Computer Science

CSC380: Principles of Data Science

Advanced ML algorithms

Kyoungseok Jang

Announcement

- Reading quiz today
  - Already announced quiz questions on Piazza.

- Uploaded Final practice exam.

Outline

- Random Forest (supervised learning, advanced decision tree)
- Recommendation algorithms (unsupervised learning)
- Generative Adversarial Network (generative model)

Disclaimer

- Today’s lecture will not be included in your final exam.
- You don’t need to study this part too hard.

Random Forest

Review: Decision Tree

Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

```
guess ← most frequent answer in data                 // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                               // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                               // base case: cannot split further
else                                                 // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← ( # of majority vote answers in NO
                     + # of majority vote answers in YES ) /
                   size(data)
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

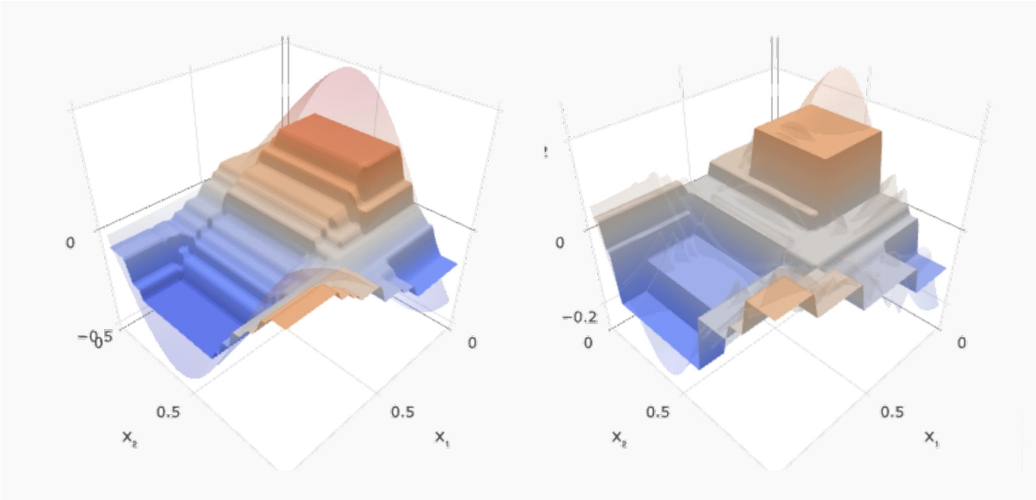
<= i.e., all data points have the same label

<= there is no point in adding a feature that appeared in its parent!

<= answer = label

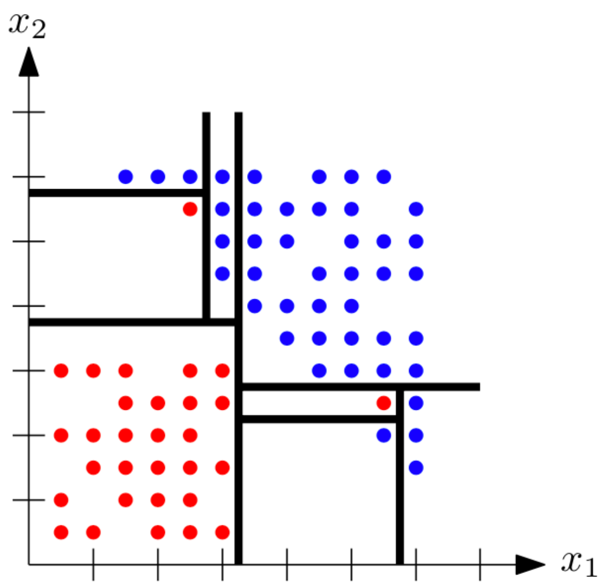
Advantages of Decision Tree

- Pros
  - Interpretability
  - Less data preparation (preprocessing)
  - Non-parametric: not like Naïve-Bayes, it does not require complicated model assumption.
