Recommendation Systems

Content-Based Recommendations

Collaborative Filtering

Hybrid Systems

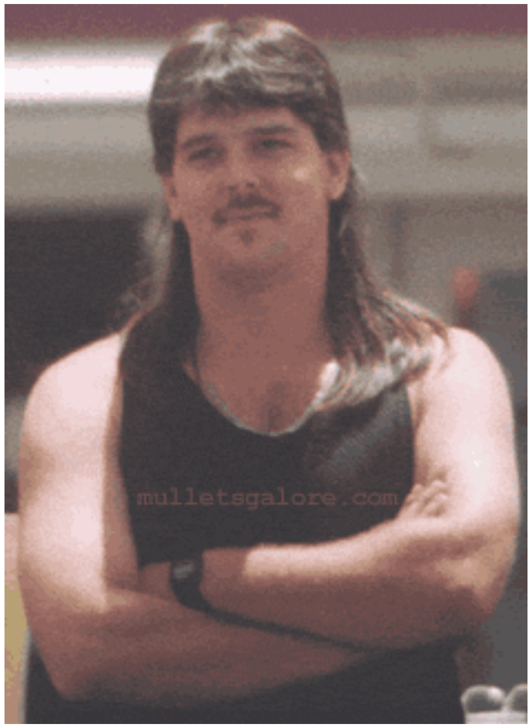
Thanks for source slides and material to: J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets

http://www.mmds.org

Additional Reading

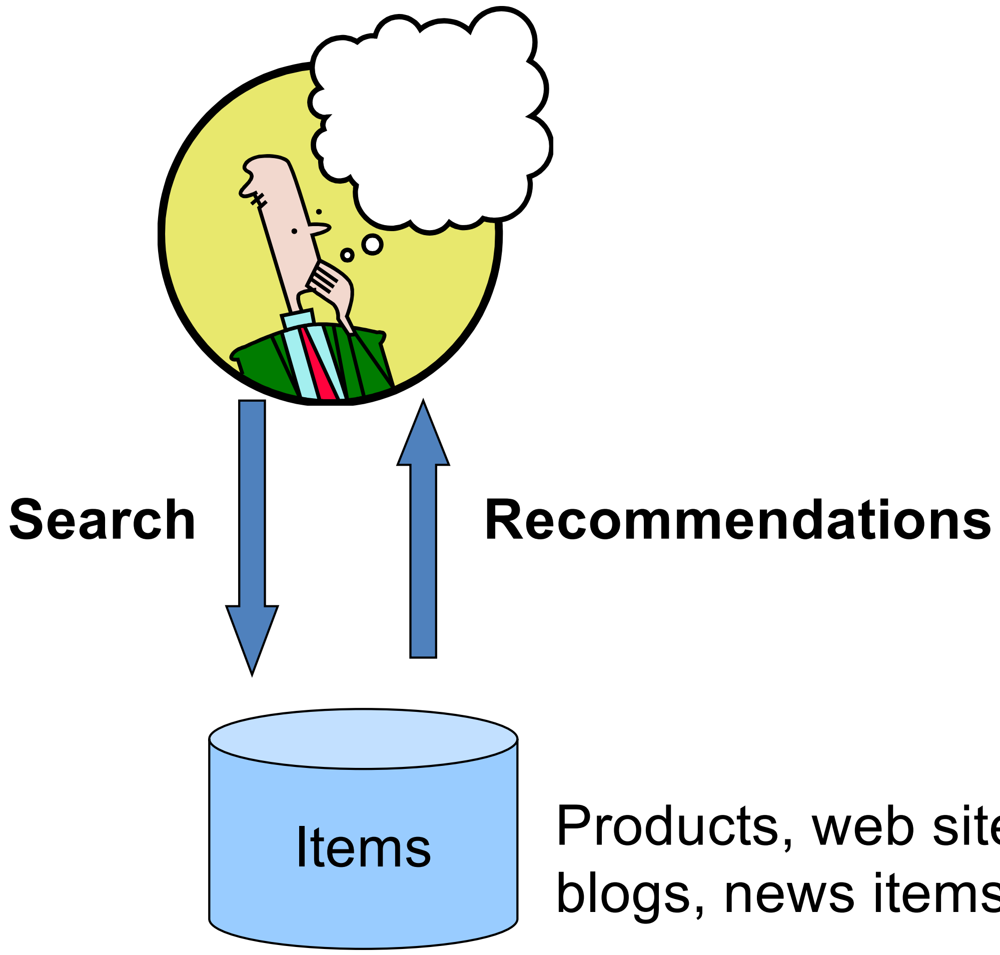
- Overview articles: Wikipedia pages on Recommender Systems and Collaborative Filtering:
  - http://en.wikipedia.org/wiki/Recommender_system
  - http://en.wikipedia.org/wiki/Collaborative_filtering
- Motivation: Article on The Long Tail
  - http://www.wired.com/wired/archive/12.10/tail.html
- Recommender Systems, Prem Melville and Vikas Sindhwani, Encyclopedia of Machine Learning, 2010
  - http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf
- A Survey of Collaborative Filtering Techniques
  - http://dl.acm.org/citation.cfm?id=1722966

