Recommendation Systems-II

CS771: Introduction to Machine Learning

Purushottam Kar

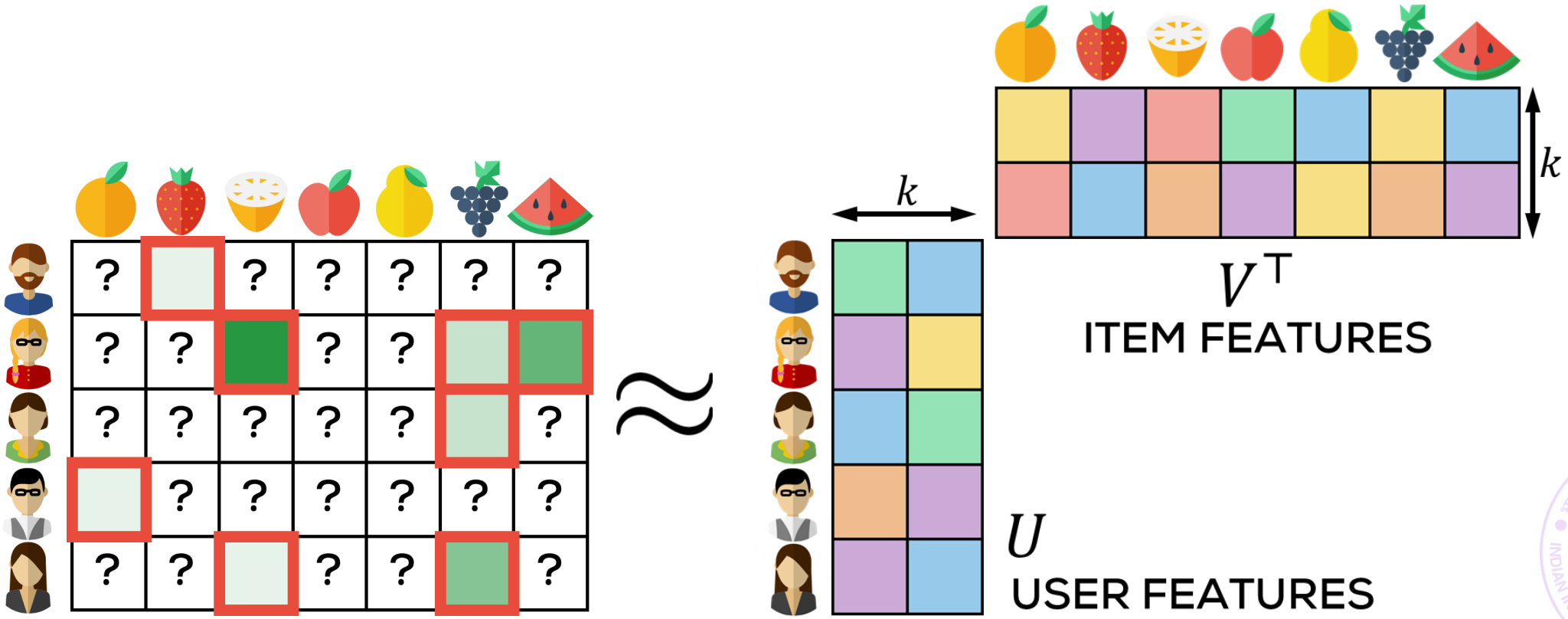
Recap

Recap

- Collaborative filtering uses low-rank matrix completion to fill up entries of a partially observed ratings matrix