  - Versatility
  - Non-linearity



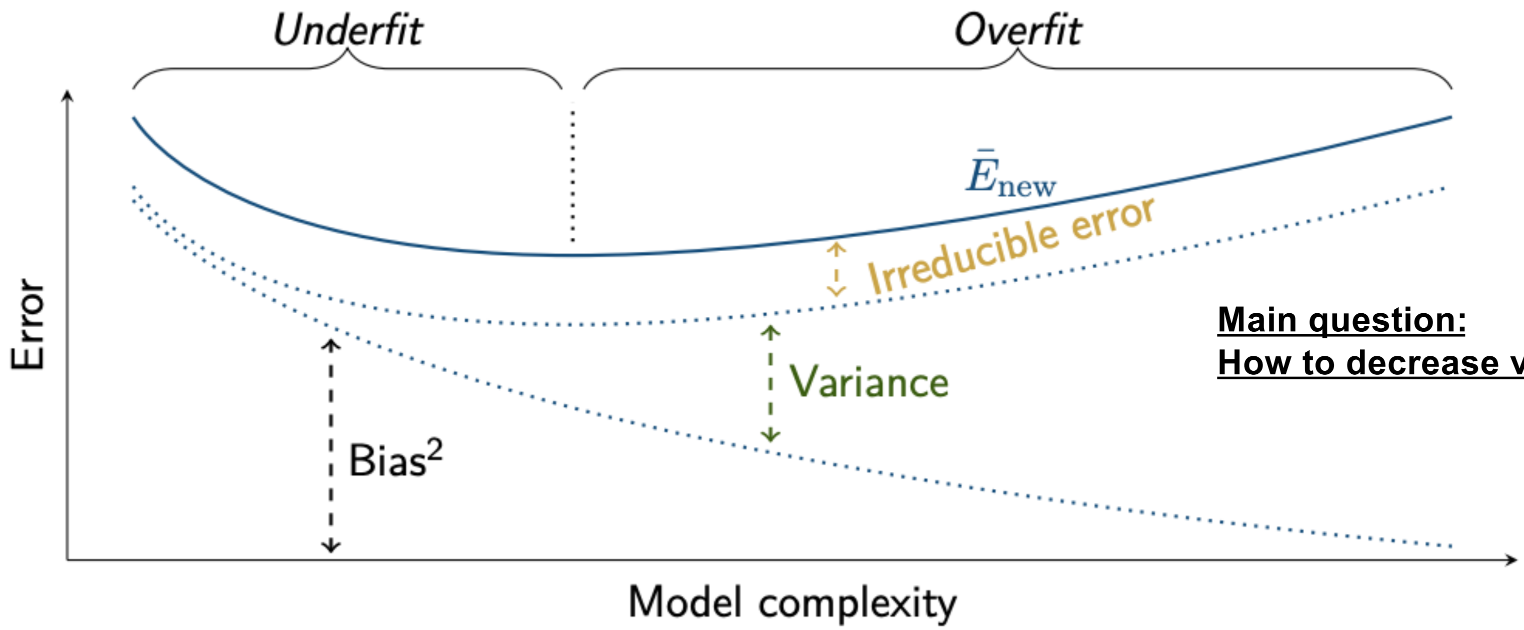
Disadvantages of Decision Tree

- Cons
  - **Overfitting: model can be complex = vulnerable to overfitting**
  - **Optimization: order** $O(dm^2 + dm \log m)$
    - **especially for the continuous feature → Too many features to consider.**



Pruning is usually not enough
+ Additional complicated computation

Bias-Variance Tradeoff



Main question:
How to decrease variance?

Finding a balanced fit (neither over- nor underfit) is called the **the bias-variance tradeoff**.

Bias-Variance Tradeoff

- Intuition: For i.i.d random variables $X_1, \cdots, X_n$

$$Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = Var(X_1)$$

- Observation: If we have D independent (and large enough) dataset, then we can train D individual decision trees and do 'majority vote' to decrease variance!

