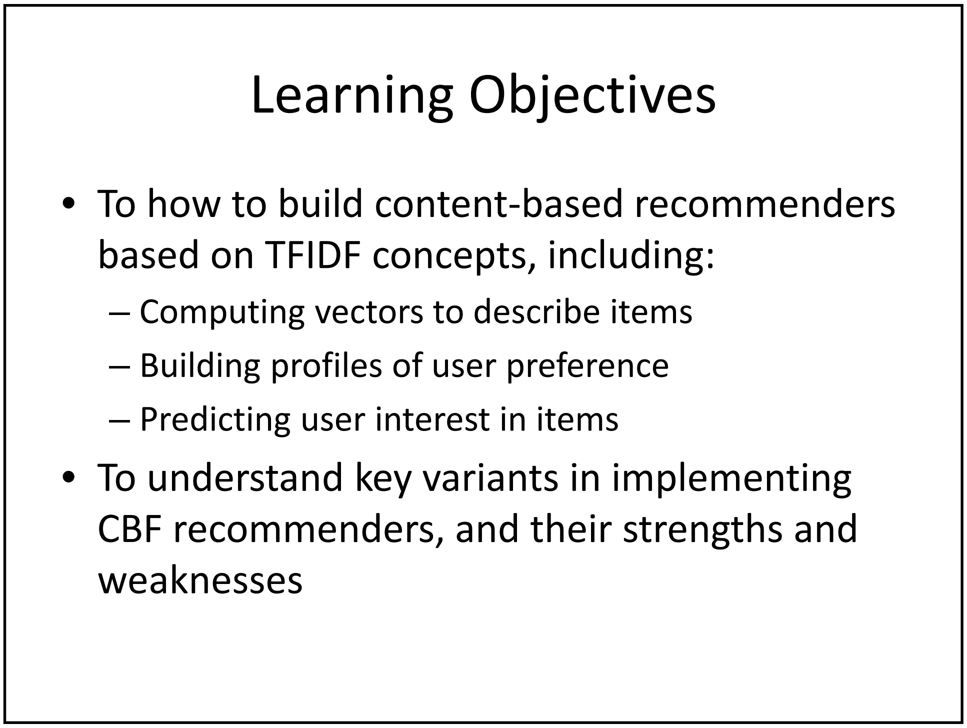
Content-Based Filtering: Deeper Dive



Learning Objectives

- To how to build content-based recommenders based on TFIDF concepts, including:
  - Computing vectors to describe items
  - Building profiles of user preference
  - Predicting user interest in items
- To understand key variants in implementing CBF recommenders, and their strengths and weaknesses

Key concept: Keyword Vector

- The universe of possible keywords defines a content space
  - Each keyword is a dimension
  - Each item has a position in that space; that position defines a vector
  - Each user has a taste profile (or more than one) that is also a vector in that space
  - The match between user preference and items is measured by how closely the two vectors align
  - May want to limit/collapse keyword space (e.g., stem and stop)

Useful reference

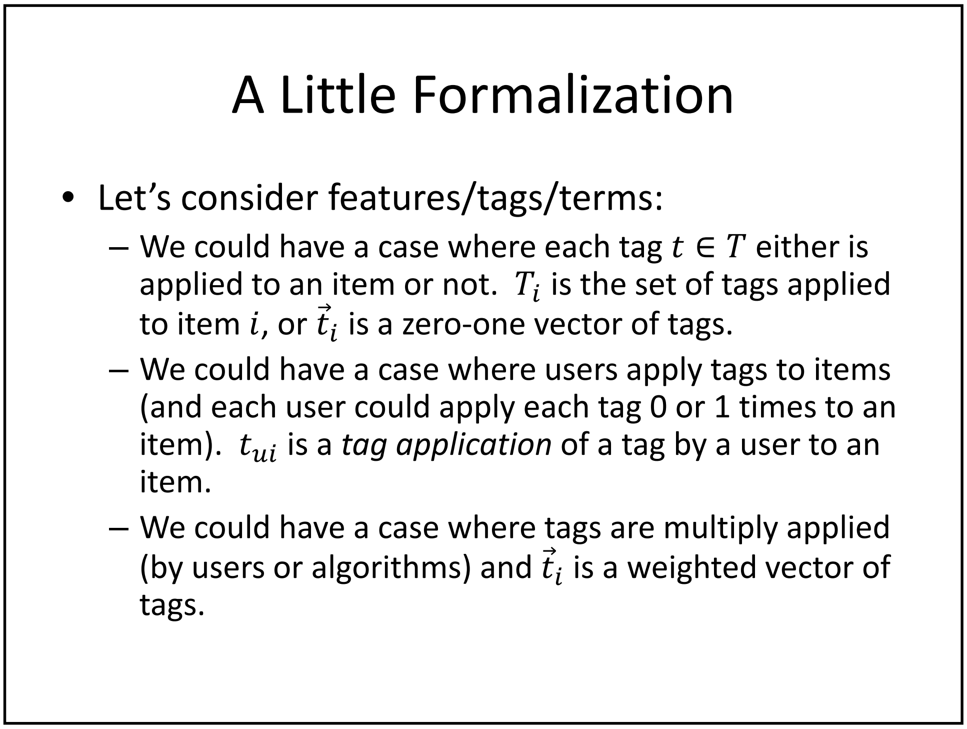
- Vector Space Model (originally conceived for queries, indexing)
  - http://en.wikipedia.org/wiki/Vector_space_model
  - Salton, Wong, and Yang (1995) “A Vector Space Model for Automatic Indexing,” *CACM 18*:11.

