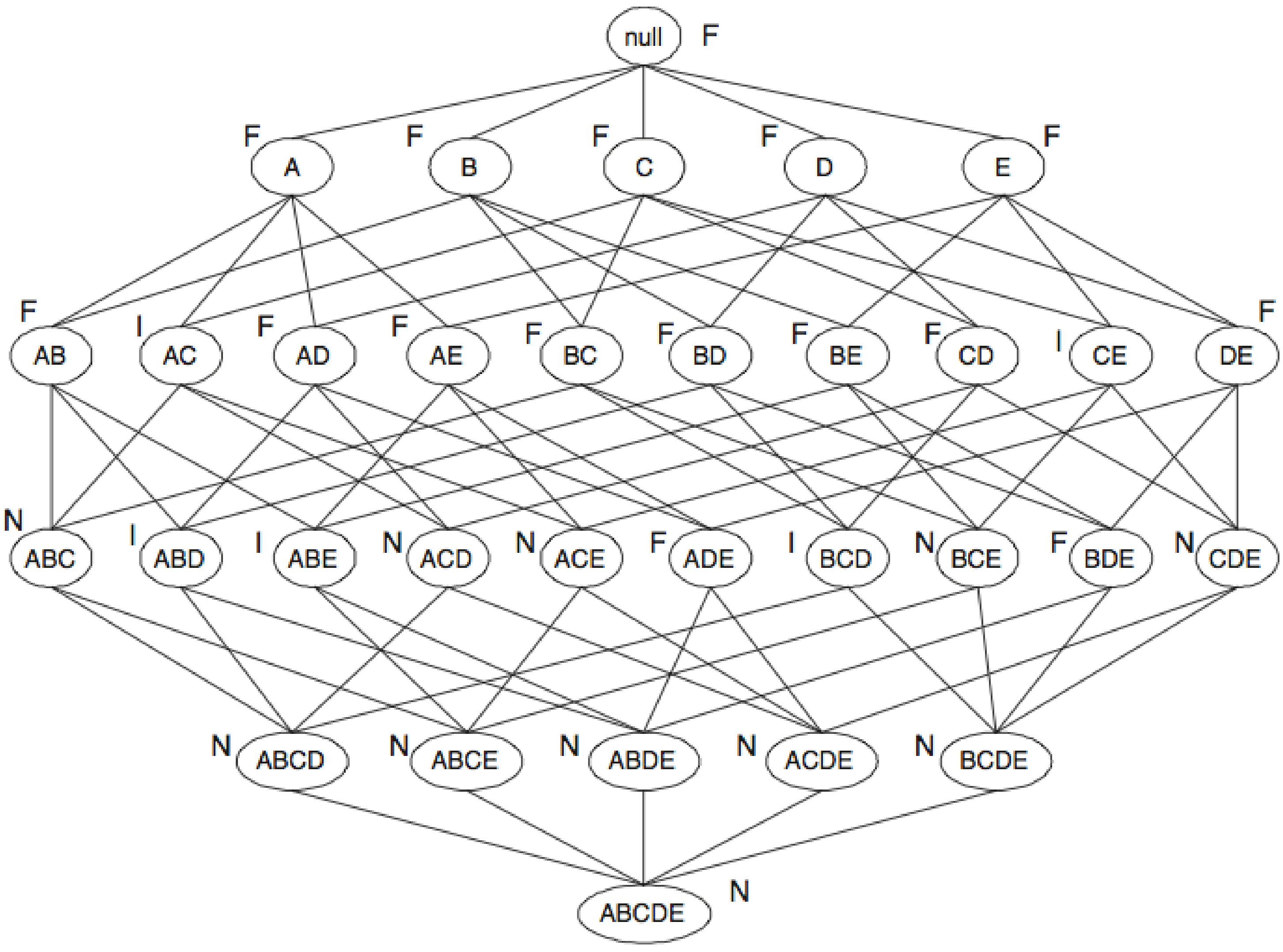
Midterm

- **Association Rules**
- State the Apriori principle. Explain the difference between the Apriori principle and the Apriori algorithm.

The lattice

- N: If the itemset is not considered to be a candidate itemset by the Apriori algorithm.
- F: If the candidate itemset is found to be frequent by the Apriori algorithm.
- I: If the candidate itemset is found to be infrequent after support counting.

Fill in the missing labels N, F or I into the green boxes. Note that you don’t need counts to answer this question. Explain your anwers. 6



Label in box 1 (ACD) = N
Label in box 2 (ACE) = N
Label in box 3 (ABDE) = N
Label in box 4 (ABCDE) = N

- 5.5) What are the two places where we apply the apriori principle during the frequent itemset generation process?

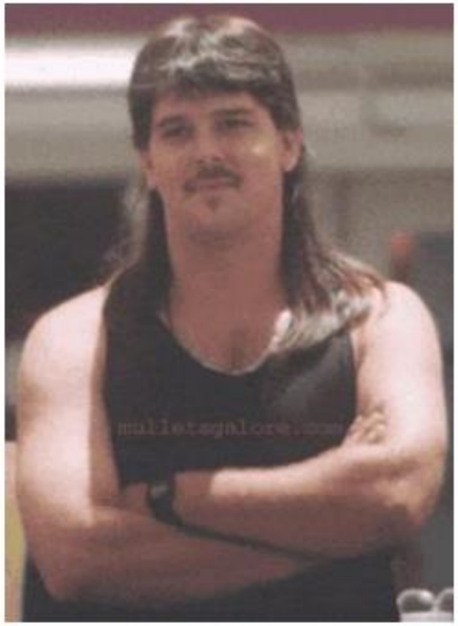
Apriori Algorithm

- Method:
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

- Recommender Systems

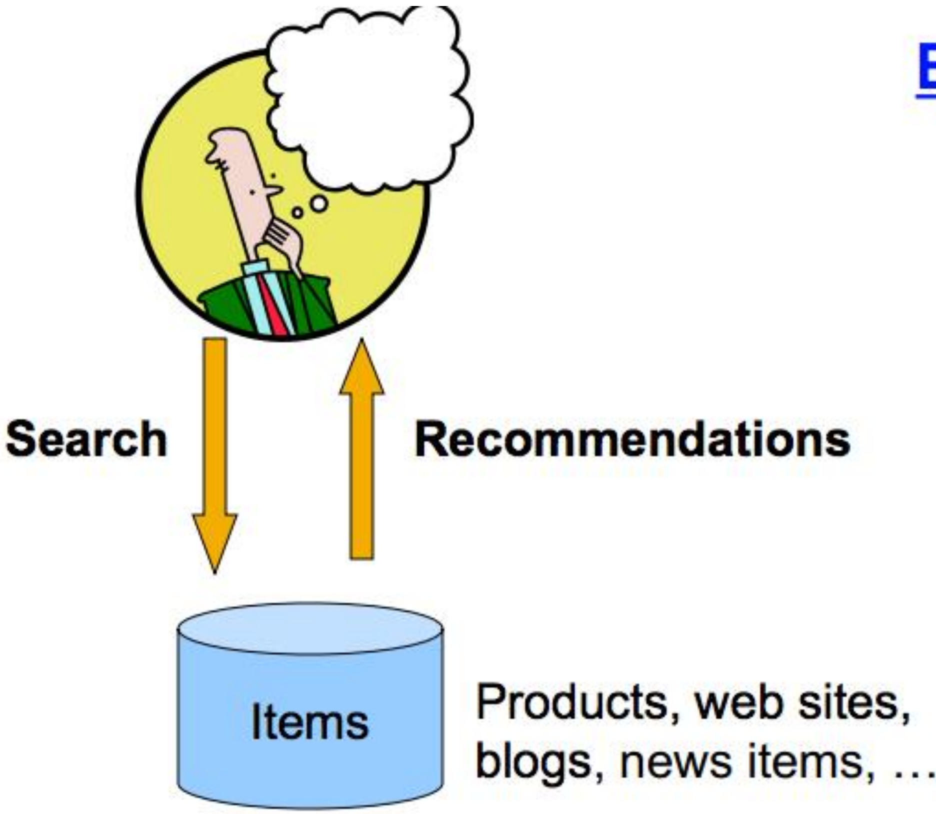
- Acknowledgment:
  - This presentation is based on the book “Mining of Massive Datasets” by Anand Rajaraman and Jeff Ullman and the presentations by Jure Leskovec

