Recommendation Systems

Cam Tu Nguyen

阮锦绣

Software Institute, Nanjing University
nguyenct@lamda.nju.edu.cn
ncamtu@gmail.com

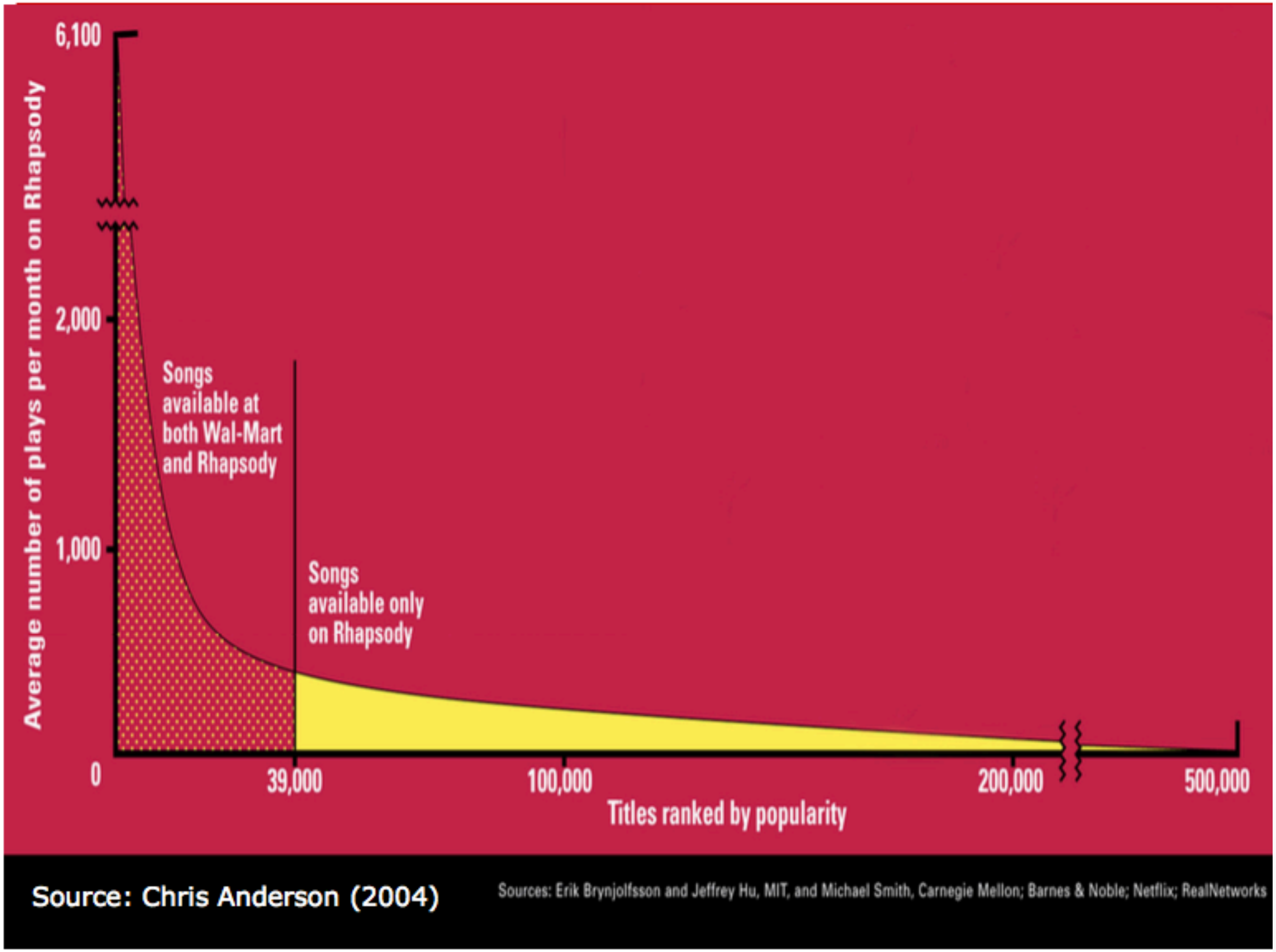
Outline

- Recommendation Systems
- Main Approaches
  - Content-Based Recommendations
  - Collaborative Filtering
- Approximate Nearest Neighbor Search
  - Locality Sensitive Hashing

Recommendations

- **Product Recommendations**
  - Online retailers such as Amazon, Alibaba
  - Return users products that they might like to buy
- **Movie Recommendations**
  - Netflix offers its customers recommendations of movies they might like.
  - The recommendations are based on ratings provided by users
- **News Articles**
  - News services have attempted to identify articles of interest to readers ,based on the articles that they have read in the past.
- Other applications: blogs recommendations, video recommendations on Youtube, etc.