Where choices come into play …

- Representing an item through a keyword vector:
  - – Simple 0/1 (keyword applies or doesn't)
  - – Simple occurrence count
  - – TFIDF, most commonly:
    - # occurrences in doc * log (# docs / #docs with term)
  - – Other variants that include factors such as document length
  - – Eventually, this vector is often normalized

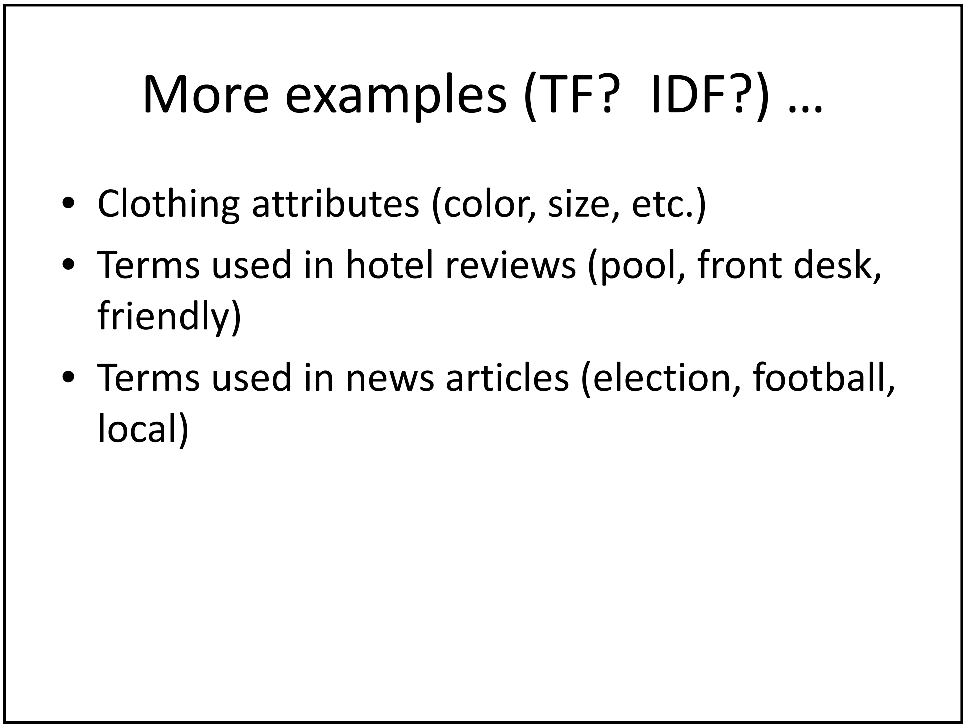
Consider the movie/tag case …

- Do we consider tags to be yes-or-no?
  - – Actor (we don't really get a measure for how much "Tom Hanks" a movie has)
  - – Descriptive (is how often a tag is applied a proxy for how relevant/significant the feature is?)
- Do we care about IDF?
  - – Actor (are infrequent actors more significant than stars?)
  - – Descriptive (is "prison scene" more significant than "car chase" or "romance"?)