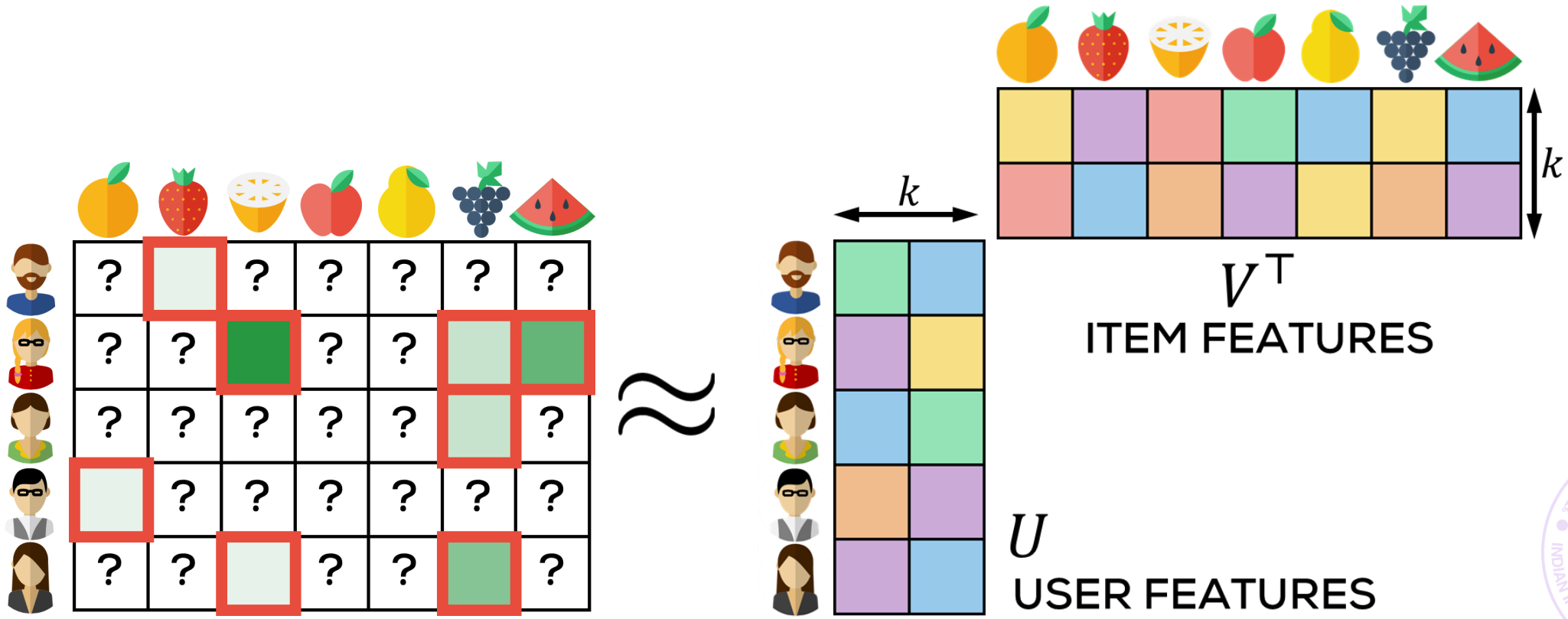
Recap

- Collaborative filtering uses low-rank matrix completion to fill up entries of a partially observed ratings matrix