Recommender Systems



- **Customer X**
  - Buys Metallica CD
  - Buys Megadeth CD
- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Megadeth from data collected about

Recommender Systems



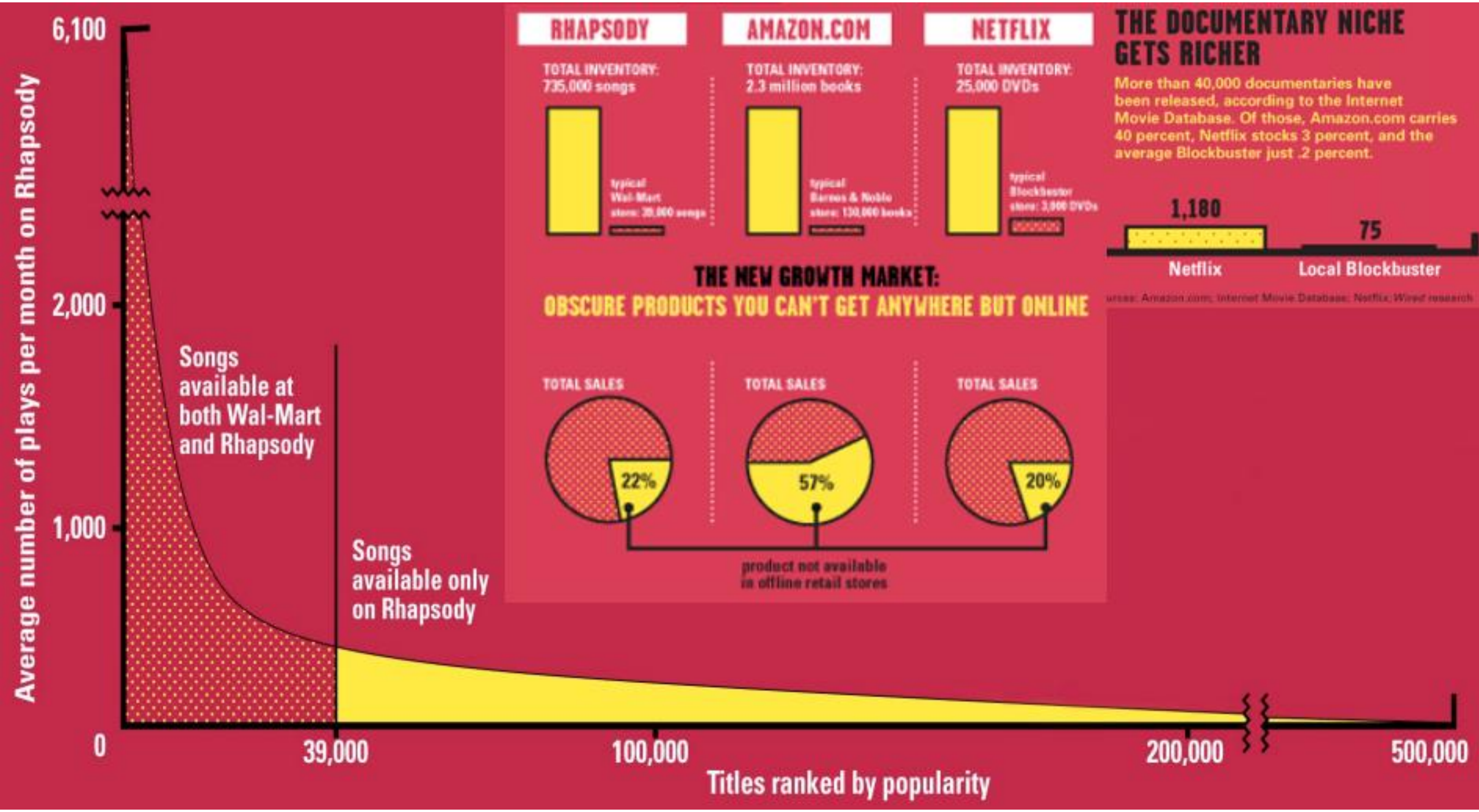
Examples:

amazon.com
PANDORA
StumbleUpon
del.icio.us
NETFLIX
movielens
helping you find the *right* movies
last.fm
the social music revolution
Google News
YouTube
XBOX LIVE

Abundance of Virtual Shelf Space

- **Shelf space is a scarce commodity for traditional retailers**
  - Also: TV networks, movie theaters,…
- **Web enables near-zero-cost dissemination of information about products**
  - From scarcity to abundance
- **More choice necessitates better filters**
  - Recommendation engines
  - How **Into Thin Air** made **Touching the Void** a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

Long Tail


6,100
2,000
1,000
0
Average number of plays per month on Rhapsody
Songs available at both Wal-Mart and Rhapsody
Songs available only on Rhapsody
39,000
100,000
200,000
500,000
Titles ranked by popularity
RHAPSODY
TOTAL INVENTORY: 735,000 songs
typical Wal-Mart store: 39,000 songs
AMAZON.COM
TOTAL INVENTORY: 2.3 million books
typical Barnes & Noble store: 130,000 books
NETFLIX
TOTAL INVENTORY: 25,000 DVDs
typical Blockbuster store: 3,000 DVDs
THE NEW GROWTH MARKET:
OBSCURE PRODUCTS YOU CAN'T GET ANYWHERE BUT ONLINE
TOTAL SALES
22%
TOTAL SALES
57%
TOTAL SALES
20%
product not available in offline retail stores
THE DOCUMENTARY NICHE GETS RICHER
More than 40,000 documentaries have been released, according to the Internet Movie Database. Of those, Amazon.com carries 40 percent, Netflix stocks 3 percent, and the average Blockbuster just .2 percent.
1,180
75
Netflix
Local Blockbuster
urces: Amazon.com; Internet Movie Database; Netflix; Wired research

Type of Recommendations

- **Editorial and hand curated**
  - List of favorites
  - Lists of “essential” items
- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads
- **Tailored to individual users**
  - Amazon, Netflix, …

Model

- X = set of Customers
- S = set of Items

- Utility function $u: X \times S \rightarrow R$
  - R = set of ratings
  - R is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]**

Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Issues

- **(1) Gathering "known" ratings for matrix**
  - How to collect the data in the utility matrix
- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
    - We are not interested in knowing what you don't like but what you like
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

Gathering Ratings

- Explicit
  - Ask people to rate items
  - Doesn’t work well in practice – people can’t be bothered
- Implicit
  - Learn ratings from user actions
    - E.g., purchase implies high rating
  - What about low ratings?