- Problem
  - D datasets should be independent! We usually don't have enough data to split it by D large enough subsets!
  - Alternative: Bootstrapping

Bagging

	training examples					
	#1	#2	#3	#4	#5	#6
original dataset	1	1	1	1	1	1
decision tree 1	1	1	0	2	1	1
decision tree 2	3	0	1	0	2	0
decision tree 3	0	1	3	1	0	1

- Bagging = **B**ootstrap + **Agg**regat**ing**
- **Objective: Create multiple decision trees**
- **Bootstrapping:**

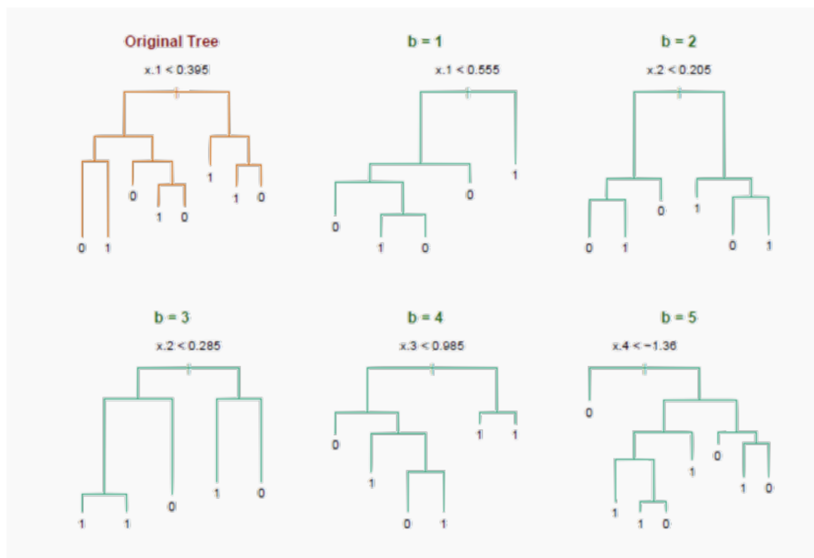
Generate multiple samples of training data via bootstrapping (sampling from your dataset)

Each tree will be trained based on this sampled dataset.

- **Aggregating:** For a given input, we output
  - Regression: the averaged outputs of all the models for that input.
  - Classification: the class that is outputted by the majority

Bagging

- This is one example of **ensemble method**
  - Method of building a single model by training and aggregating multiple models



Out-of-bag evaluation

- How to evaluate generalization errors?
  - Validation set? Yes, it works, but… it would be better if we don't split!
  - Turns out we don't need to split!
  - Traditionally, ~67% of original data is in a sampled dataset.
  - You can use the rest of the data as your validation set for each tree!

	Training examples						Examples for OOB Evaluation
	#1	#2	#3	#4	#5	#6	
original dataset	1	1	1	1	1	1	
decision tree 1	1	1	0	2	1	1	#3
decision tree 2	3	0	1	0	2	0	#2, #4, and #6
decision tree 3	0	1	3	1	0	1	#1 and #5

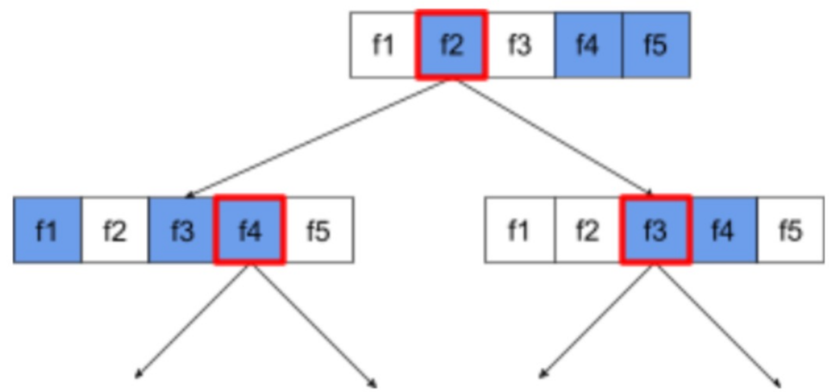
Good, are we done?