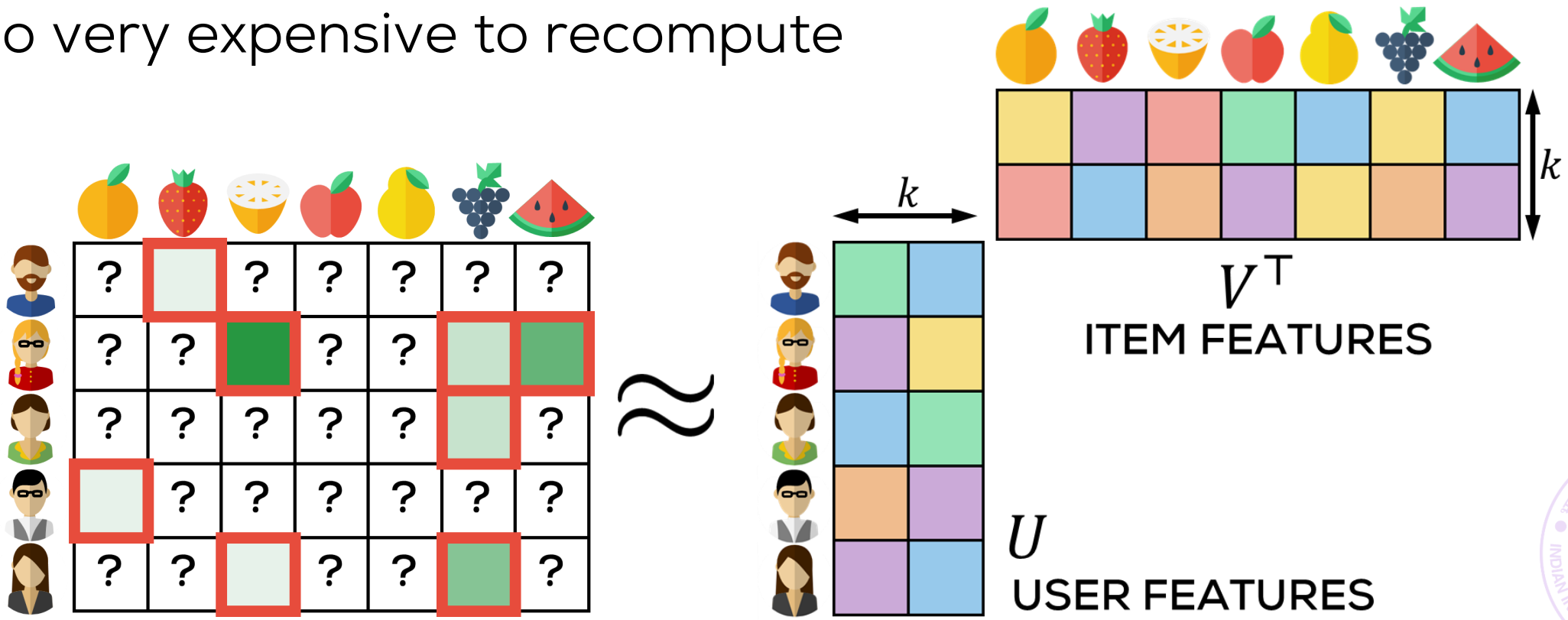
Recap

- Collaborative filtering uses low-rank matrix completion to fill up entries of a partially observed ratings matrix
- Exploits behavioural data but cannot utilize user/item features



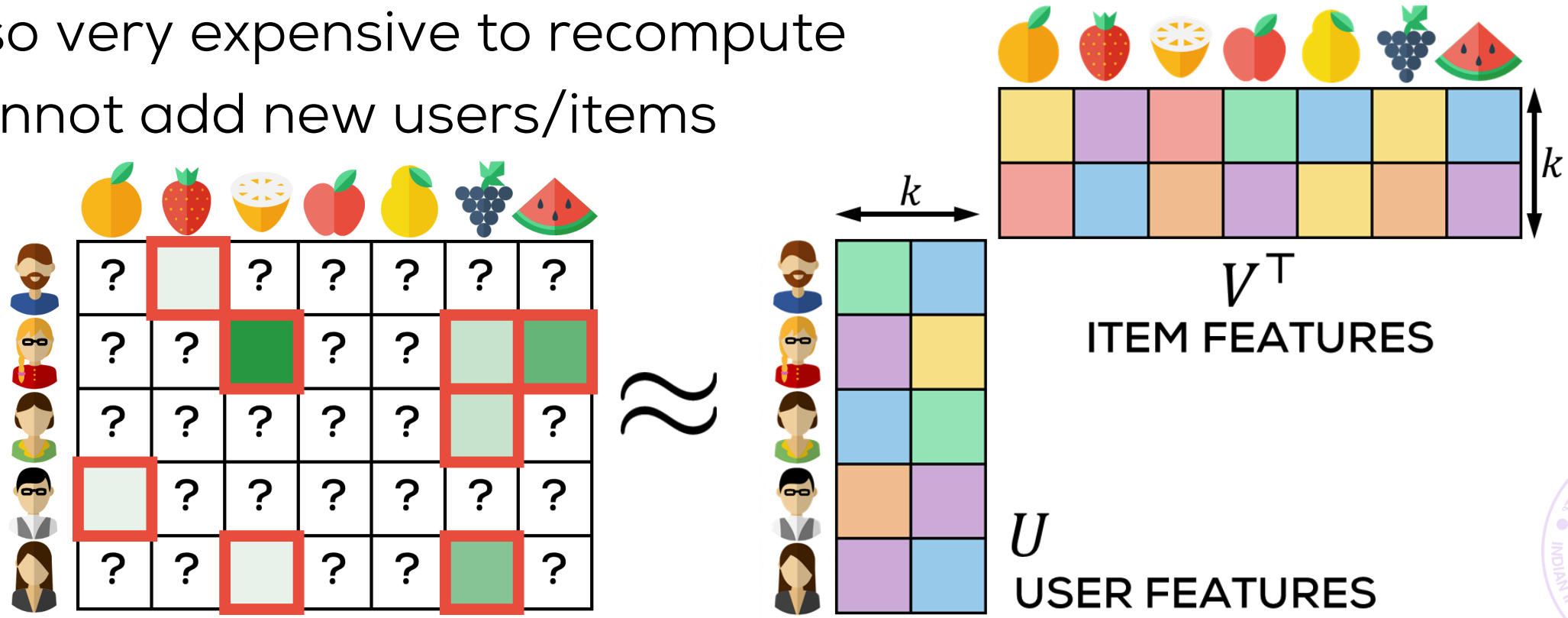
Recap

- Collaborative filtering uses low-rank matrix completion to fill up entries of a partially observed ratings matrix
- Exploits behavioural data but cannot utilize user/item features
- Also very expensive to recompute