Extrapolating

- **Key problem:** matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history
- **Three approaches to recommender systems:**
  - **1)** Content-based
  - **2)** Collaborative
  - **3)** Latent factor based

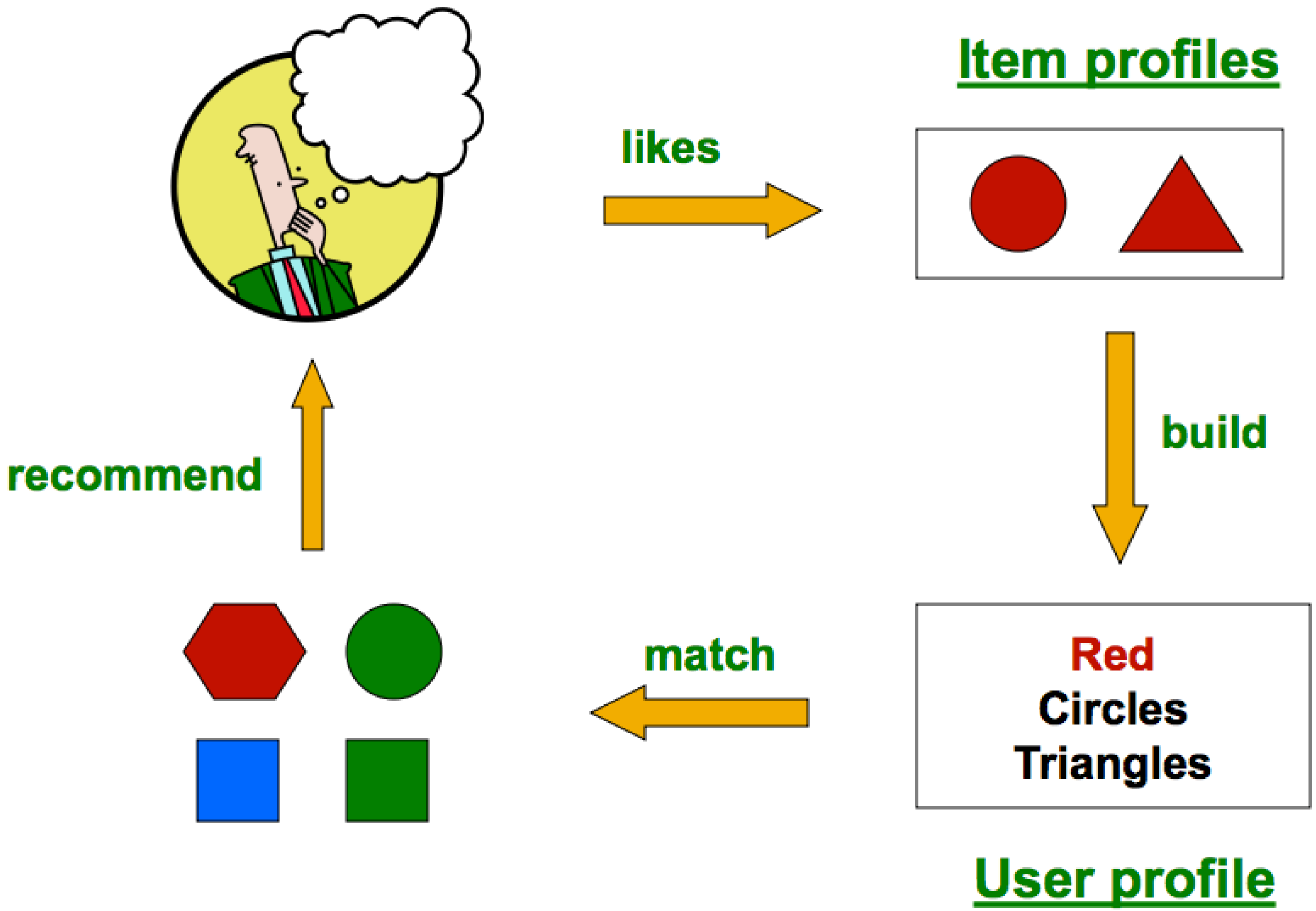
Content-Based Recommendation Systems

- **Main idea:** Recommend items to customer ***x*** similar to previous items rated highly by ***x***

Example:

- **Movie recommendations**
  - Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - Recommend other sites with "similar" content

Content-Based Recommendation Systems



Content-Based Approach "+"

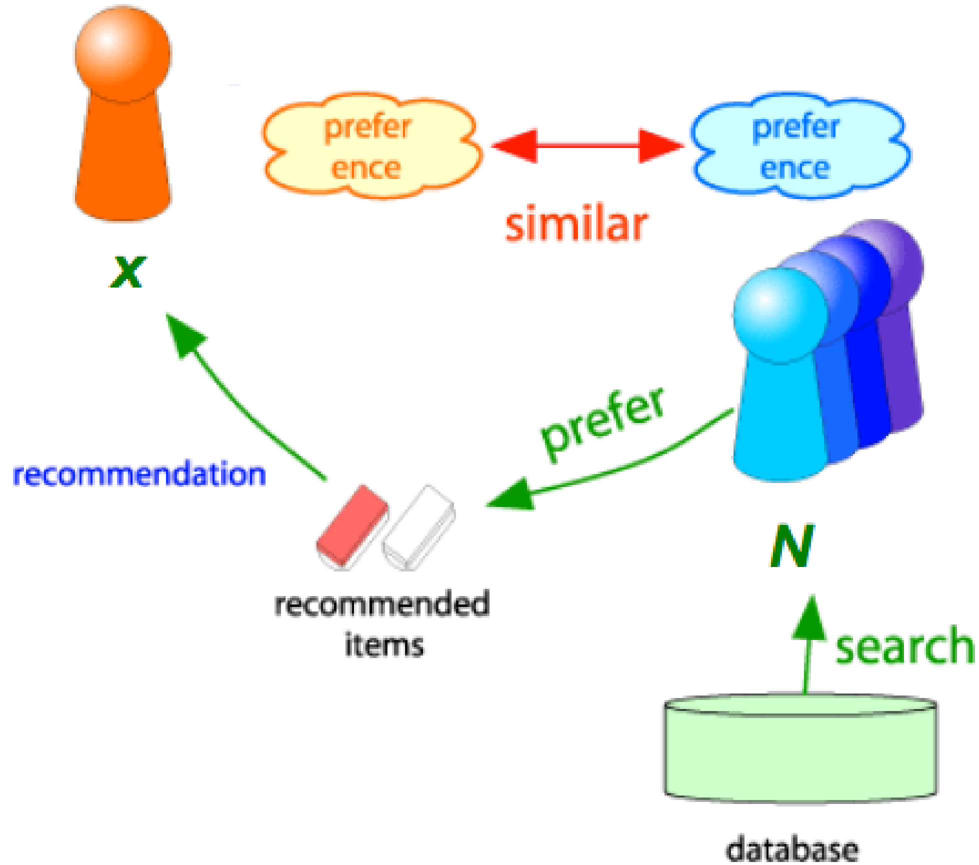
- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Content-Based Approach “-”

- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Overspecialization**
  - Never recommends items outside user’s content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users
- **–: Recommendations for new users**
  - **How to build a user profile?**

Collaborative Filtering

- Consider user ***x***
- Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings
- Estimate ***x***'s ratings based on ratings of users in ***N***



Item-Item Collaborative Filtering

- **So far: User-user collaborative filtering**
- **Another view: Item-item**
  - For item ***i***, find other similar items
  - Estimate rating for item ***i*** based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$\boxed{r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}}$$

s_{ij}… similarity of items ***i*** and ***j***
r_{xj}…rating of user ***u*** on item ***j***
N(i;x)… set items rated by ***x*** similar to ***i***

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating

- rating between 1 to 5

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie **1** by user **5**

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to
movie **1, rated by user 5**

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
<u>3</u>	2	4		1	2		3		4	3	5		<u>0.41</u>
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
<u>6</u>	1		3		3			2			4		<u>0.59</u>

Neighbor selection:
Identify movies similar to movie **1, rated by user 5**

Here we use Pearson correlation as similarity:

1) Subtract mean rating m_i from each movie i

$m_1 = (1+3+5+5+4)/5 = 3.6$

row 1: *[-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]*

2) Compute cosine similarities between rows

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights:

s_{13}=0.41, s_{16}=0.59

Item-Item Collaborative Filtering N=2

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		2.6	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average:

$r_{15} = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6$

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

Common Practice

- Define **similarity** s_{ij} of items i and j
- Select k nearest neighbors $N(i; x)$
  - Items most similar to i, that were rated by x
- Estimate rating r_{xi} as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$b_{xi} = \mu + b_x + b_i$$

- μ = overall mean movie rating
- b_x = rating deviation of user x
  = *(avg. rating of user* x*)* $- \mu$
- b_i = rating deviation of movie i

Item-Item vs User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- **In practice, it has been observed that <u>item-item</u> often works better than user-user**
- **Why?** Items are simpler, users have multiple tastes

Collaborative Filtering "+" and "-"

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Hybrid Methods

- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model
- **Add content-based methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

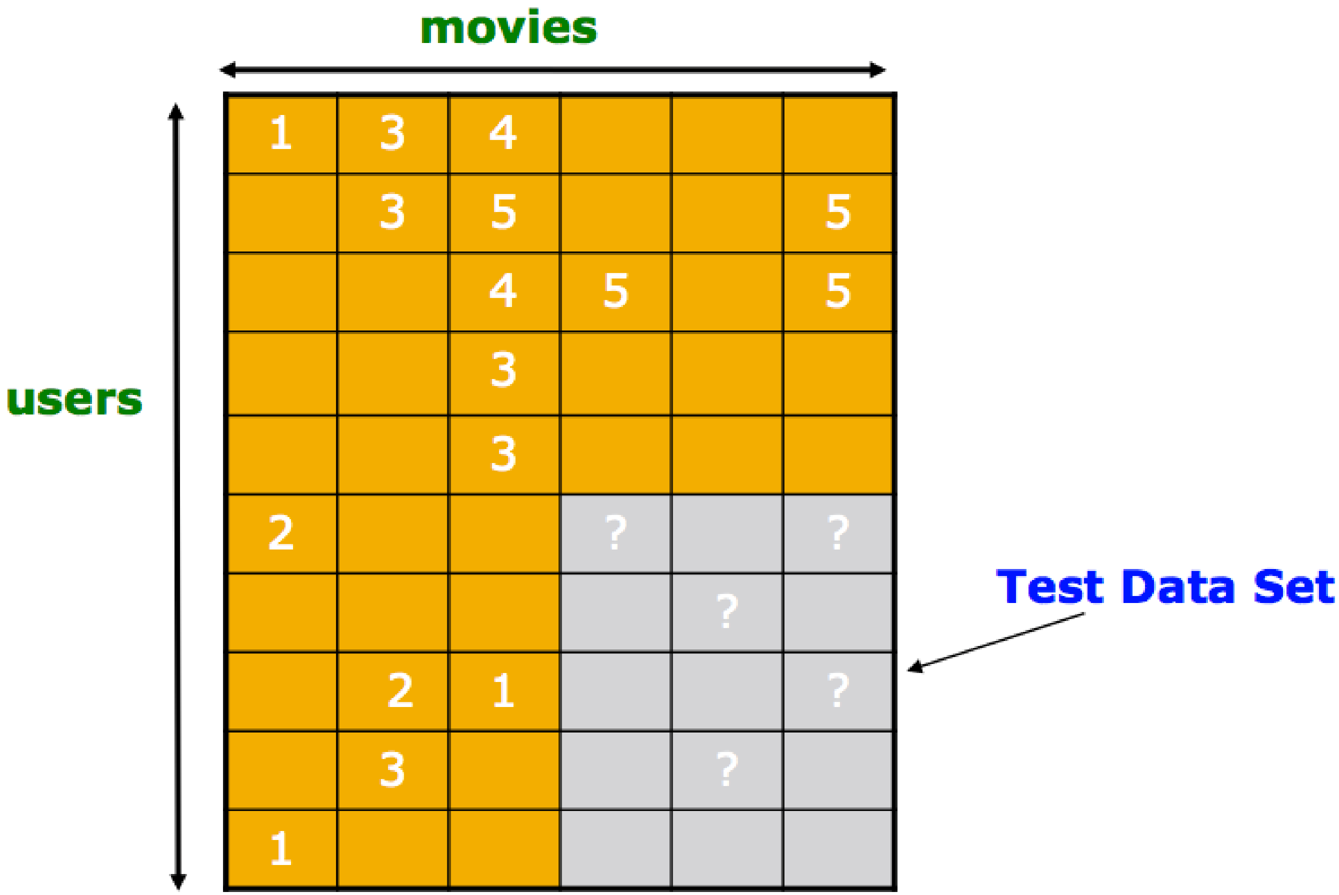
Evaluation

movies

users

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

Evaluation



Evaluation

- **Compare predictions with known ratings**
  - **Root-mean-square error** (RMSE)
    - $\sqrt{\sum_{xi}\left(r_{xi} - r_{xi}^{*}\right)^{2}}$ where $\boldsymbol{r}_{xi}$ is predicted, $\boldsymbol{r}_{xi}^{*}$ is the true rating of $\boldsymbol{x}$ on $\boldsymbol{i}$
  - **Precision at top 10**:
    - % of those in top 10
  - **Rank Correlation**:
    - Spearman's *correlation* between system's and user's complete rankings
- **Another approach: 0/1 model**
  - **Coverage:**
    - Number of items/users for which system can make predictions
  - **Precision:**
    - Accuracy of predictions
  - **Receiver operating characteristic** (ROC)
    - Tradeoff curve between false positives and false negatives

Problems with Error Measure

- **Narrow focus on accuracy sometimes misses the point**
  - Prediction Diversity
  - Prediction Context
  - Order of predictions
- **In practice, we care only to predict high ratings:**
  - RMSE might penalize a method that does well for high ratings and badly for others

Complexity of Collaborative Filtering

- Expensive step is finding ***k*** most similar customers: **O(|X|)**
- **Too expensive to do at runtime**
  - Could pre-compute
- Naïve pre-computation takes time **O(N ·|C|)**

Data

- **Leverage all the data**
  - Don’t try to reduce data size in an effort to make fancy algorithms work
  - Simple methods on large data do best
- **Add more data**
  - e.g., add IMDB data on genres
- **More data beats better algorithms**

- The Netflix Prize

The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion: Root Mean Square Error (**RMSE**)
  - **Netflix's system RMSE: 0.9514**
- Competition
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