- Not yet… are those trees 'truly' independent?
- These ensemble models are meaningless when models are dependent on each other.
  - E.g.) $X_1, \cdots, X_n$: All equal random variable ($X_i = X$)
  - Then $\mathrm{Var}(\frac{1}{n}\sum_{i=1}^{n} X_i) = Var(X) = Var(X_1)$: variance does not decrease!
- In practice, these ensembles of trees in Bagging tend to be highly correlated!
  - E.g.) One extremely strong predictor, x_j, in the training set amongst moderate predictors.
  - Then most of your trees will start with x_j, and no big changes.
  - Multiple identical trees!

Random Forest

- How do make trees 'different' from each other?
  - Don't allow your algorithm to use all the features!

1. Create separate bootstrap samples (same as bagging)
2. For each tree, at each split (node training), **<u>we randomly select a set of J′ predictors from the full set of predictors.</u>**
3. Find the best feature among J'



Blue: randomly selected candidate J'
Red box: chosen feature for this node

Random Forest - properties

- Parallel training is possible → Not that heavier optimization than one decision tree

- Three main hyperparameters to tune
  - Number of predictors to randomly select for each split (node)
  - Depth of the tree, or the minimum leaf node size (complexity of tree)
  - Number of trees: turns out, increasing number of trees **does not** increase variance!

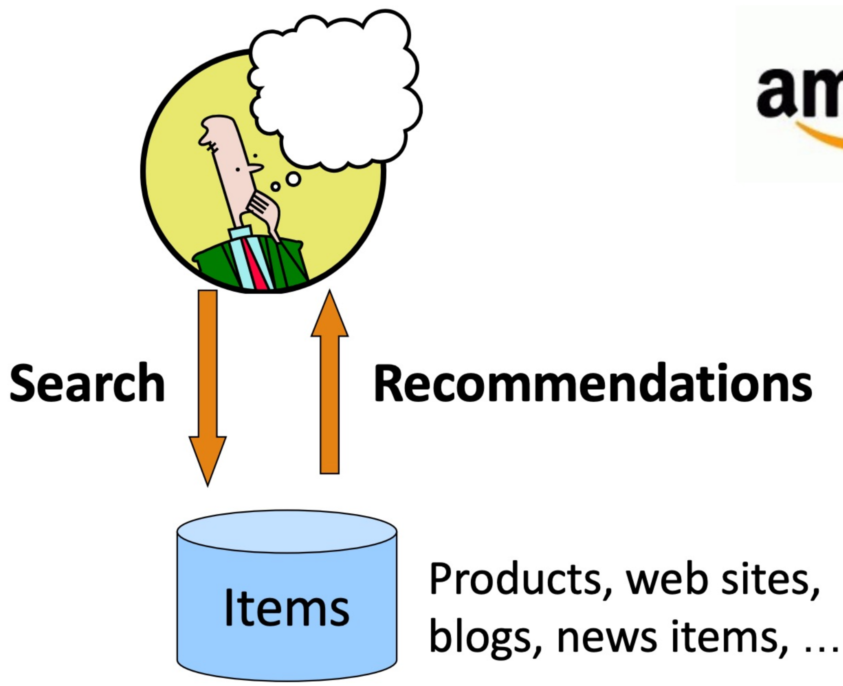
- When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

Pros and Cons

- Pros
  - **Hard to overfit: "Random Forests do not overfit" – Leo Breiman**
  - Even without worrying about overfitting, one can create a complex model!
  - Inherits most of the advantages of decision trees
    - Less data preparation (preprocessing)
    - Non-parametric
    - Versatility
    - Non-linearity
  - Can be trained parallel - Faster learning relative to its size
- Cons
  - **Lose Interpretability.**
  - **The size of the model can be large.** Models with more than 1M nodes are common.
  - Random forests cannot learn and reuse internal representations.