Example: Recommender Systems



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

Recommendations



Products, web sites, blogs, news items, …

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

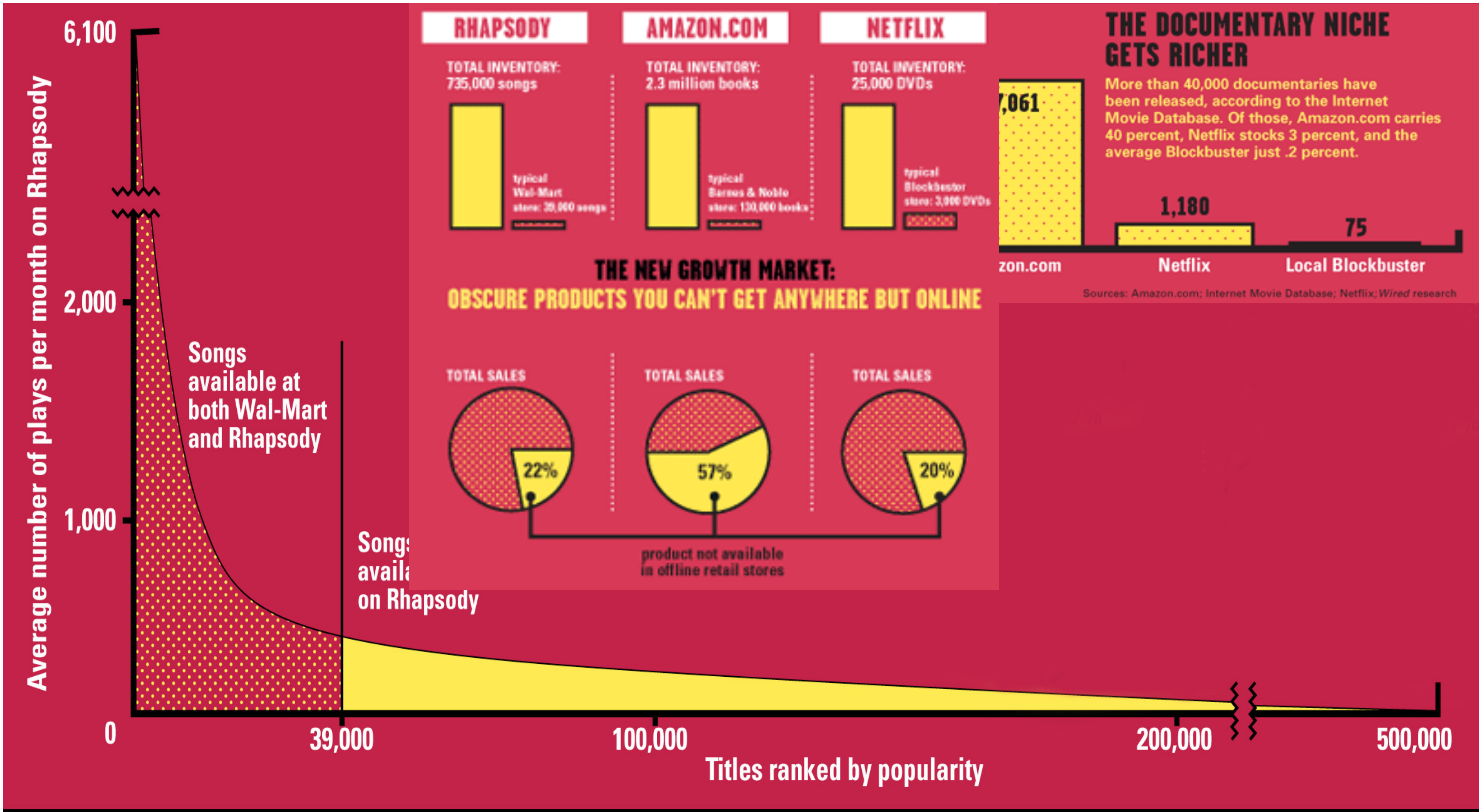
XBOX LIVE

Motivation: The Long Tail

From Scarcity to Abundance

- **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

The Long Tail



Sources: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks

Source: Chris Anderson (2004)

Change in thinking compared with online stores

- "What percentage of the top 10,000 titles in any online media store (Netflix, iTunes, Amazon, or any other) will rent or sell at least once a month?"
- Most people guess 20 percent
  - 80-20 rule, also known as Pareto's principle (1896)
  - Only 20 percent of major studio films, TV shows, books, etc. will be hits
- The right answer: 99 percent
  - Demand for nearly every one of those top 10,000 titles

Counterintuitive to old way of thinking

- The 20 percent rule in the entertainment industry is about ***hits***, not sales of any sort
  - **Hit-driven mindset**: think that if something isn't a hit, it won't make money
  - Makes sense with scarce shelf space in a retail store
  - iTunes, Amazon, and Netflix: discovered that **"misses" usually make money, too**
  - And because there are so many more of them, that money can add up quickly to a huge new market
- Industry has a poor sense of what people want
  - Turns out that people like a wide range of things when they are easily available