The Netflix Utility Matrix R

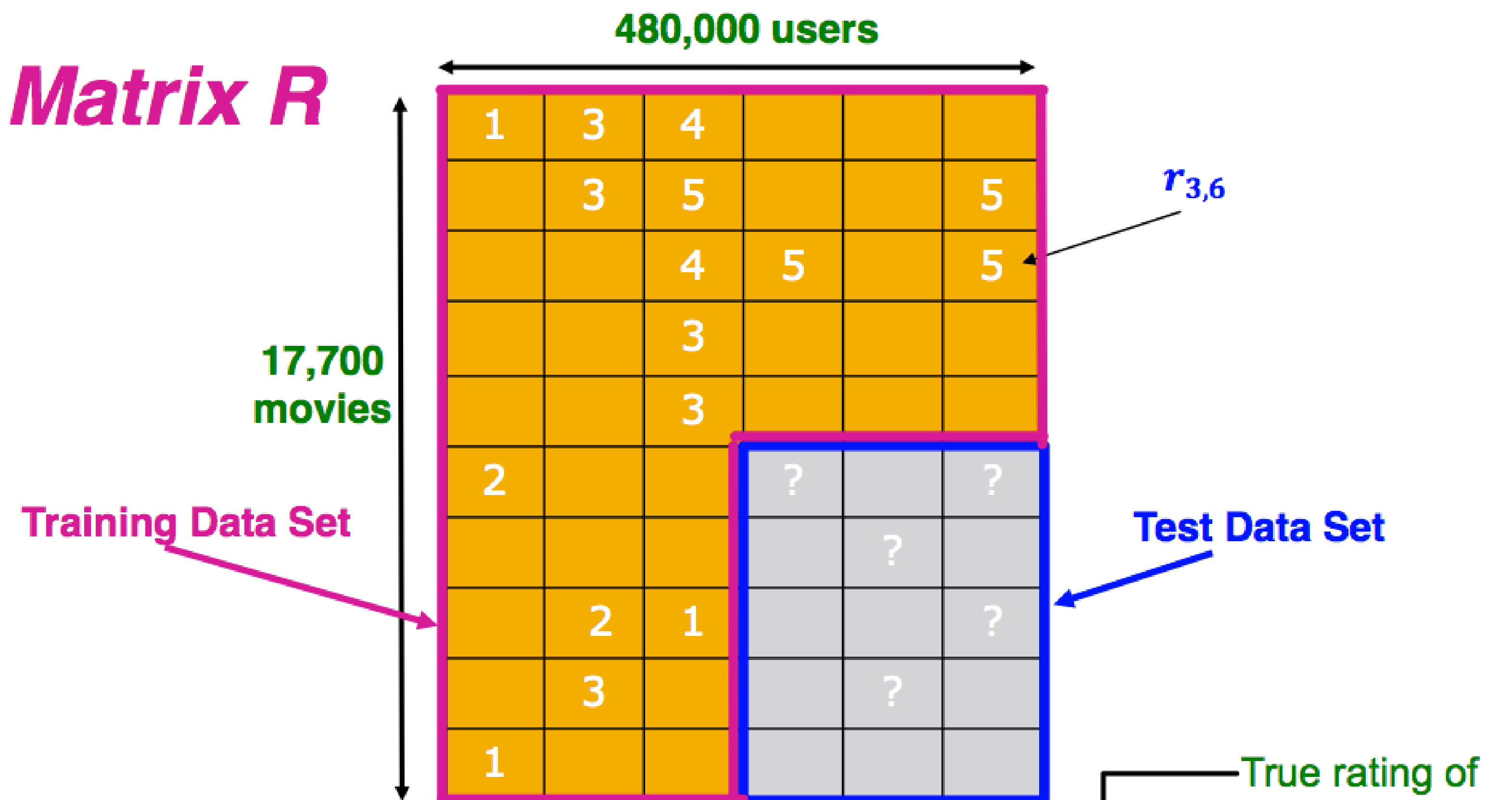
Matrix R

480,000 users

17,700 movies

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

Evaluation



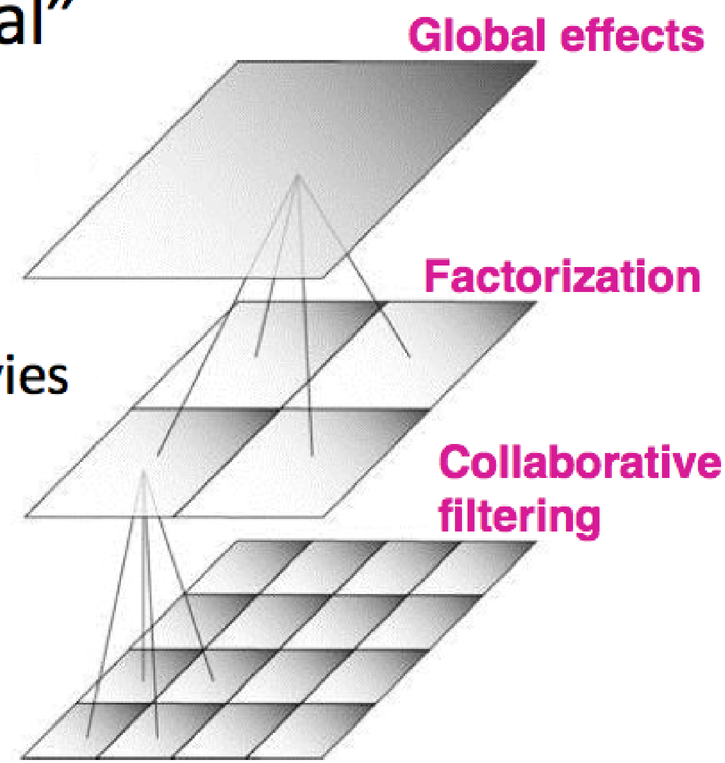
$$\mathbf{SSE} = \sum_{(i,x)\in R} (\hat{r}_{xi} - r_{xi})^2$$

True rating of user ***x*** on item ***i***

Predicted rating

Bellkor Recommender System

- **The winner of the Netflix Challenge**
- **Multi-scale modeling of the data:**
  Combine top level, “regional” modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing “regional” effects
  - **Collaborative filtering:**
    - Extract local patterns



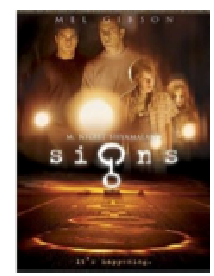
Modelling Global and Local Effects

- **Global:**
  - Mean movie rating: **3.7 stars**
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.

    ⇒ **Baseline estimation:**

    ***Joe* will rate *The Sixth Sense* 4 stars**
- **Local neighborhood (CF/NN):**
  - *Joe* didn't like related movie *Signs*
  - ⇒ **Final estimate:**

    ***Joe* will rate *The Sixth Sense* 3.8 stars**



Standard Collaborative Filtering

- **Earliest and most popular collaborative filtering method**
- Derive unknown ratings from those of "**similar**" movies (item-item variant)
- Define **similarity measure** s_{ij} of items i and j
- Select k-nearest neighbors, compute the rating
  - $N(i; x)$: items most similar to i that were rated by x

$$\hat{r}_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij}… similarity of items i and j
r_{uj}…rating of user x on item j
$N(i;x)$… set of items similar to item i that were rated by x

Modelling Global and Local Effects

- **In practice we get better estimates if we model deviations:**

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$\boldsymbol{b_{xi} = \mu + b_x + b_i}$$

$\boldsymbol{\mu}$ = overall mean rating

$\boldsymbol{b_x}$ = rating deviation of user $\boldsymbol{x}$

= *(avg. rating of* user $\boldsymbol{x}$*)* $- \boldsymbol{\mu}$

$\boldsymbol{b_i}$ = *(avg. rating of* movie $\boldsymbol{i}$*)* $- \boldsymbol{\mu}$

Problems/Issues:

1) Similarity measures are "arbitrary"

2) Pairwise similarities neglect interdependencies among users

3) Taking a weighted average can be restricting

Solution: **Instead of s_{ij} use w_{ij} that we estimate directly from data**

Interpolation Weights w_ij

- Use a **weighted sum** rather than **weighted avg.**:

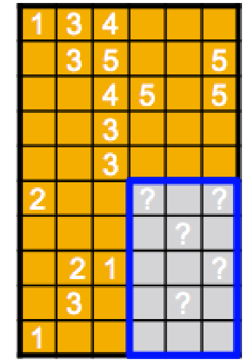
$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$$