The Long Tail



Recommendation Types

- Editorial
- Simple aggregates
  - Top 10, Most Popular, Recent Uploads
- Tailored to individual users
  - Amazon, Netflix, …

Formal Model

- C = set of Customers
- S = set of Items

- Utility function u: C x S → R
  - R = set of ratings
  - R is a totally ordered set
  - E.g., 0-5 starts, 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

Populating the Utility Matrix

- 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 ratings
  - What is about

Two basic architectures for a recommendation system

- Content-based systems focus on properties of items
  - Similarity of items is determined by measuring the similarity in their properties.
- Collaborative-Filtering systems focus on the relationship between users and items.
  - Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

Outline

- Recommendation Systems
- Main Approaches
  - Content-Based Recommendations
  - Collaborative Filtering
- Approximate Nearest Neighbor Search
  - Locality Sensitive Hashing

Item Profiles

- For each item, create an item profile

- Profile is a set of features/attributes
  - Movies: author, title, actor, director, …
  - Text: set of “important” words in document
  - Music Product: artist, composer, and genre.

TF.IDF

- How to pick important words?
  - Use heuristic is TF.IDF (Term Frequency times Inverse Doc Frequency)
- TF.IDF
  - f_{ij} = frequency of term t_i in document d_j

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

  - n_i = number of docs that mention term I
  - N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

  - TF.IDF score $w_{ij} = Tf_{ij} \times IDF_i$
- Doc profile = set of words with highest TF.IDF scores, together with their scores.

Representing Item Profiles

- Example: Suppose the only features of movies are the set of actors and the average rating. Consider two movies with five actors each.
  - Two of the actors are in both movies
  - One movie has an average rating of 3, and the other an average of 4.

$$\begin{array}{ccccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 3\alpha \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 4\alpha \end{array}$$

  - **Alpha** is introduced as a scaling factor for the average rating feature.

  - Similarity between 2 items can be measured using **cosine**

User profiles and predictions

- User profiles describe users' preferences
- User profile possibilities
  - Average of rated item profiles.
  - Variation: normalize the utilities by subtracting the average value for a user. That way, we get negative weights for items with a below-average rating, and positive weights for items with above-average rating.
  - …

Recommending Items to Users Based on Content

- Given user profile **c** and item profile **s**
  - Estimate u(**c,s**) = cos(**c,s**)=**c.s**/(|**c**||**s**|)
  - Need efficient method to find items with high utility
    - Locality Sensitive Hashing (stay tune!)

Limitations of content-based approach

- Find the appropriate features
  - E.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

Collaborative Filtering

- Consider user **c**
- Find set **D** of other users whose ratings are “similar” to c’s ratings
- Estimate user’s ratings based on ratings of users in D

Similar Users

- Let r_x be the vector of user x's ratings
- Cosine similarity measure
  - $Sim(x,y) = \cos(r_x, r_y)$
- Pearson correlation coefficient
  - S_{xy} = items rated by both users x and y

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r_x})(r_{ys} - \bar{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r_x})^2 (r_{ys} - \bar{r_y})^2}}$$

- Other similarity measures: Jaccard Distance, etc.