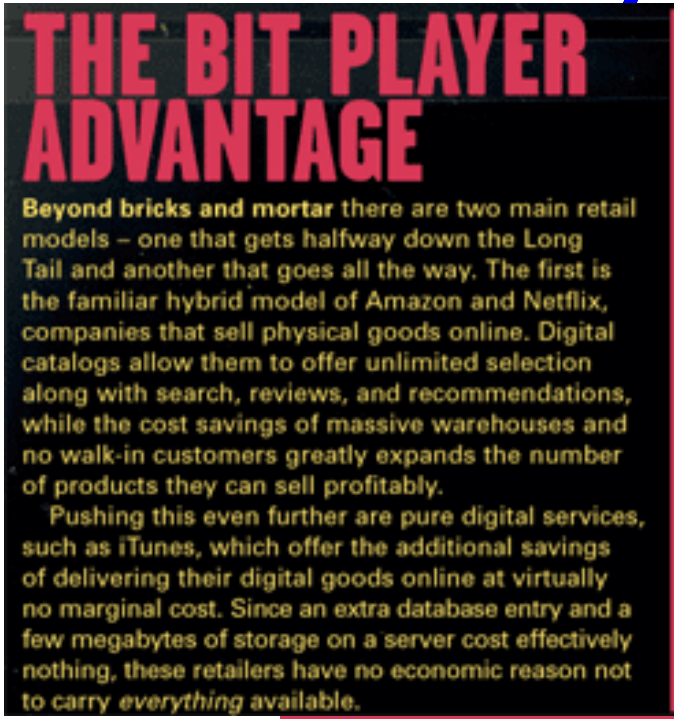
Rules of thumb

- **Rule 1: Make everything available**
  - Embrace under-served markets, niches (e.g., obscure video genres)
  - What matters is not where customers are, or how many of them are seeking a particular title, but that some number of them exist, anywhere
  - As a result, almost anything is worth offering on the chance it will find a buyer

Rules of thumb (cont.)

- **Rule 2: Lower costs**
  - Have taken away the unnecessary costs of the retail channel: manufacturing, distribution, and retail overheads
  - Leaves the costs of finding, making, and marketing content
  - Ensure that the people on the creative and business side still make money
- **Rule 3: Help users find new content**
  - Edited recommendations
  - Content-based recommendations
  - Collaborative filtering: uses browsing and purchasing patterns of users to guide those who follow them
  - "Customers who bought this also bought ..."
  - **Use recommendations to drive demand down the Long Tail**