Recommendation algorithms

Recommendations are everywhere



Types of recommendations

- Editorial and hand-curated
  - List of ‘essential’ by XYZ
  - Personal recommendation of critic ABC
- Simple Aggregation
  - Top 100, recent uploads, most popular
- Tailored to individual users
  - Amazon, Youtube, Netflix, etc… ⟵ Today’s main topic

The task

- Given the information of a user, what should we recommend next?
- Formal model

- X=set of users
- S=set of items
- Utility function $u: X \times S \rightarrow R$
  - R: set of ratings (e.g. 1-5 stars, real numbers in [0,1])
- Want to maximize customer’s happiness! (so that the company can earn more)

The task

- This model naturally induces ‘utility matrix’

		Harry Potter			Twilight	Star Wars		
		HP1	HP2	HP3	TW	SW1	SW2	SW3
Anita	A	4			5	1		
Beyonce	B	5	5	4				
Calvin	C				2	4	5	
David	D		3		?			3

- How to fill out the ‘empty spaces?’
- If we can ‘guess’ the numbers in empty spaces, we can recommend best item for the customer?

Two approaches

- Content-Based Filtering
  - Recommend items to customer x similar to previous items rated highly by x.
  - User-Item interaction
- Collaborative Filtering
  - Find a set of other users whose ratings are “similar” to x’s ratings
  - 1) User-user interaction (straightforward)
  - 2) Item-item interaction
  - 3) Model-based: google developer version

Content-based filtering

- For each item, create an item profile (vector)
  - E.g.) Movie: genre, director, actor, year, …

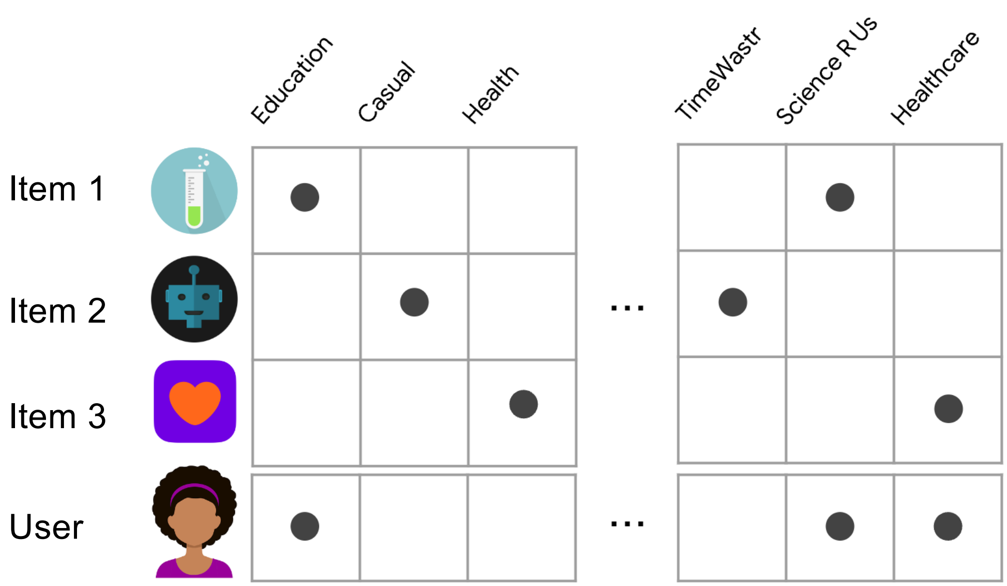
	Melissa McCarthy	Actor A	Actor B	…	Johnny Depp	Comic Genre	Spy Genre	Pirate Genre
Movie X	0	1	1	0	1	1	0	1
Movie Y	1	1	0	1	0	1	1	0

- For each user, create a user profile (vector) with same features!
  - E.g.) Average out all movies that user have seen.