Recap

- Collaborative filtering uses low-rank matrix completion to fill up entries of a partially observed ratings matrix
- Exploits behavioural data but cannot utilize user/item features
- Also very expensive to recompute
- Cannot add new users/items



Extreme Classification

The Extreme Classification Philosophy

- Treat recommendation as a massive multi-label learning problem
- Millions of labels and only 5-10 of them are switched on!
- Needle-in-a-haystack-on-steroids
- One of the leading methods for ranking and recommendation
- In use in search engines (Google, Bing) as well as YouTube
- Audacious gamble that turned out to be very successful
- Extremely simple but highly efficient methods
  - Embeddings + k-nearest neighbours
  - Decision trees and random forests

Extreme Classification

- Every user represented as a vector $\mathbf{x} \in \mathbb{R}^d$

- L items - every item $j \in [L]$ is a potential "label"
- Training data gives us $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1,\ldots,n}$ where $\mathbf{y}^i \in \{0,1\}^L$
- Can train a decision tree (see lecture 4) to solve the problem
- State-of-the-art FAST-XML uses a linear SVM on user feature $\mathbf{x}$ to split nodes and send users down to a leaf
- At a leaf, a new user is assigned all items liked by training users that reached that leaf

Extreme Classification

- Every user represented as a vector $\mathbf{x} \in \mathbb{R}^d$

- L items - every item $j \in [L]$ is a potential "label"
- Training data gives us $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1,\ldots,n}$ where $\mathbf{y}^i \in \{0,1\}^L$
- Can train a decision tree (see lecture 4) to solve the
- State-of-the-art FAST-XML uses a linear SVM on u to split nodes and send users down to a leaf
- At a leaf, a new user is assigned all items liked by training users that reached that leaf

+1: liked item
0: don't know

0 does not mean "not-like". It means unrated

Many other approaches – see project page

Pros and Cons

Pros

- One of the state-of-the-art methods in recommendation
- Very scalable – can handle 10^7 items and 10^6 users
- Can add new users very easily

Cons

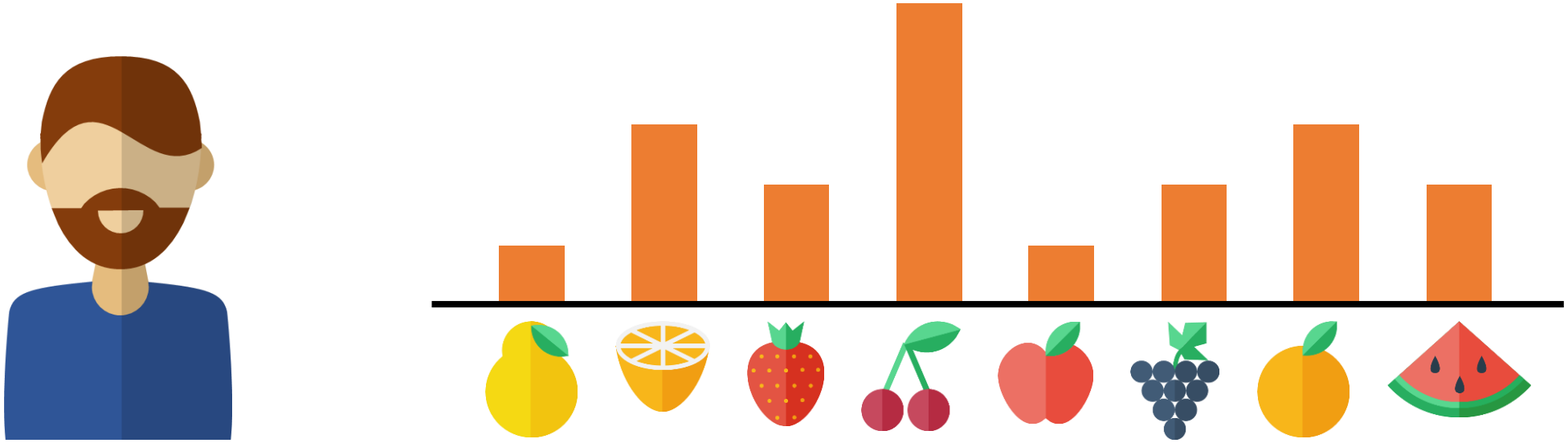
- No explicit collaboration - does not infer similarity of items/users
- Rare items (which very few users like) suffer
- Static – need to keep relearning the classifier again and again
- Not easy to add new items