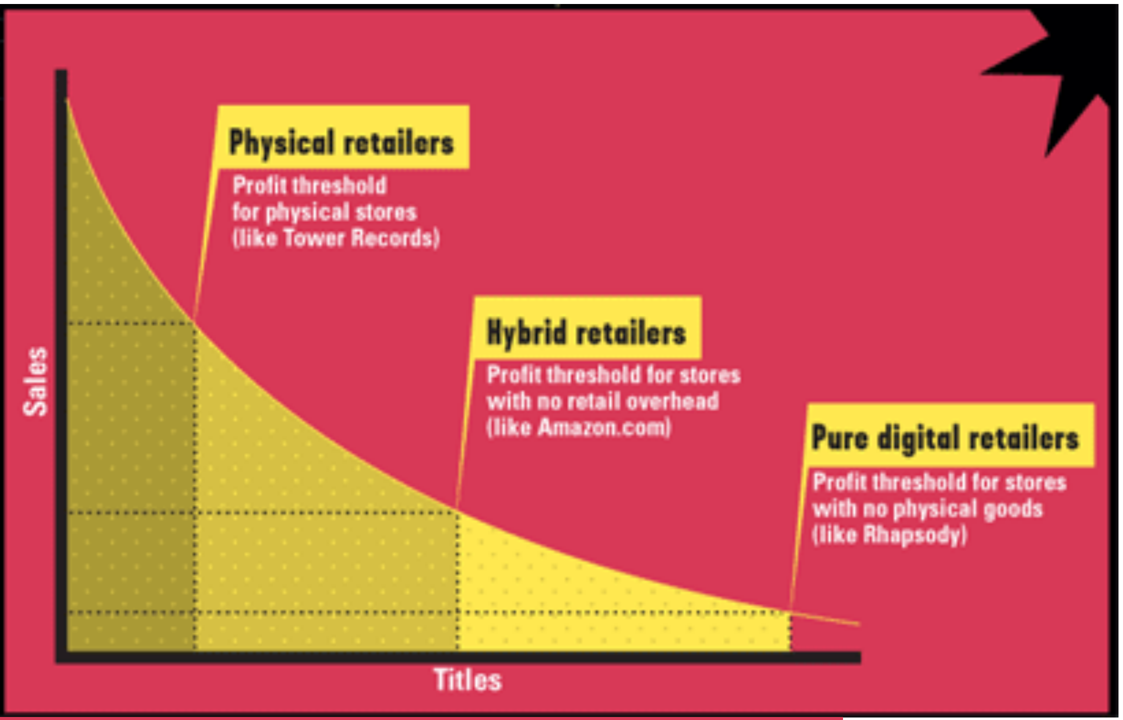
Physical vs. Online

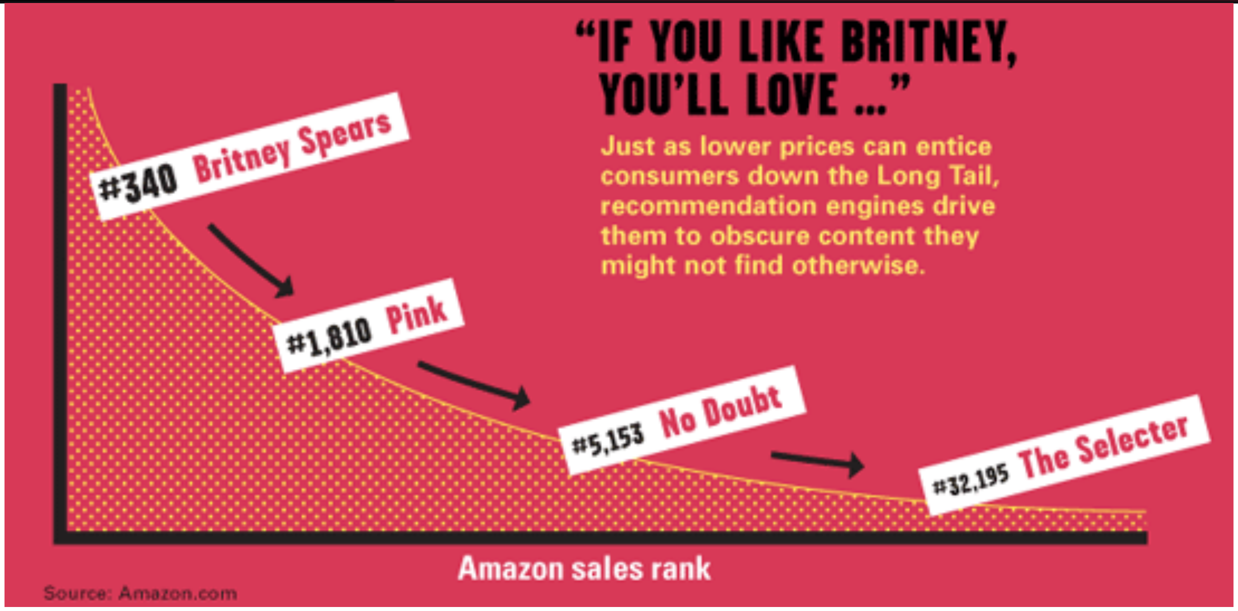


THE BIT PLAYER ADVANTAGE

Beyond bricks and mortar there are two main retail models – one that gets halfway down the Long Tail and another that goes all the way. The first is the familiar hybrid model of Amazon and Netflix, companies that sell physical goods online. Digital catalogs allow them to offer unlimited selection along with search, reviews, and recommendations, while the cost savings of massive warehouses and no walk-in customers greatly expands the number of products they can sell profitably.

Pushing this even further are pure digital services, such as iTunes, which offer the additional savings of delivering their digital goods online at virtually no marginal cost. Since an extra database entry and a few megabytes of storage on a server cost effectively nothing, these retailers have no economic reason not to carry *everything* available.





http://www.wired.com/wired/archive/12.10/tail.html

Types of Recommendations

Types 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, …

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**
- Users have preferences for certain items
- **Want to extract preferences from data**

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

Utility Matrix

- For each user-item pair, value represents degree of preference of that user for that item (e.g., rating)
- Matrix is sparse (most entries unknown)

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

Predictions

- Goal of a Recommendation System is to **predict the blanks in the utility matrix**
  - Would Alice like Pirates?
  - Would David like Avatar?
- Not necessary to predict every blank entry
- **Want to discover some entries in each row that are likely to be high**
- **Usually recommend a few items the user should value highly**
  - **Most likely to generate additional revenue**

Key Problems

- **(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
    - To generate revenue
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

(1) 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?

(2) Extrapolating Utilities

- **Key problem:** Utility matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - **New items have no ratings**
    - **New users have no history**

- **How to extrapolate missing entries?**

Three Approaches to Recommendation Systems

- **1) Content-based**
  - Use characteristics of an item
  - Recommend items that have similar content to items user liked in the past
  - Or items that match pre-defined attributes of the user
- **2) Collaborative filtering**
  - Build a model from a user's past behavior (items previously purchased or rated) and similar decisions made by other users
  - Use the model to predict items that the user may like
  - Collaborative: suggestions made to a user utilize information across the entire user base
- **3) Hybrid approaches**