Similar Users

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- **Cosine Similarity**
  - We can treat blanks as a 0 value (it might be not the best choice)
  - The cosine of the angle between A and B is

$$\frac{4 \times 5}{\sqrt{4^2 + 5^2 + 1^2}\sqrt{5^2 + 5^2 + 4^2}} = 0.380$$

  - The cosine of the angle between A and C is

$$\frac{5 \times 2 + 1 \times 4}{\sqrt{4^2 + 5^2 + 1^2}\sqrt{2^2 + 4^2 + 5^2}} = 0.322$$

Rating predictions

- Let **D** be the set of **k** users most similar to **c** who have rated item **s**
- Possibilities for prediction function (item s):

$$r_{cs} = 1/k \; \Sigma_{d \text{ in } D} \; r_{ds}$$

$$r_{cs} = (\Sigma_{d \text{ in } D} \; sim(c,d) \; r_{ds})/(\Sigma_{d \text{ in } D} \; sim(c,d))$$

Complexity

- Expensive step is finding k most similar customers
  - For each user, O(|U|)
  - How to make it faster? (Again, Locality Sensitive Hashing comes to rescue!)
- Too expensive to do at runtime
  - Could pre-compute (e.g. using MapReduce in offline mode)
- Can use clustering, partitioning as alternatives, but quality degrades.

Item-Item Collaborative Filtering

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

- In practice, it has been observed that item-item often works better than user-user.

Pros and cons of Collaborative Filtering

- Works for any kind of item
  - No feature selection needed
- New user problem
- New item problem
- Sparsity of rating matrix
  - Cluster-based smoothing
  - Add more data.

Evaluating Predictions

- Compare predictions with known ratings
  - Root-mean-square error (RMSE)

- 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.

Finding similar vectors

- Common problem that comes up in many settings
- Given a large number N of vectors in some high-dimensional space (M dimensions), find pairs of vectors that have high similarity
  - E.g. User profiles, item profiles

- We need a method to solve this problem fast! (Locality Sensitive Hashing)

Outline

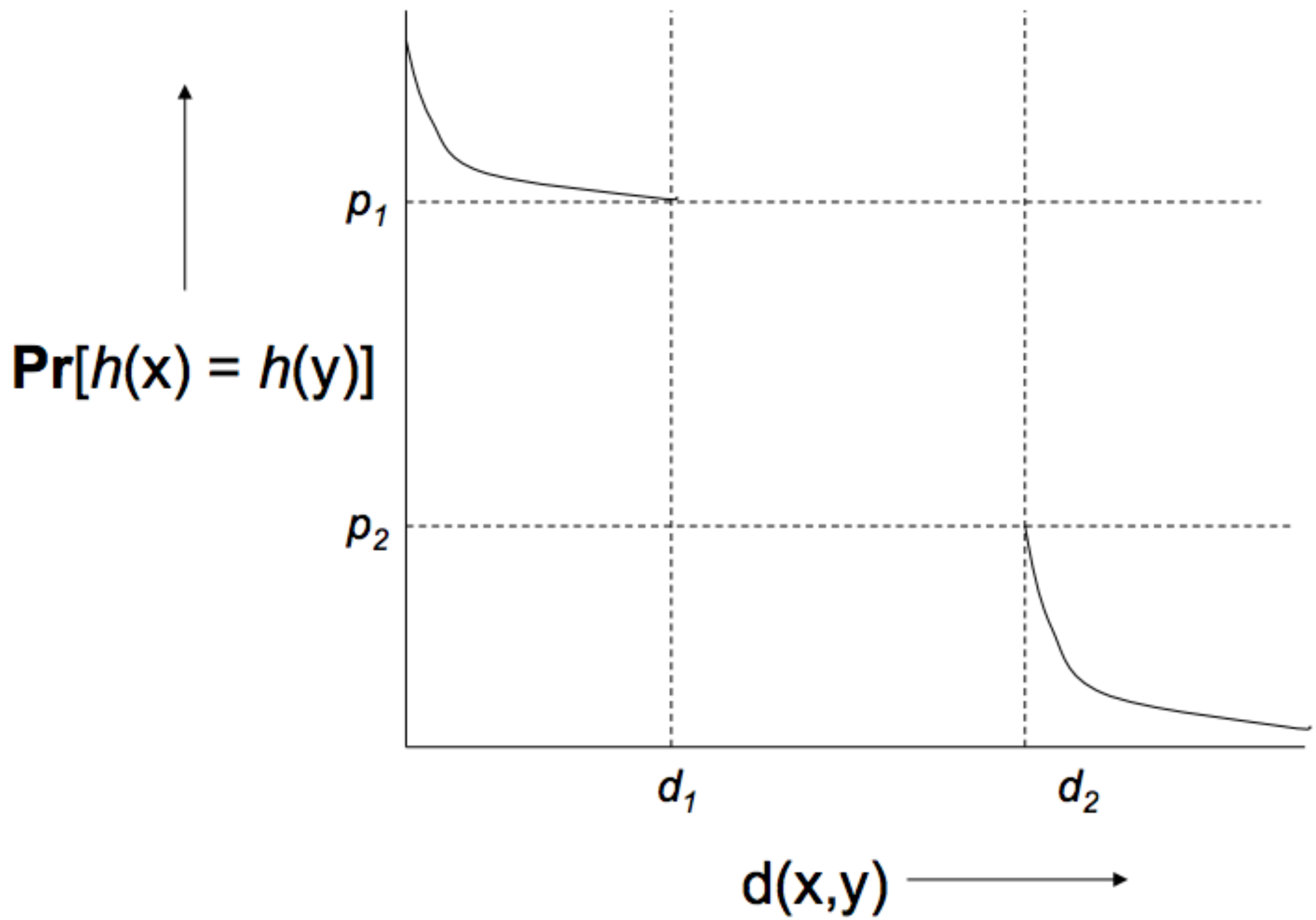
- Recommendation Systems
- Main Approaches
  - Content-Based Recommendations
  - Collaborative Filtering
- Near Neighbor Search in High Dimensional Data
  - Locality Sensitive Hashing