	Melissa McCarthy	Actor A	Actor B	…	
User U	0.2	.005	0	0	…

Content-based filtering

- Measure the ‘similarity score’
  - Various types of scores: dot product, cosine similarity, Pearson similarity, Euclidean distance
- Recommend an item with the highest similarity score



If we use the dot product as our similarity score, what item should we recommend to the user?

Pros and Cons

- Pros
  - No need for data from other users
  - Able to recommend to users with unique tastes
    - No first-rater problem: if the movie is well featured
  - Able to provide an explanation: ‘Since you liked …’
- Cons
  - How to determine features appropriately?
  - Frequently involves hand-engineering
  - **Overspecialization** – hard to introduce or expand new content
    - Never recommend content outside the user’s profile

Collaborative Filtering

- Instead of somewhat hand-engineered features, we will use the utility matrix directly!
  - Since it uses a real dataset, it will reflect reality better.
- But how?
  - 1) User-to-user interaction: directly compute using rows!
  - 2) Item-to-item interaction: directly compute using columns!
  - 3) Model-based interaction: create a content-based filtering model latently!

		Harry Potter			Twilight	Star Wars		
		HP1	HP2	HP3	TW	SW1	SW2	SW3
Anita	A	4			5	1		
Beyonce	B	5	5	4				
Calvin	C				2	4	5	
David	D		3					3

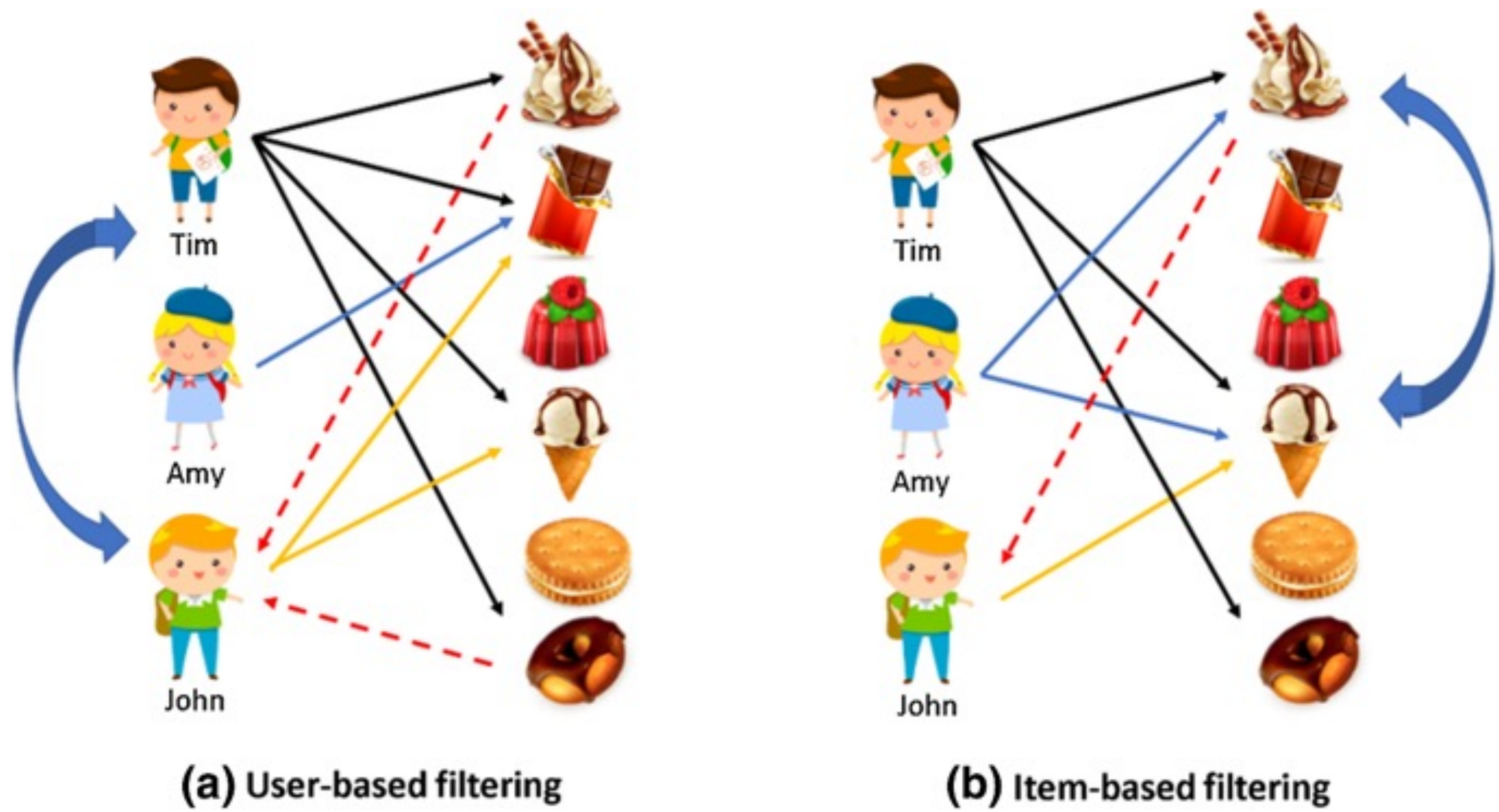
Collaborative Filtering – User-user based

- Basic concept: Find similar users and recommend items that they like!
- How to define similarity?