Content-based Recommender Systems

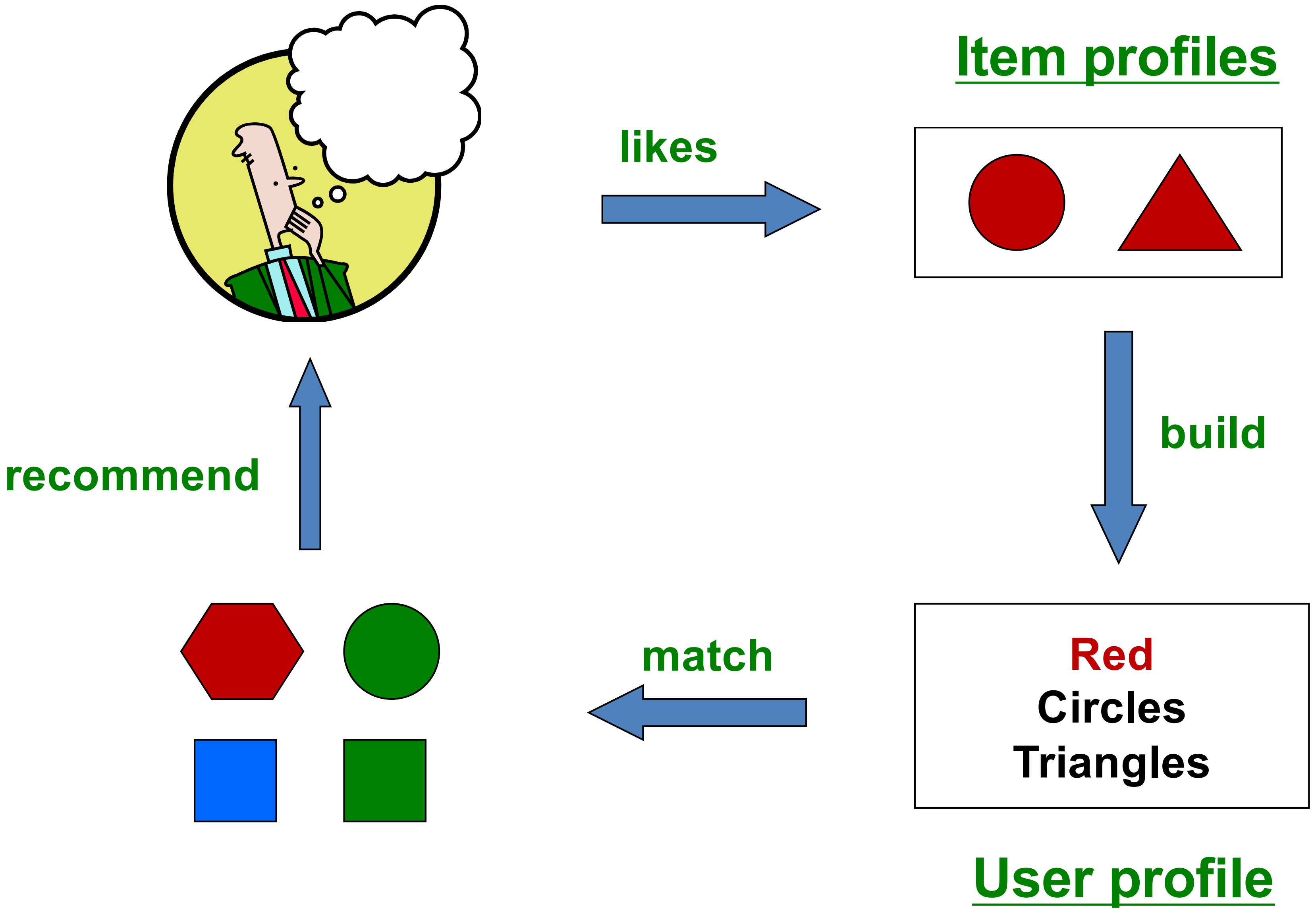
Content-based Recommendations

- **Main idea: Recommend items to customer *x* that are similar to previous items rated highly by *x***
  - ***Requires characterizing the content of items in some way***

Examples:

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

Plan of Action



General Strategy for Content-Based Recommendations

- **Construct item profiles**
  - Explicit features in a database, discovering features in documents, Tags
  - **Create vectors representing items**
    - Boolean vectors indicate occurrence of high TF.IDF word
    - Numerical vectors might contain ratings
- **Construct user profiles**
  - **Create vectors with same components that describe user's preferences**
- **Recommend items to users based on content**
  - **Calculate cosine distance between item and user vectors**
  - Classification algorithms

ITEM PROFILES

Item Profiles

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - **Movies:** screenwriter, title, actor, director,…
  - **Text:** Set of “important” words in document

- **Example 9.2**
  - Features of movies are set of actors and average rating
  - Each movie has 5 actors, two in both movies
  - Average ratings 3 and 4 (with unknown scaling factor alpha)
  - **Must scale non-boolean components so they are not dominant or irrelevant**

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

Cosine Distance (Section 3.5.4)

- **Cosine distance is used in spaces with dimensions, including Euclidean spaces**
  - Where points are vectors with integer or Boolean components
- Points thought of as directions
- Cosine distance between 2 points is the angle that the vectors to those points make
  - Range from 0 to 180 degrees
- **First compute cosine of the angle** between vectors x and y
- **Then apply arc-cosine function to translate to 0-180 degrees: the cosine distance**

Cosine Similarity
(Used to Calculate Cosine Distance)

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum\limits_{i=1}^{n} A_i \times B_i}{\sqrt{\sum\limits_{i=1}^{n} (A_i)^2} \times \sqrt{\sum\limits_{i=1}^{n} (B_i)^2}}$$

- **Dot product x.y divided by Euclidean distance of x and y from origin** (Section 3.5.2)
- Dot product of vectors:

  $[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n]$ is $\sum_{i=1}^{n} x_i y_i$.
- **Euclidean distance of two vectors x, y**