Locality Sensitive Functions

- *Locality-sensitive (LS) family is a family of functions that can be combined to distinguish strongly between pairs at a low distance from pairs at a high distance.*

- Three conditions for a LS family function
  - They must be more likely to make close pairs be candidate pairs than distant pairs
  - They must be statistically independent
  - They must be efficient, in two ways
    - Serve to identify candidates pairs in time much less than the time it takes to look at all pairs.
    - They must be combinable to build functions that are better at avoiding false positives and negatives.

A (d_1, d_2, p_1, p_2)-sensitive function



Amplifying a LS-family

- Two constructions:
  - AND construction
  - OR construction
- **AND of Hash functions**
  - Given family H, construct family H' consisting of r functions from H
  - For $h=[h_1,\ldots,h_r]$ in H', $h(x)=h(y)$ if and only if $h_i(x)=h_i(y)$ for all i.
  - Theorem: If H is (d_1, d_2, p_1, p_2)-sensitive, then H' is $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensitive.
- **OR of Hash functions**
  - Given family H, construct family H' consisting of b functions from H
  - For h=[h1, …, hb] in H', $h(x)=h(y)$ if and only if $h_i(x)=h_i(y)$ for **some** i.
  - Theorem: If H is (d_1, d_2, p_1, p_2)-sensitive, then H' is $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive.

AND-OR Composition

- Apply a r-way AND construction followed by an b-way OR construction.
- Transforms probability p into $1-(1-p^r)^b$.
- Example: Take H and construct H' by the AND construction with r=4. Them from H', construct H'' by the OR construction with b=4.

AND-OR Composition

- Example: Take H and construct H' by the AND construction with r=4. Them from H', construct H'' by the OR construction with b=4.

p	$1-(1-p^4)^4$
.2	.0064
.3	.0320
.4	.0985
.5	.2275
.6	.4260
.7	.6666
.8	.8785
.9	.9860

Example: Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.8785,.0064)-sensitive family.

OR-AND Composition

- Apply a b-way OR construction followed by an r-way AND construction.
- Transforms probability p into $(1-(1-p)^b)^r$.
- Example: Take H and construct H' by the OR construction with b=4. Then from H', construct H'' by the AND construction with r =4.

OR-AND Composition

- Example: Take H and construct H’ by the OR construction with b=4. Then from H’, construct H’’ by the AND construction with r =4.

p	$(1-(1-p)^4)^4$
.1	.0140
.2	.1215
.3	.3334
.4	.5740
.5	.7725
.6	.9015
.7	.9680
.8	.9936

Example: Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.9936,.1215)-sensitive family.