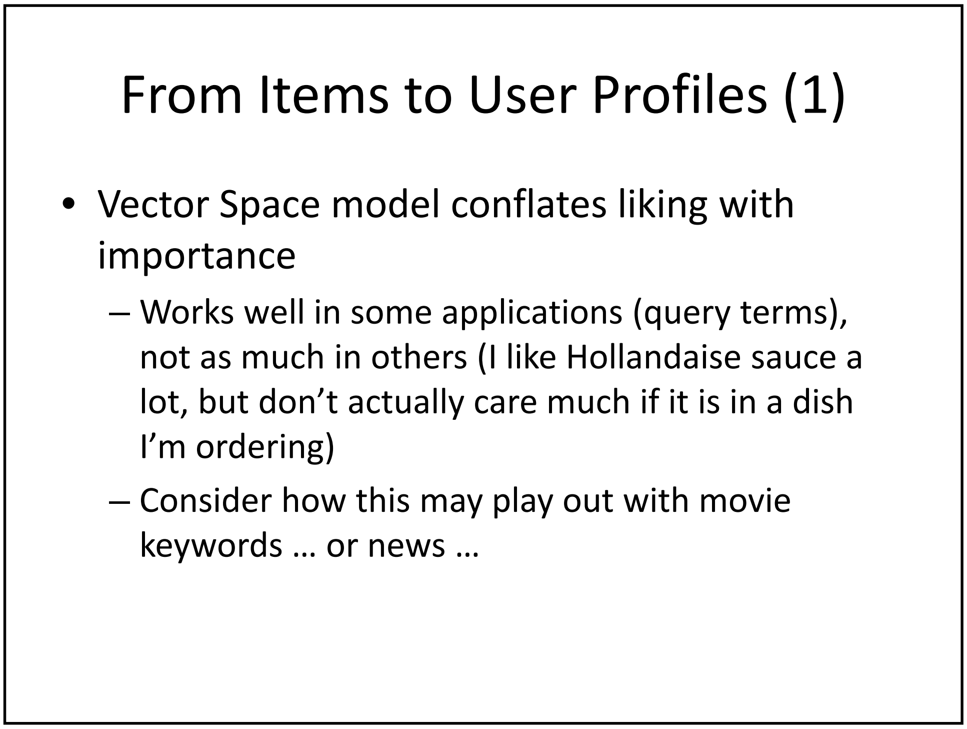
A Little Formalization

- Let’s consider features/tags/terms:
  - We could have a case where each tag $t \in T$ either is applied to an item or not. T_i is the set of tags applied to item i, or $\vec{t}_i$ is a zero-one vector of tags.
  - We could have a case where users apply tags to items (and each user could apply each tag 0 or 1 times to an item). t_{ui} is a *tag application* of a tag by a user to an item.
  - We could have a case where tags are multiply applied (by users or algorithms) and $\vec{t}_i$ is a weighted vector of tags.