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Example 3.1.3: Cosine Distance

- x = [1,2,-1]   y = [2,1,1]
- **Dot product** x.y = 1x2 + 2x1 + (-1)x1 = 3
- **Euclidean distance** of two vectors x, y

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Euclidean distance of a vector from the origin: vector for origin is all zeros**
- Distance of x from origin is: $\sqrt{1^2 + 2^2 + (-1)^2} = \sqrt{6}.$
- Distance of y from origin also sqrt(6)
- Cosine of angle from x to y is $3/(\sqrt{6}\sqrt{6})$ or $1/2.$
- **Arccosine** of ½ is 60 degrees
- That is the **cosine distance between x and y**

Back to Item Profiles Example (9.2)

- Features of movies are set of actors and average rating (scaled)

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

- Dot product of these two item vectors is:

$$[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n] \text{ is } \sum_{i=1}^{n} x_i y_i.$$

  - Or $2 + 12\alpha^2$

- Distance from origin using

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Lengths of vectors (distance from origin) are:

$$\sqrt{5 + 9\alpha^2} \text{ and } \sqrt{5 + 16\alpha^2}.$$

- Cosine of angle between vectors is $\dfrac{2 + 12\alpha^2}{\sqrt{25 + 125\alpha^2 + 144\alpha^4}}$
- Scaling factor alpha affects how similar items are

ITEM PROFILES BASED ON TEXTUAL CONTENT

Item Profiles based on Textual Content

- Much research on content-based recommendations focuses on **textual content**
  - Recommend items (web pages, books, movies) based on associated textual content
  - Descriptions, user reviews
- Can treat this as an **Information Retrieval task (IR)**
- **How to identify whether two documents are about similar things?**
- **How to pick important features of documents?**
- Want to **identify the significant words** in documents

TF.IDF: Measure of Word Importance

- Classification of documents as being about similar things starts with finding significant words in those documents
- **Not most frequent words**
  - (the, and, a, …) – called "stop words"
- **Not just rare words either**
- **Want concentration of useful words in just a few documents**
- Usual heuristic from text mining is **TF-IDF:**

  (Term frequency * Inverse Doc Frequency)
- **Words with highest TF.IDF score are often the terms that best characterize the topic of a document**
- When constructing an item profile for Recommender system:
  - **Term … Feature**
  - **Document … Item**

TF-IDF: From Section 1.3.1 (Term frequency * Inverse Document Frequency)

f_{ij} **= frequency of term (feature)** ***i*** **in document (item)** ***j***

Term Frequency: $TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$

- Term frequency of term i in document j is **normalized**
- **Divide by maximum occurrences of any term in document j**
- Most frequent term has TF=1

n_i = number of docs that mention term ***i***

N = total number of docs

Inverse Document Frequency: $IDF_i = \log_2(N/n_i)$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Item profile for a document = **set of words with highest TF-IDF scores, together with their scores**

TF.IDF Example (Example 1.3)

- Repository of 2^{20} = 1,048,576 documents
- Suppose word w appears in 2^{10} = 1024 documents
- **Inverse document frequency:**

  $\boldsymbol{IDF_w = \log_2(2^{20}/2^{10}) = \log_2(2^{10}) = 10}$ (logarithm scaled)
- Consider document *j* in which word *w* appears 20 times
  - This is the maximum number of times any word appears in document j (after eliminating stop words)
  - So $\boldsymbol{TF_{wj} = 1}$
- **TF.IFD score for w in document j is 10**
- Consider document *k* where word *w* appears once
  - Maximum number of occurrences of any word in k is 20
  - So $\boldsymbol{TF_{wk} = 1/20}$
- **TF.IDF score for w in document k is 1/2**

Recommender Systems: Make Recommendations Based on Features of Documents

- **Want to suggest articles, pages, blogs a user might want to see**
- **Hard to classify items by topic**
- In practice, **try to identify words that characterize the topic of a document**
- **Eliminate stop words:** several hundred most common words
- **For remaining words, calculate the TF.IDF score** for each word in the document
- **The words with the highest TF.IDF scores characterize the document**

Represent documents by a set of words

- Take as features of the document the *n* words with highest TF.IDF scores
  - Could pick **same *n* for all documents**
  - Or let ***n* be fixed percentage** of words in the document
  - Could also make **all words with TF.IDF scores above a given threshold** are part of feature set
- Documents then represented by set of words
- Expect these words to express subjects or main ideas of documents
- Then can measure the similarity of two documents using:
  - Cosine distance between the sets, treated as vectors (last time)
  - Jaccard distance (Ch. 3) between sets of word

Document Similarity using Cosine Distance

- First compute cosine of the angle between vectors A and B

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

- Then apply arc-cosine function to translate to 0-180 degrees: the cosine distance

Recall: Item Profiles Example (9.2)

- Features of movies are set of actors and average rating (scaled)

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

- Dot product of these two item vectors is:

$$[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n] \text{ is } \sum_{i=1}^{n} x_i y_i.$$