Summary of LS families

- Pick any two distances $x < y$
- Start with a $(x,y, (1-x), (1-y))$-sensitive family
- Apply constructions to produce (x,y, p, q)-sensitive family, where p is almost 1 and q is almost 0.
- The closer to 0 and 1 we get, the more hash functions must be used.

LSH for Cosine Distance

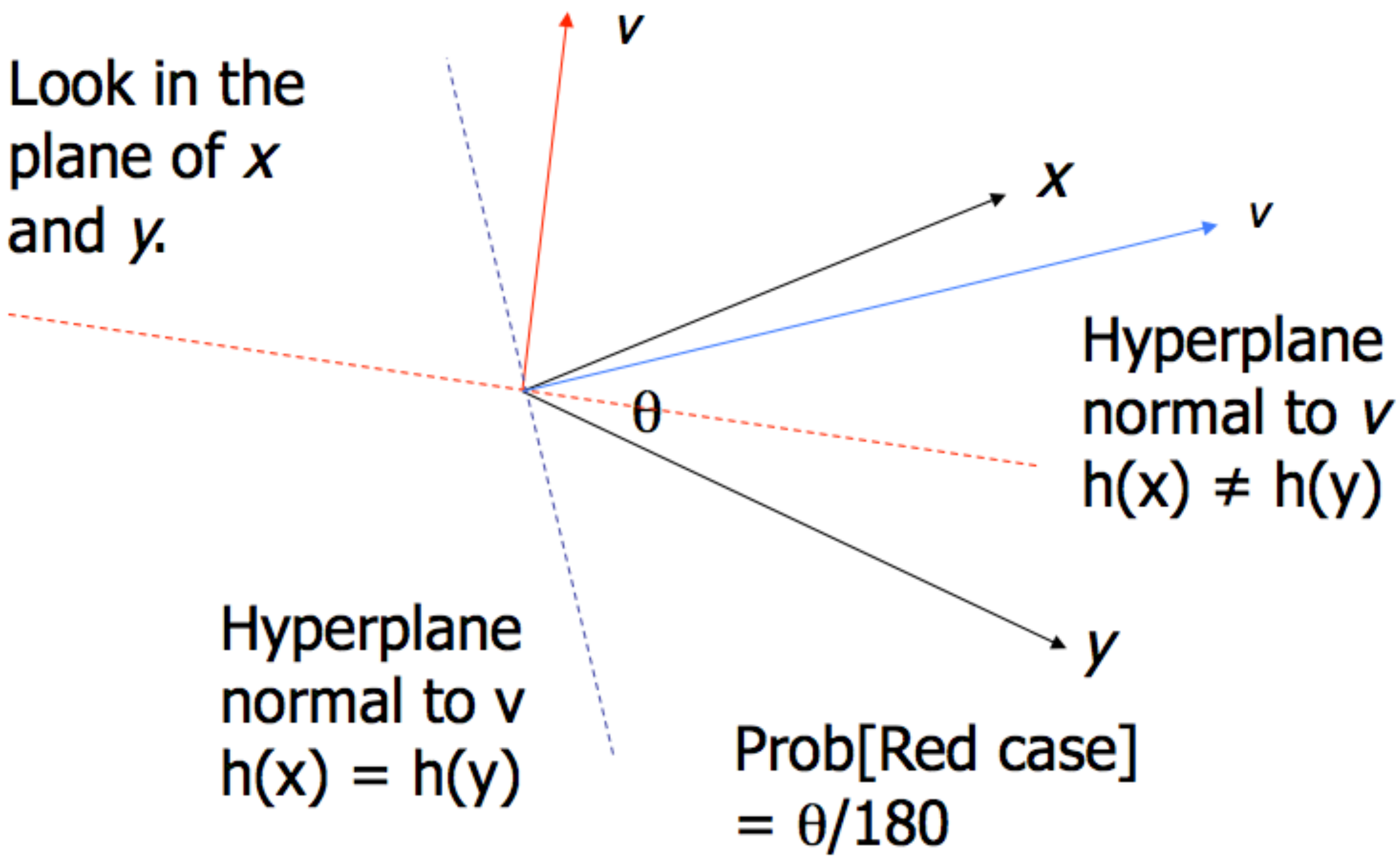
- Random Hyper planes
  - Convert data matrix to signature matrix (each signature corresponds to one object).
  - A $(d_1,d_2, (1-d_1/180), (1-d_2/180))$-sensitive family for any d_1, d_2.
- Apply hashing to the signature matrix
  - Objects in the same buckets are candidate pairs.