- A few notes:
  - We sum over all movies ***j*** that are similar to ***i*** and were rated *by* ***x***
  - $\boldsymbol{w_{ij}}$ is the interpolation weight (some real number)
    - We allow: $\sum_{j \in N(i,x)} \boldsymbol{w_{ij}} \neq \mathbf{1}$
  - $\boldsymbol{w_{ij}}$ models interaction between pairs of movies (it does not depend on user ***x***)
  - $\boldsymbol{N(i;x)}$ … set of movies rated by user ***x*** that are similar to movie ***i***

Interpolation Weights w_ij

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij}(r_{xj} - b_{xj})$
- **How to set w_{ij}?**
  - Remember, error metric is **SSE:** $\sum_{(i,u)\in R}(\hat{r}_{ui} - r_{ui})^2$
  - Find w_{ij} that minimize **SSE** on **training data!**
    - Models relationships between item ***i*** and its neighbors ***j***
  - w_{ij} can be **learned/estimated** based on ***x*** and all other users that rated ***i***

Recommendation as Optimization



- **Here is what we just did:**
  - **Goal:** Make good recommendations
    - Quantify goodness using **SSE:**
      So, **Lower SSE means better recommendations**
    - We want to make good recommendations on items that some user has not yet seen. Can't really do this. Why?
    - **Let's set values *w* such that they work well on known (user, item) ratings**
      And **hope** these ***w***s will predict well the unknown ratings

Recommendation as Optimization

- **Idea: Let's set values *w* such that they work well on known (user, item) ratings**
- **How to find such values *w*?**
- **Idea:** Define an objective function and solve the optimization problem
- Find $\boldsymbol{w_{ij}}$ that minimize **SSE** on **training data**!

$$\min_{w_{ij}} \sum_x \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij} (r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

- Think of ***w*** as a vector of numbers

Interpolation Weights

- **We have the optimization problem, now what?**

$$\min_{w_{ij}} \sum_x \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij} (r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

- **Gradient decent**
  - **Iterate until convergence:** $\boldsymbol{w} \quad \boldsymbol{w} - \eta \boldsymbol{\nabla w}$ $\qquad \eta$ … learning rate
  - **where $\boldsymbol{\nabla w}$ is gradient (derivative evaluated on data):**

$$\nabla w = \left[\frac{\partial}{\partial w_{ij}} \right] = 2 \sum_x \left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik} (r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj})$$

for $\boldsymbol{j} \in \{\boldsymbol{N}(\boldsymbol{i}; \boldsymbol{x}), \forall \boldsymbol{i}, \forall \boldsymbol{x}\}$

else $\frac{\partial}{\partial w_{ij}} = \mathbf{0}$

  - **Note:** we fix movie $\boldsymbol{i}$, go over all $\boldsymbol{r_{xi}}$, for every movie $\boldsymbol{j} \in \boldsymbol{N}(\boldsymbol{i}; \boldsymbol{x})$, we compute $\frac{\partial}{\partial w_{ij}}$

while $|w_{new} - w_{old}| > \varepsilon$:

$\quad w_{old} = w_{new}$

$\quad w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

Gradient Descent

GD for Matrix Factorization

Based on Jeff Howbert Lectures

Collaborative filtering algorithms

- Common types:
  - Global effects
  - Nearest neighbor
  - Matrix factorization
  - Restricted Boltzmann machine
  - Clustering
  - Etc.

Optimization

- Optimization is an important part of many machine learning methods.

- The thing we're usually optimizing is the loss function for the model.
  - For a given set of training data **X** and outcomes **y**, we want to find the model parameters **w** that minimize the total loss over all **X**, **y**.

Loss function

- Suppose target outcomes come from set Y
  - Binary classification: $Y = \{0, 1\}$
  - Regression: $Y = \Re$ (real numbers)
- A loss function maps decisions to costs:
  - $L(y_i, \hat{y}_i)$ defines the penalty for predicting $\hat{y}_i$ when the true value is y_i.
- Standard choice for classification: 0/1 loss (same as misclassification error)

$$L_{0/1}(y_i, \hat{y}_i) = \begin{cases} 0 & \text{if } y_i = \hat{y}_i \\ 1 & \text{otherwise} \end{cases}$$

- Standard choice for regression: squared loss

$$L(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2$$

Least squares linear fit to data

- Calculate sum of squared loss (SSL) and determine **w**:

$$\mathrm{SSL} = \sum_{j=1}^{N} \left(y_j - \sum_{i=0}^{d} w_i \cdot x_i\right)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}} \cdot (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$\mathbf{y}$ = vector of all training responses y_j

$\mathbf{X}$ = matrix of all training samples $\mathbf{x}_j$

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

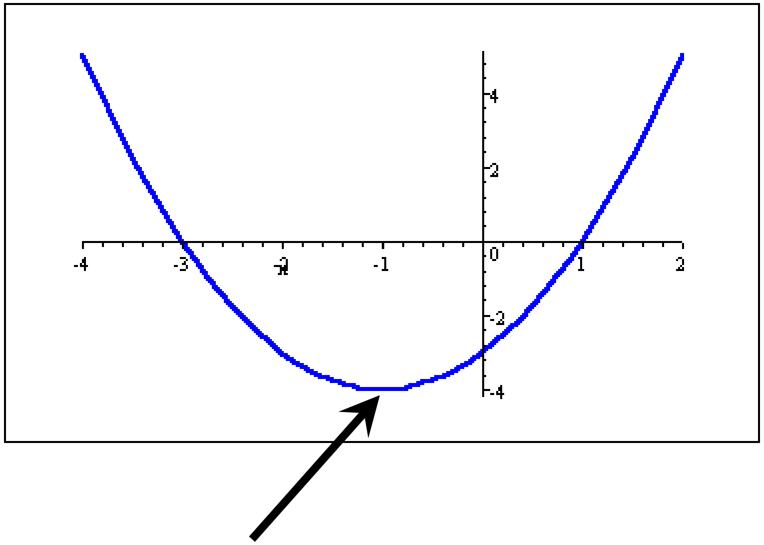
$$\hat{y}_t = \mathbf{w} \cdot \mathbf{x}_t \qquad \text{for test sample } \mathbf{x}_t$$

- Can prove that this method of determining **w** minimizes SSL.

- Simplest example - quadratic function in 1 variable:

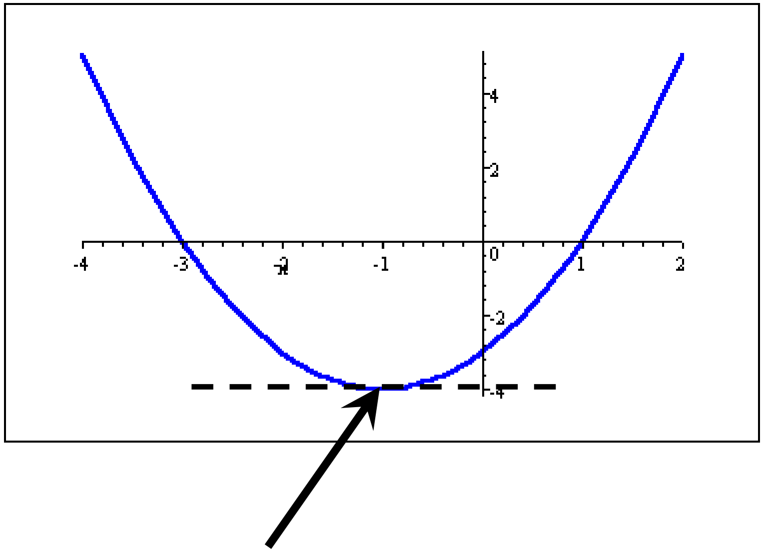
$$f(x) = x^2 + 2x - 3$$

- Want to find value of x where $f(x)$ is minimum



Optimization

- This example is simple enough we can find minimum directly
  - Minimum occurs where slope of curve is 0
  - First derivative of function = slope of curve
  - So set first derivative to 0, solve for *x*