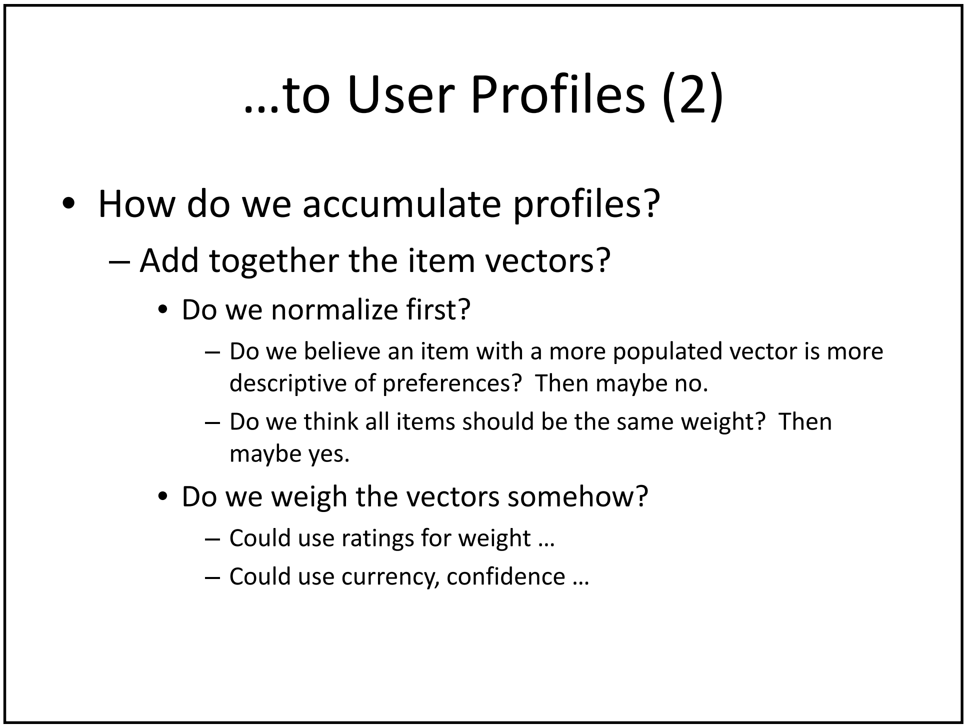
More examples (TF? IDF?) …

- Clothing attributes (color, size, etc.)
- Terms used in hotel reviews (pool, front desk, friendly)
- Terms used in news articles (election, football, local)



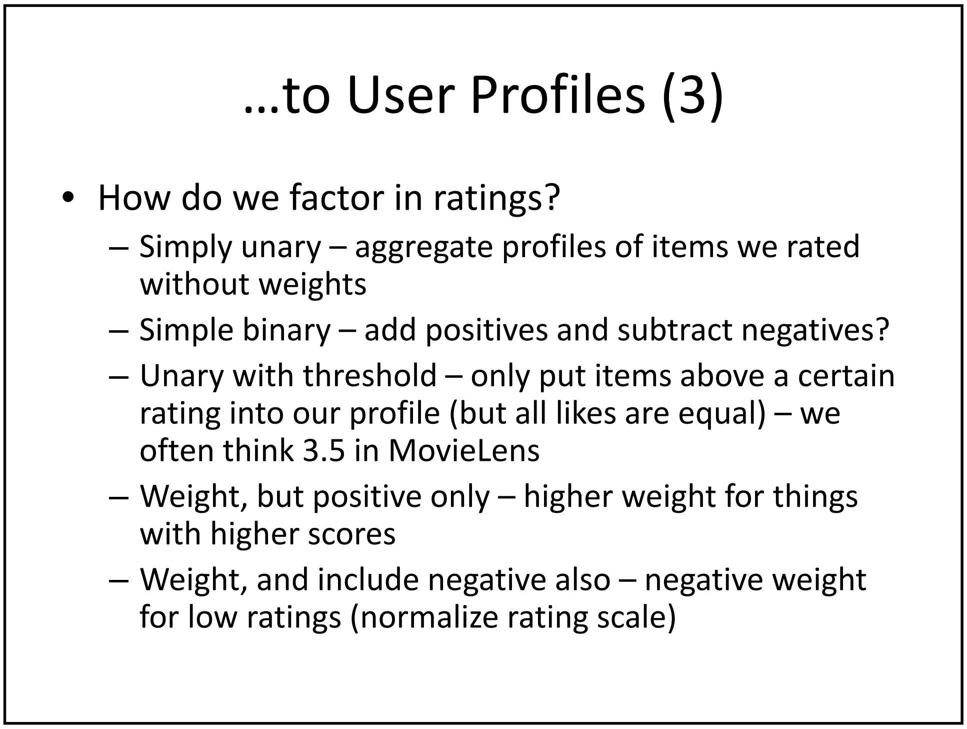
From Items to User Profiles (1)

- Vector Space model conflates liking with importance
  - Works well in some applications (query terms), not as much in others (I like Hollandaise sauce a lot, but don’t actually care much if it is in a dish I’m ordering)
  - Consider how this may play out with movie keywords … or news …