  - Or $2 + 12\alpha^2$

- Distance from origin using

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Lengths of vectors (distance from origin) are:

$$\sqrt{5 + 9\alpha^2} \text{ and } \sqrt{5 + 16\alpha^2}.$$

- Cosine of angle between vectors is $\dfrac{2 + 12\alpha^2}{\sqrt{25 + 125\alpha^2 + 144\alpha^4}}$

Document Similarity using Cosine Distance

- **Think of set of high-TF.IDF words as a vector, with one component for each possible word**
- Vector has 1 if word is in the set for that document and 0 if not
- Between two documents, only a finite number of words among their two sets
- Almost all components are 0; do not affect dot product
- **Dot products** are size of intersection of the two sets of words
- **Lengths of vectors** are square roots of number of words in each set
- Cosine of angle between vectors: dot product divided by product of vector lengths

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Document Similarity: Two Kinds

- **Chapter 3: find (nearly) identical documents**
  - **Lexical similarity:** Documents are similar if they contain large fraction of identical sequences of characters
  - **Shingling:** convert documents to sets
  - **Minhashing:** convert large sets to short signatures, preserving similarity
  - **Locality-sensitive hashing (LSH):** Focus on parts of signatures likely to be from similar documents to **identify candidate pairs**
- **For Recommendation Systems (Chapter 9):**
  - Interested in occurrences of **many important words in both documents** (even if little lexical similarity between documents)

  Similar methodology:
  - **High TF.IDF words form a vector**, with a component for each possible word set to 1 or 0 *(analogous to sets of shingles)*
  - Based on a **distance measure** (Jaccard or cosine distance)

Another Option to Describe Item Content: Obtaining Item Profile Features from Tagging Systems

- Useful for content-based recommendations for **images**
- E.g., Flickr photosharing, Facebook, Instagram, Snapchat
- Users enter words or phrases that describe items
- GPS information/geofilters: e.g., automatically add location information when a photo is uploaded
- **Can use tags as a recommendation system**
  - E.g., if user retrieves or bookmarks pages with certain tags, recommend other pages with same tags
- Only works if users create tags or allow automatic geotagging

USER PROFILE

General Strategy for Content-Based Recommendations

- **Construct item profiles**
  - Source we discussed:
    - Explicit features in a database
    - Discovering features in documents
    - Tags
  - **Create vectors representing items**
    - Boolean vectors indicate occurrence of high TF.IDF word
    - Numerical vectors might contain ratings
- **Construct user profiles**
  - **Create vectors with same components that describe user's preferences**
- **Recommend items to users based on content**
  - Calculate cosine distance between item and user vectors
  - Classification algorithms

User Profiles (Examples 9.3 and 9.4)

- **Construct user profiles**
  - Create vectors with same components that describe user's preferences
  - Best estimate regarding which items a user likes is **some aggregation of the profiles of those items**
- **User profile possibilities:**
  - **Boolean utility matrix:** average the components of vectors representing item profiles for the items in which utility matrix has a 1 for that user
    - E.g., 20% of movies that user U likes have actor A (has a 1)
    - User profile for U will have 0.2 in component for actor A
  - **Non-boolean utility matrix: (e.g., ratings)** weight the vectors representing profiles of items by utility (rating) value
    - U gives average rating of 3; rates three movies with actor A: 3, 4, 5
    - Normalize by subtracting user's average rating: new ratings 0, 1, 2
    - Then user profile component for actor A will have value of 1
    - **Negative weights for below-average ratings, positive for above-avg.**

Content-Based Recommendations

- **Prediction heuristic**
  - Given user profile ***x*** and item profile ***i***
  - **Estimate degree to which a user would prefer an item by computing cosine distance between *x* and *i* vectors**
- **Classification Algorithms**
  - Use **machine learning techniques**
  - Regard given data as a training set
  - For each user, **build a classifier that predicts the rating of all items**
  - Ratings on a scale of 1 to k can be directly mapped to k classes
  - Many different classifiers
    - Naïve Bayes classifier
    - K-nearest neighbor
    - Decision trees
    - Neural networks

Example 9.5

- Building User Profile in Previous Example (9.4):
  - U gives average rating of 3; rates three movies with actor A: 3, 4, 5
  - Normalize by subtracting user's average rating: new ratings 0, 1, 2
  - Negative weights for below-average ratings, positive for above-average
- **Movie with many actors the user likes:**
  - **Cosine of angle will be large positive fraction**
  - **After applying the arccosine, will have a small cosine distance between vectors (angle close to 0 degrees)**
- **Movie with mix of actors the user likes/doesn't like: Cosine of angle will be around 0 (angle close to 90 degrees)**
- **Movie with many actors user doesn't like: Cosine will be a large negative fraction => large cosine distance between vectors (angle close to 180 degrees)**