Random Hyperplanes

- Pick a random vector v, which determines a hash function hv with two buckets
  - $h_v(x) = +1$ if $v.x > 0$; -1 if $v.x < 0$

- LS-family H = set of all functions derived from any vector.
- Claim: For points x and y
  - $P[h(x)=h(y)] = 1 – d(x,y)/180$

Proof of Claim



Locality Sensitive Hashing for Cosine Distance

- Signatures for Cosine Distance
  - Pick some number of random vectors, and hash your data for each vector.
    - It suffices to consider only vectors consisting of +1 and -1 components
  - The result is a signature (**sketch)** of +1 and -1's for each data point.

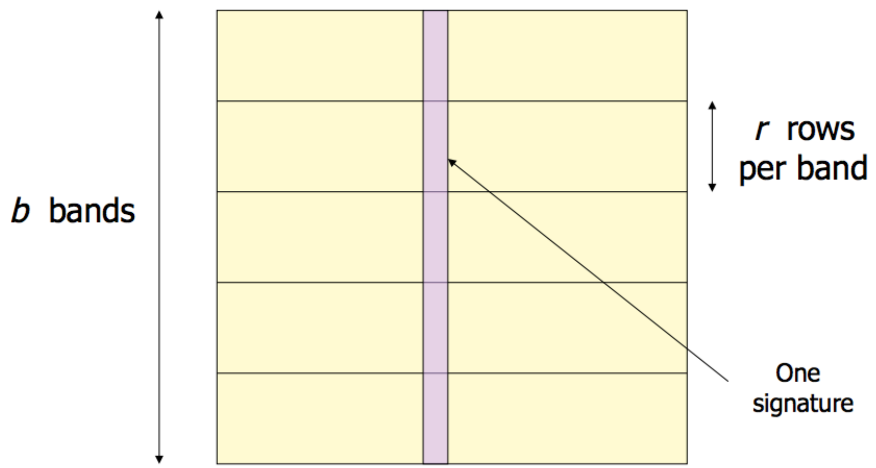
Locality Sensitive Hashing for Cosine Distance

- **Example**: Suppose our space is 4-dimensional space
  - Consider two vectors x=[3,4,5,6] and y=[4,3,2,1]
  - The cosine of the angle between x and y is 0.7875, or the angle between x and y is about 38 degrees.
  - We pick 3 random vectors: v1=[+1, -1, +1, +1]; v2=[-1, +1, -1, +1], and v3=[+1, +1, -1, -1].
    - For the vector x=[3,4,5,6], the sketch is [+1,+1, -1]
    - For the vector y=[4,3,2,1], the sketch is [+1, -1, +1]
    - The sketches for x and y agree in 1/3 of the positions, we estimate the angle between them is 120 degrees. (**not even close!)**
  - If we look at all 16 random vectors (why 16?)
    - There are only 4 of v vectors where v.x and v.y have different signs.
      - The vectors include v2, and v3
    - The estimate of the angle would have been 180/4=45 degrees. (**better**)