  - Use the i-th row of the utility table (items that user i has used) as the feature vector of user i
  - Use the similarity measure we explained before!
    - Cosine measure, dot product, Pearson similarity, …
    - There are lots of details added for this process, but I will skip them.
- The importance of the user is proportional to the similarity.
  - Because it is highly probable that similar people with input user x will tell you more about the taste of x.

Collaborative Filtering – User-to-user based

- There are lots of variants for this user-user based CF, but here's one example:

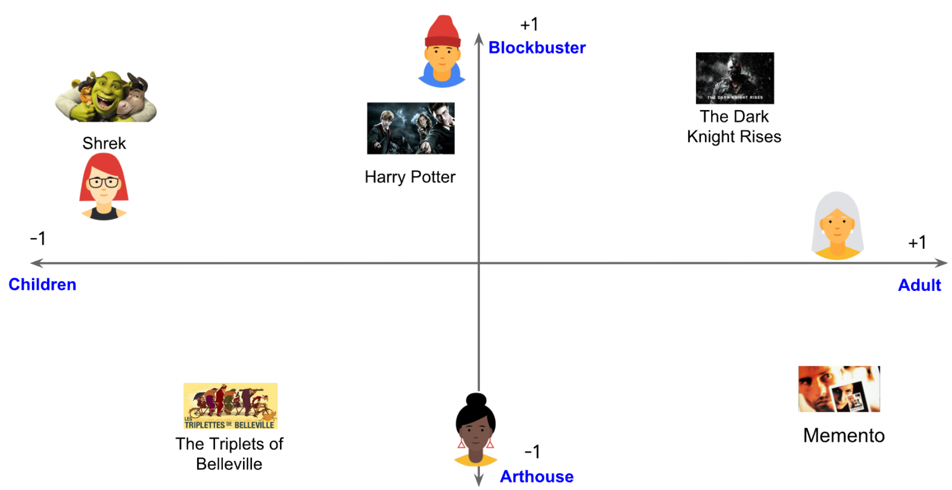
- r_x: the vector of the user x's ratings
- N: be the set of k 'neighborhoods' (k most similar to x)
- Prediction for the item i of user x (can be vary):
- $r_{xi} = \frac{\sum_{n \in \mathrm{N}} sim(x,n) r_n}{\sum_{n \in N} sim(x,n)}$
- Recommended item $i^* = \arg\max_i r_{xi}$



Tim
Amy
John
(a) User-based filtering
Tim
Amy
John
(b) Item-based filtering

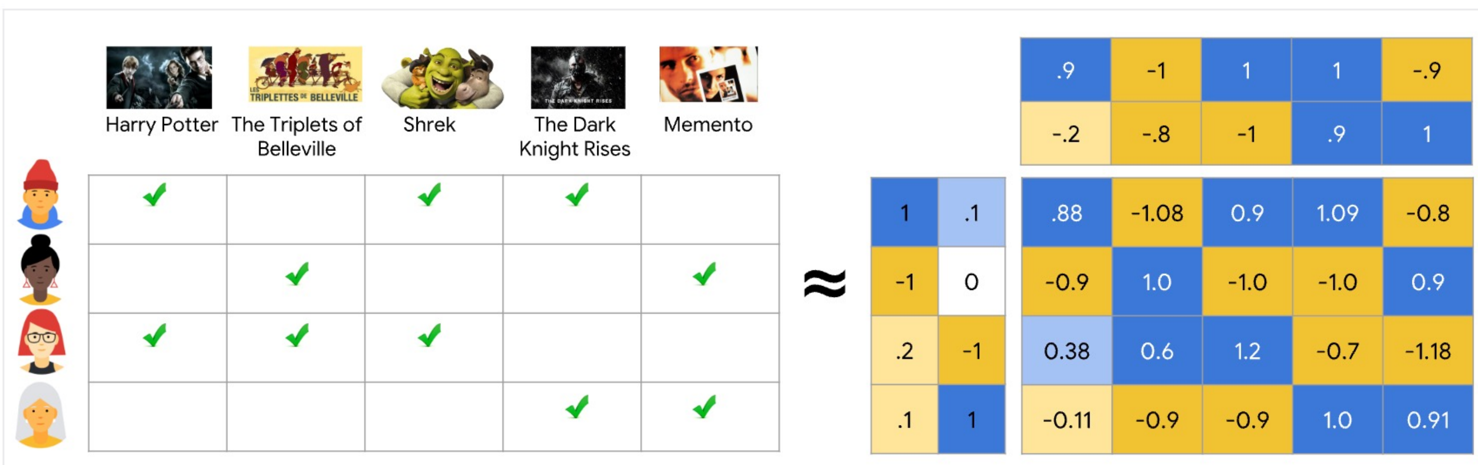
Collaborative Filtering: Model-based

- Imagine that we already have hidden ‘embedding’ – feature vectors that affects utility.
- Then our prediction will be easy – as we did in Content-based filtering. E.g.) Inner product between user vec and item vec!



Collaborative Filtering: Model-based

- It will be great if we can determine the features 'automatically'
- But how? Using matrix factorization…