Multi-armed Bandits

The Need for Exploration

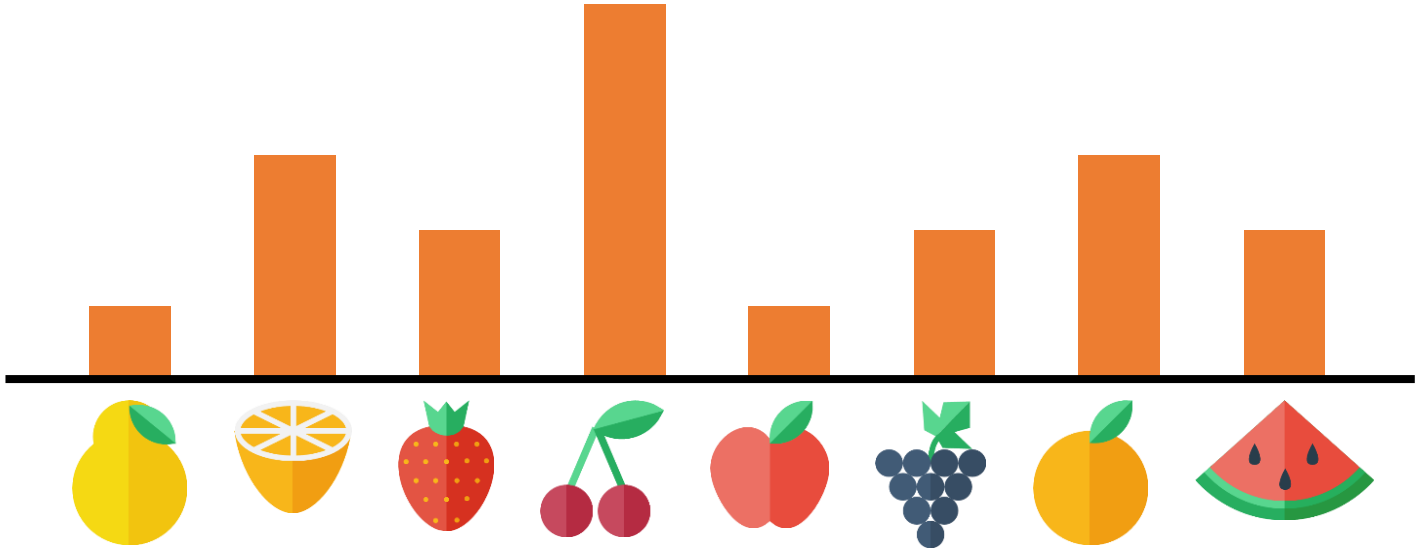
- Collaborative Filtering and Extreme Classification are static
- They may learn a good set of items for a user …
- … but will keep making the same recommendation no matter how many times the user visits the website
- Can make users feel bored – content seems stale
- Also missed opportunity to suggest something new and fresh that even the user did not know s/he liked!
- No exploration – usually a small set of items get recommended to everybody - other items get starved and those sellers may leave
- Bandit algorithms remedy this by using explore-exploit techniques
- Too advanced for CS771 – will only look at a simple version

The Multi-armed Bandit



- Assume we have just one user (to handle multiple users we need something called *contextual bandits* – covered in CS773)
- Each of the L items $j \in [L]$ has an associated probability p_j
- If recommended item j the user will buy it with probability p_j
- We want to maximize total number of purchases (can maximize total revenue instead by an easy extension)

The Multi-armed Bandit



- If we knew the purchase probabilities p_j then we would just keep recommending the item $j^* = \arg\max p_j$ again and again
- Can think of the L items as corresponding to L coins with biases p_j
- So what is a the big deal? Lecture 5 taught me how to estimate the bias of a coin. Can't I use that method on each item?
- Yes, you can but lets see what happens

The Pure Exploration Algorithm

PURE EXPLORATION

1. One user, L items
2. For $j = 1, \ldots L$
   1. For $t = 1, \ldots, T_0$
      1. Recommend item j to user
      2. Let $X_{jt} = \begin{cases} 1; \text{ if user purchases} \\ 0; \text{otherwise} \end{cases}$
   2. Let $\hat{p}_j = \frac{1}{T_0} \sum_{t=1}^{T_0} X_{jt}$