Recommending items based on cosine distance

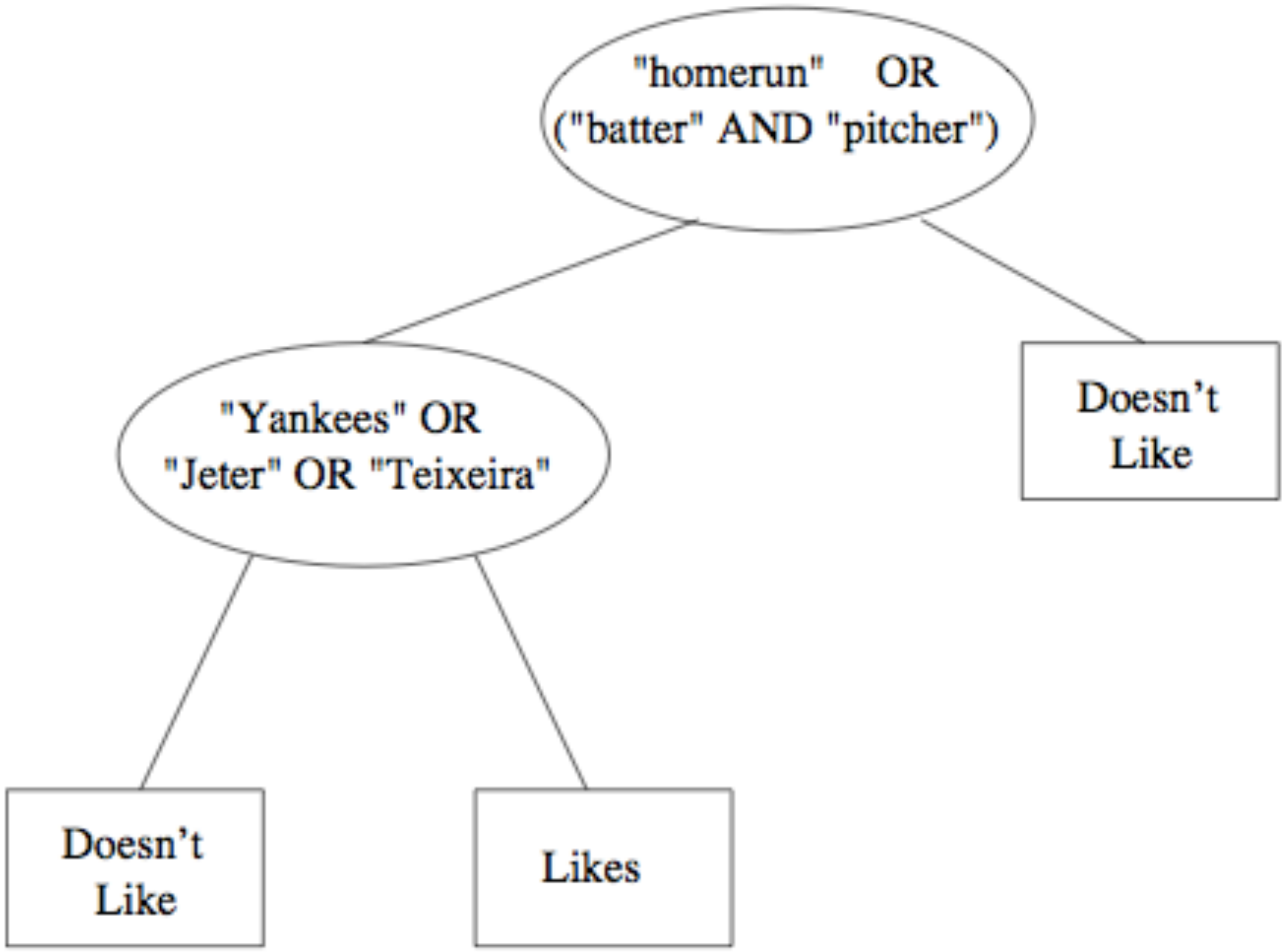
- Estimate degree to which a user would prefer an item by **computing cosine distance** between x and i vectors
- Scale components with values that are not boolean (e.g., ratings)
- Use Random hyperplanes (RH) and Locality Sensitive Hashing (LSH) techniques to place item profiles (i vectors) in buckets
- **For a given user (x vector), apply RH and LSH techniques: identify in which bucket we look for items that might have a small cosine distance from user**

DECISION TREES

Example 9.6: Decision Trees

- Items are **news articles**
- **Features are high TF.IDF words (keywords) in these documents**
- Suppose user U likes articles about baseball except articles about New York Yankees
- Row of utility matrix for U has a 1 if U has read the article, blank otherwise
- **Construct decision tree:**
  - select a predicate for each interior node
- **Classify an item:**
  - Start at root and apply predicate to the item
  - If predicate is true, go to left child; if false, go to right child
  - Repeat until a leaf is reached: leaf classified liked or not liked

Example 9.6 (cont.)


"homerun" OR ("batter" AND "pitcher")
"Yankees" OR "Jeter" OR "Teixeira"
Doesn't Like
Doesn't Like
Likes

Classifiers

- Classifiers of all types take a long time to construct
  - E.g., for decision trees: need one tree per user
- Constructing a tree requires looking at all item profiles
- Have to consider many different predicates
- Could involve complex combinations of features
- Typically applied only to small problem sizes

SUMMARY OF CONTENT-BASED APPROACH

Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start (for item) or sparsity problems (i.e., new items can receive recommendations)
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem (i.e., new products never have been rated, therefore they cannot be recommended)
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Recommendations for new users**
  - **How to build a user profile?**
- **–: Overspecialization**
  - Never recommends items outside user’s content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users (don’t use ratings!)**