- Suppose that there exists a matrix $U \in \mathbb{R}^{m \times d}$ (features of users) and $V \in \mathbb{R}^{n \times d}$ (features of items) such that $UV^\top$ represents the true utility matrix.



Collaborative Filtering: Model-based

- If this model is true, then it should satisfy
  - If the actual rating of user i for item j is A_{ij}, then this model should predict similar rating for this user-item pair, which means
    - $\left(A_{ij} - \langle U_i, V_j \rangle\right)^2$ should be small for all observed i and j.
  - For unobserved items, it might 'implicitly' mean that the user is not interested in that items.
    - So, it would be great $\left(0 - \langle U_i, V_j \rangle\right)^2$ is also small for unobserved
- You need to optimize

$$\underset{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}}{\arg\min} ||A - UV^\top||_F^2$$

Where $||B||_F^2 = \sum b_{ij}^2$, sum of squares of all entries in matrix.

- Turns out, it can be done efficiently!

Pros and Cons

- Pros
  - **No domain knowledge:** everything can be done automatically and mathematically. Only needs the feedback matrix!
  - **Serendipity:** The model will help you discover your new interests based on the interest of other users.
  - Relatively robust and performs well in practice.
- Cons
  - **Cold-start problem:** new user or new item – no similarity to use!
  - **Interpretability:** all the features are created ‘automatically’, so it is hard to understand what each feature means.
  - **Popularity bias:** tend to recommend popular one – someone with unique taste will not like it.
  - Sometimes ethical issues…