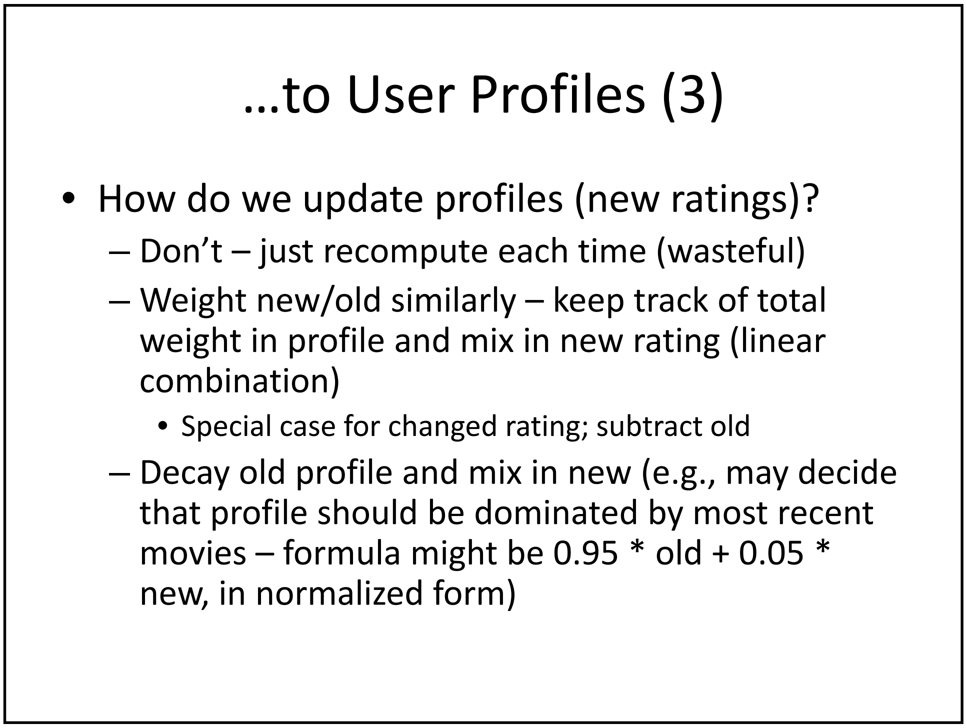
…to User Profiles (2)

- How do we accumulate profiles?
  - Add together the item vectors?
    - Do we normalize first?
      - Do we believe an item with a more populated vector is more descriptive of preferences? Then maybe no.
      - Do we think all items should be the same weight? Then maybe yes.
    - Do we weigh the vectors somehow?
      - Could use ratings for weight …
      - Could use currency, confidence …



…to User Profiles (3)

- How do we factor in ratings?
  - – Simply unary – aggregate profiles of items we rated without weights
  - – Simple binary – add positives and subtract negatives?
  - – Unary with threshold – only put items above a certain rating into our profile (but all likes are equal) – we often think 3.5 in MovieLens
  - – Weight, but positive only – higher weight for things with higher scores
  - – Weight, and include negative also – negative weight for low ratings (normalize rating scale)



…to User Profiles (3)

- How do we update profiles (new ratings)?
  - – Don’t – just recompute each time (wasteful)
  - – Weight new/old similarly – keep track of total weight in profile and mix in new rating (linear combination)
    - Special case for changed rating; subtract old
  - – Decay old profile and mix in new (e.g., may decide that profile should be dominated by most recent movies – formula might be 0.95 * old + 0.05 * new, in normalized form)

Computing Predictions …

- Prediction is the cosine of the angle between the two vectors (profile, item)
- This is the dot-product of normalized vectors, or if you prefer, the dot product divided by the product of the two lengths
- Cosine ranges between -1 and 1 (0 and 1 if all positive values in vectors) – closer to 1 is better.
- Top-n, or scale for rating-scale predictions.

Strengths of this Approach

- Entirely content-based
- Understandable profile
- Easy computation

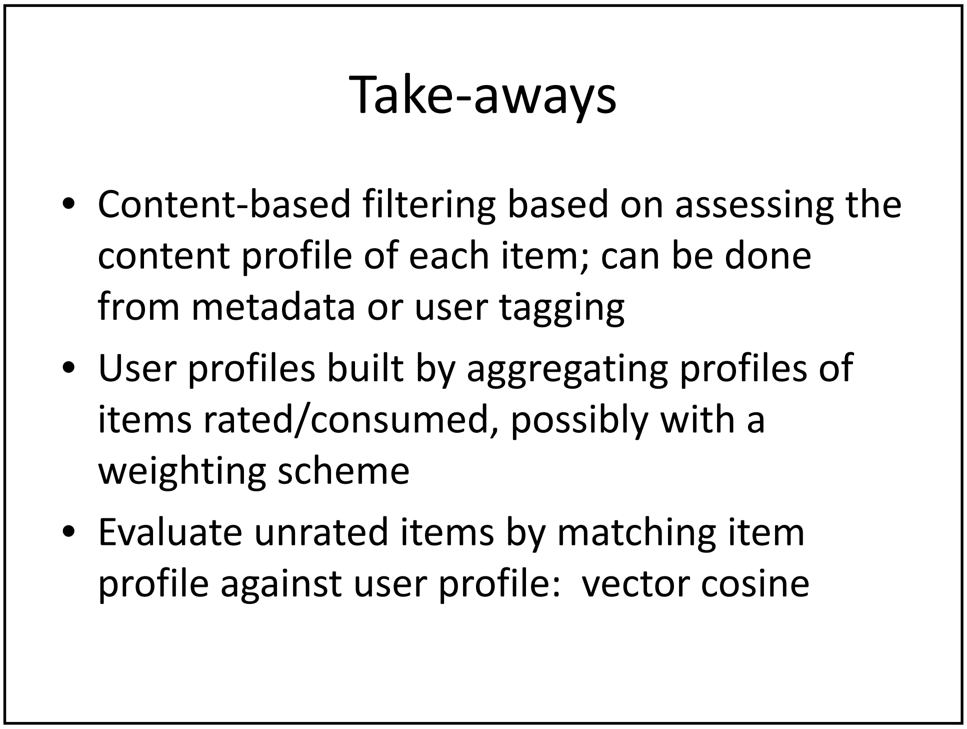
- Flexibility – can integrate with query-based systems, case-based approaches