Optimization

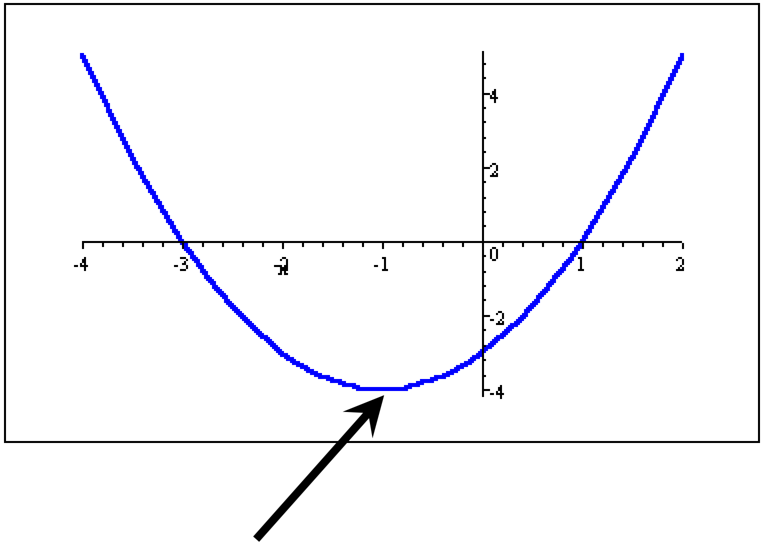
$$f(x) = x^2 + 2x - 3$$

$$f(x)\,/\,dx = 2x + 2$$

$$2x + 2 = 0$$

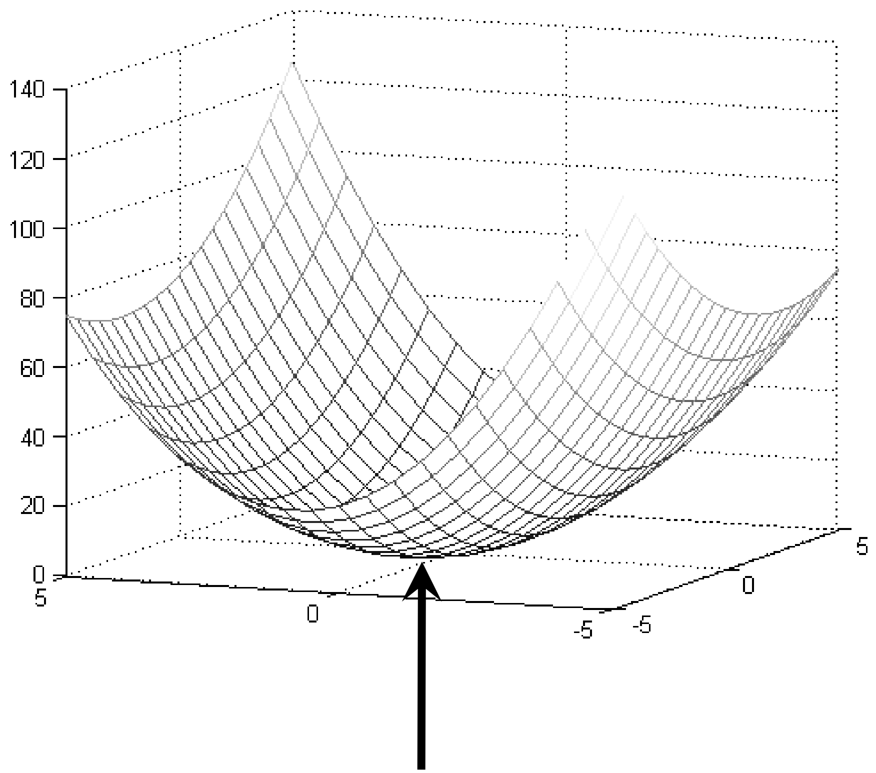
$$x = -1$$

is value of x where $f(x)$ is minimum



- Another example - quadratic function in 2 variables:

$$f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + x_1 x_2 + 3x_2^2$$



- $f(\mathbf{x})$ is minimum where gradient of $f(\mathbf{x})$ is zero in all directions

- Gradient is a vector
  - Each element is the slope of function along direction of one of variables
  - Each element is the partial derivative of function with respect to one of variables

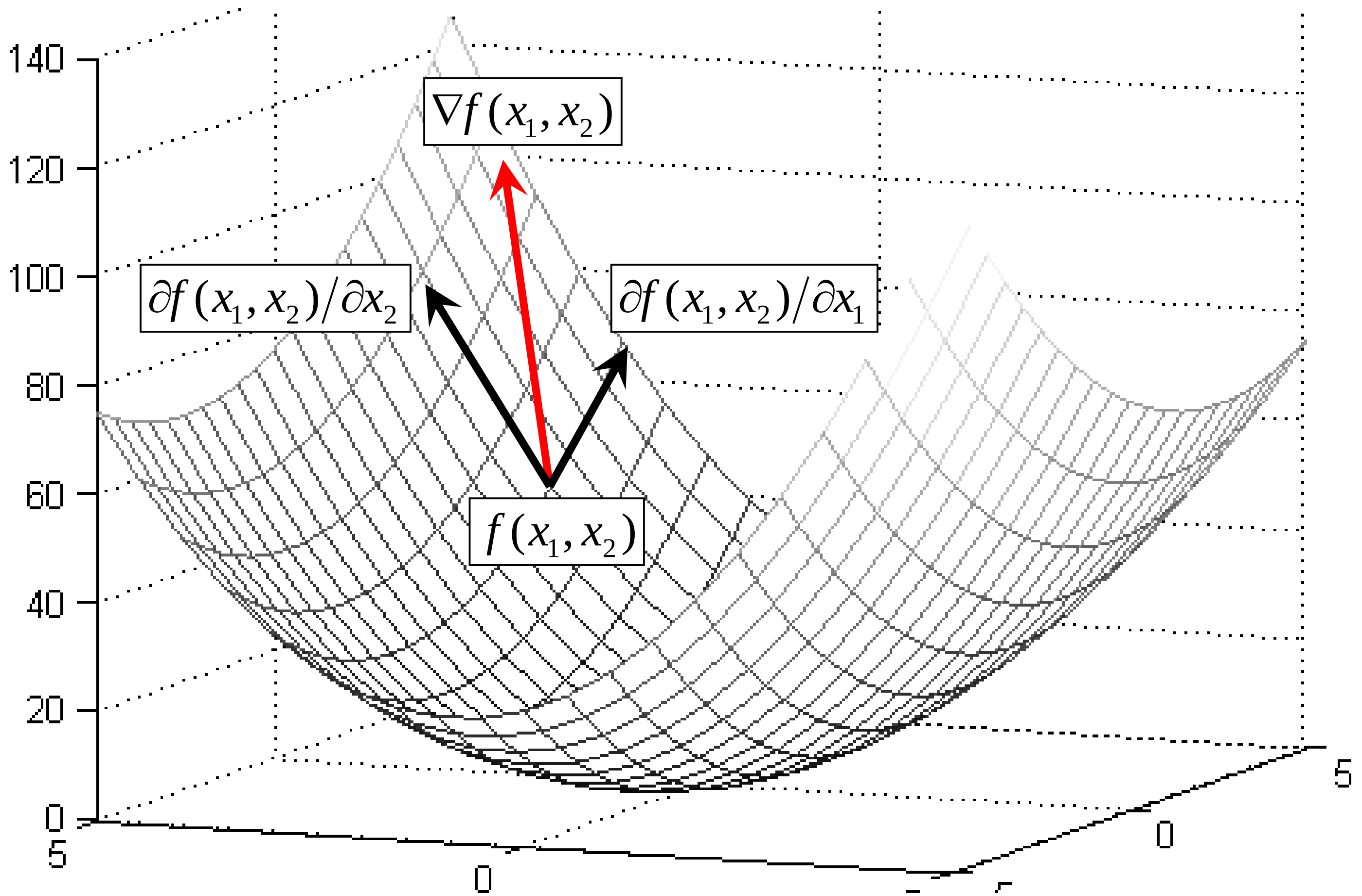
$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2, \ldots, x_d) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