Locality Sensitive Hashing for Cosine Distance

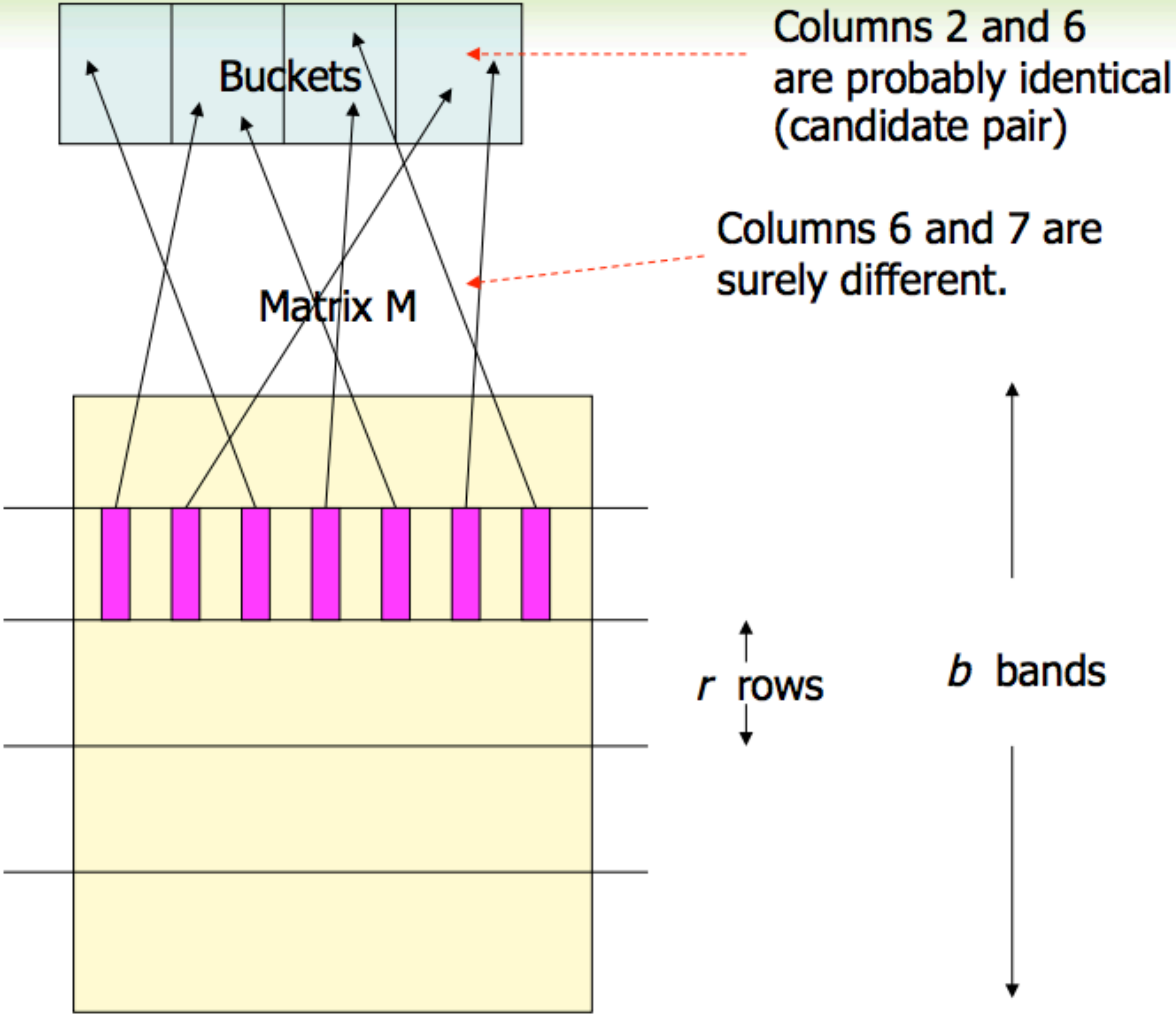
- Signatures for Cosine Distance
  - Pick some number of random vectors, and hash your data for each vector.
  - The result is a signature (**sketch)** of +1 and -1's for each data point.

- LSH: Partition into bands



Signature Matrix M

Signature Matrix:
#rows = # random vectors
#columns = # data objects


Buckets
Columns 2 and 6 are probably identical (candidate pair)
Columns 6 and 7 are surely different.
Matrix M
r rows
b bands

Partition into Bands (cont.)

- Divide signature matrix **M** into **b** bands of **r** rows.
  - Create one hash table per band.
- For each band, hash its portion of each column to its hash table
- ***Candidate pairs*** are columns that hash to the same bucket for >= band.
- Tune **b** and **r** to catch most similar pairs, but few non-similar pairs.

Other LSH

- LSH for Jaccard distance
- LSH for Euclidean distance
- LSH for Hamming distance

Summary

- Recommendation Systems
- Main Approaches
  - Content-Based Recommendations
  - Collaborative Filtering
- Approximate Nearest Neighbor Search
  - Locality Sensitive Hashing