Challenges

- Figuring out the right weights and factors
  - Is more *more*, or just reiteration of the same
  - How to deal with ratings

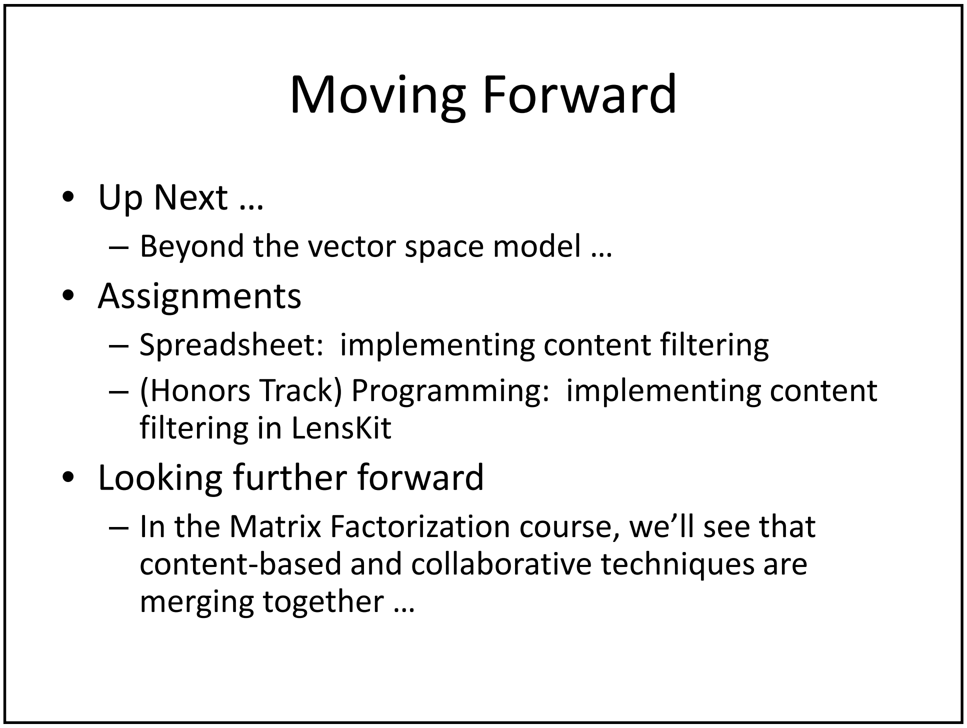
Limitations

- This is a highly simplified model, cannot handle interdependencies
  - I like Sandra Bullock in Action movies, but Meg Ryan in Romantic Comedy movies
  - I like comedies with violence, and historical documentaries, but not historical comedies or violent documentaries



Take-aways

- Content-based filtering based on assessing the content profile of each item; can be done from metadata or user tagging
- User profiles built by aggregating profiles of items rated/consumed, possibly with a weighting scheme
- Evaluate unrated items by matching item profile against user profile: vector cosine



Moving Forward

- Up Next …
  - Beyond the vector space model …
- Assignments
  - Spreadsheet: implementing content filtering
  - (Honors Track) Programming: implementing content filtering in LensKit
- Looking further forward
  - In the Matrix Factorization course, we'll see that content-based and collaborative techniques are merging together …

Content-Based Filtering: Deeper Dive