  - Example:

$$f(\mathbf{x}) = f(x_1, x_2) = x_1^2 + x_1 x_2 + 3x_2^2$$

$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 & x_1 + 6x_2 \end{bmatrix}$$

- Gradient vector points in direction of steepest ascent of function



- This two-variable example is still simple enough that we can find minimum directly

$$f(x_1, x_2) = x_1^{\ 2} + x_1 x_2 + 3x_2^{\ 2}$$
$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 + x_2 & x_1 + 6x_2 \end{bmatrix}$$

  - Set both elements of gradient to 0
  - Gives two linear equations in two variables
  - Solve for x_1, x_2

$$2x_1 + x_2 = 0 \qquad\qquad x_1 + 6x_2 = 0$$
$$x_1 = 0 \qquad\qquad x_2 = 0$$

Optimization

- Finding minimum directly by closed form analytical solution often difficult or impossible.
  - Quadratic functions in many variables
    - system of equations for partial derivatives may be ill-conditioned
    - example: linear least squares fit where redundancy among features is high
  - Other convex functions
    - global minimum exists, but there is no closed form solution
    - example: maximum likelihood solution for logistic regression
  - Nonlinear functions
    - partial derivatives are not linear
    - example: $f(x_1, x_2) = x_1(\sin(x_1 x_2)) + x_2^2$
    - example: sum of transfer functions in neural networks

Optimization

- Many approximate methods for finding minima have been developed
  - Gradient descent
  - Newton method
  - Gauss-Newton
  - Levenberg-Marquardt
  - BFGS
  - Conjugate gradient
  - Etc.

Gradient descent optimization

- Simple concept: follow the gradient *downhill*
- Process:
  1. Pick a starting position: $\mathbf{x}^0 = (x_1, x_2, \ldots, x_d)$
  2. Determine the descent direction: $- \nabla f(\mathbf{x}^t)$
  3. Choose a learning rate: η
  4. Update your position: $\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \cdot \nabla f(\mathbf{x}^t)$
  5. Repeat from 2) until stopping criterion is satisfied
- Typical stopping criteria
  - $\nabla f(\mathbf{x}^{t+1}) \sim 0$
  - some validation metric is optimized

Gradient descent optimization

Slides thanks to Alexandre Bayen

(CE 191, Univ. California, Berkeley, 2006)

http://www.ce.berkeley.edu/~bayen/ce191www/lecturenotes/lecture10v01_descent2.pdf

Gradient descent optimization

Example in MATLAB

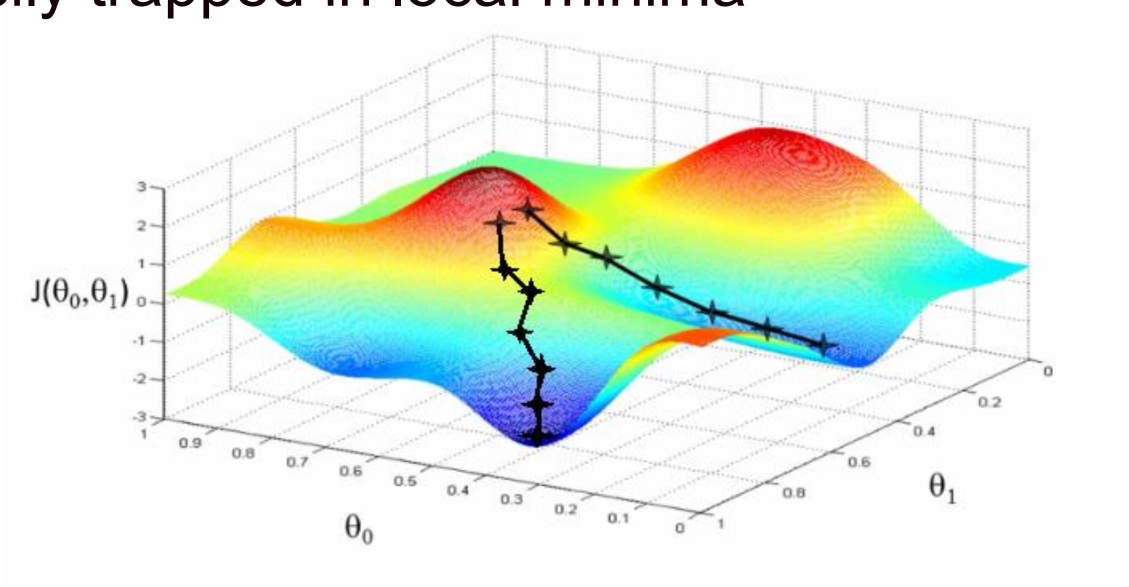
Find minimum of function in two variables:

$$y = x_1^2 + x_1 x_2 + 3x_2^2$$

http://www.youtube.com/watch?v=cY1YGQQbrpQ

Gradient descent optimization

- Problems:
  - Choosing step size
    - too small $\rightarrow$ convergence is slow and inefficient
    - too large $\rightarrow$ may not converge
  - Can get stuck on “flat” areas of function
  - Easily trapped in local minima



Stochastic gradient descent

Stochastic (definition):

1. involving a random variable
2. involving chance or probability; probabilistic

Stochastic gradient descent

- Application to training a machine learning model:
  1. Choose one sample from training set
  2. Calculate loss function for that single sample
  3. Calculate gradient from loss function
  4. Update model parameters a single step based on gradient and learning rate
  5. Repeat from 1) until stopping criterion is satisfied
- Typically entire training set is processed multiple times before stopping.
- Order in which samples are processed can be fixed or random.

Matrix factorization in action

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
user 1			1		2							3
user 2		2		3	3			4				
user 3							5	3		4		
user 4	2				3			2				2
user 5		4				5			3			4
user 6			2									
user 7			2					4	2	3		
user 8	3	4				4						
user 9									3			
user 10			1		2							2
…												
user 480189		4			3			3				

training data

→ factorization (training process)

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
feature 1												
feature 2												
feature 3												
feature 4												
feature 5												

< a bunch of numbers >

+

	feature 1	feature 2	feature 3	feature 4	feature 5
user 1					
user 2					
user 3					
user 4					
user 5					
user 6					
user 7					
user 8					
user 9					
user 10					
…					
user 480189					

< a bunch of numbers >

Matrix factorization in action

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
feature 1												
feature 2												
feature 3												
feature 4												
feature 5												

+

	feature 1	feature 2	feature 3	feature 4	feature 5
user 1					
user 2					
user 3					
user 4					
user 5					
user 6					
user 7					
user 8					
user 9					
user 10					
…					
user 480189					

multiply and add features (dot product) for desired < *user*, *movie* > prediction

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10	…	movie 17770
user 1			1		2							3
user 2		2		3	3			4				
user 3							5	3		4		
user 4	2				3			2				2
user 5		4				5			3			4
user 6			2									
user 7			2					4	2	3		
user 8	3	4				4	?					
user 9									3			
user 10			1		2							2
…												
user 480189		4			3			3				