MLE estimate

Pros

- Simple algorithm
- Gives us idea of every item, not just j^*

Cons

- T_0 needs to be large for accurate estimation
- What if $L = 10^6$?
- Too many opportunities wasted in figuring out that some p_j is small

Optimism in the Face of Uncertainty

- Main drawback of PureExp is that it keeps recommending an item even if after, say $\frac{T_0}{2}$ recommendations, that item is not purchased
- Need to ditch an item if it appears to be disliked by the user
- Need to be careful though since the user may decide not to purchase even j^* purely by chance – can't be too hasty
- **Optimism in the Face of Uncertainty** – very powerful principle
- Suppose we recommended item j to the user a total of T_j times so far and it was purchased a $\hat{p}_j$ fraction of times.
- If T_j is large then p_j cannot be much larger than $\hat{p}_j$
- However, if T_j is small then it is possible that p_j much larger than $\hat{p}_j$
- What to do? Give item j some form of benefit of doubt

Optimism in the Face of Uncertainty

- Main drawback of PureExp is that it keeps recommending an item even if after, say $\frac{T_0}{2}$ recommendations, that item is not purchased
- Need to ditch an item if it appears to be disliked by the user
- Need to be careful though since the user may decide not to purchase even j^* purely by chance – can't be too hasty
- **Optimism in the Face of Uncertainty** – ver
- Suppose we recommended item j to the u so far and it was purchased a $\hat{p}_j$ fraction of ti
- If T_j is large then p_j cannot be much larger tha
- However, if T_j is small then it is possible that p_j p_j
- What to do? Give item j some form of benefit of doubt

Theoretical results like Chernoff-Hoeffding inequality ensure this

Optimism in the face of uncertainty ☺

The UCB Algorithm

UPPER CONFIDENCE BOUND

1. One user, L items
2. For $t = 1,2,3, \ldots,$
   1. Let $j_t \leftarrow \arg\max_j \hat{p}_j + \sqrt{1/T_j}$
   2. Recommend item j_t to user
   3. Let $n_{j_t} = \begin{cases} n_{j_t} + 1; \text{if user buys} \\ n_{j_t}; \text{otherwise} \end{cases}$
   4. Update $T_{j_t} \leftarrow T_{j_t} + 1$
   5. Update $\hat{p}_{j_t} \leftarrow n_{j_t}/T_{j_t}$

- Benefit of doubt is $1/\sqrt{T_j}$
- If an item is starved, T_j will be small and large benefit
- Starved items will get recommended eventually
- However, if $T_{j'}$ is large for item j', then tiny benefit
- Item j' recommended only if $\hat{p}_{j'}$ is large
- UCB gives every item a fighting chance but ditches losers very quickly

The UCB Algorithm

For each item j,
T_j: number of times j was recommended
n_j: number of items j was purchased

UPPER CONFIDENCE BOUND

1. One user, L items
2. For $t = 1,2,3, \ldots,$
   1. Let $j_t \leftarrow \arg\max_j \hat{p}_j + \sqrt{1/T_j}$
   2. Recommend item j_t to user
   3. Let $n_{j_t} = \begin{cases} n_{j_t} + 1; \text{if user buys} \\ n_{j_t}; \text{otherwise} \end{cases}$
   4. Update $T_{j_t} \leftarrow T_{j_t} + 1$
   5. Update $\hat{p}_{j_t} \leftarrow n_{j_t}/T_{j_t}$

- Be… $\sqrt{T_j}$
- If … j will … nefit
- Starved items will get recommended eventually
- However, if $T_{j'}$ is large for item j', then tiny benefit
- Item j' recommended only if $\hat{p}_{j'}$ is large
- UCB gives every item a fighting chance but ditches losers very quickly

Initially, all items recommended at least once to avoid divide-by-zero errors

Explore-exploit Strategy

- Notice the tug-of-war played to choose the item to recommend

$$j_t \leftarrow \arg\max_j \hat{p}_j + \sqrt{1/T_j}$$

- Can be see as a sum of an exploitation and an exploration term

$$j_t \leftarrow \arg\max_j \text{EXPLOIT}(j) + \text{EXPLORE}(j)$$

- EXPLOIT encourages choosing items that have been bought often
- EXPLORE encourages choosing items not tried out before
- Can be creative in designing exploitation and exploration terms
- There exists a "Bayesian" version of this called Thompson Sampling
- It maintains a Gaussian posterior for p_j of the form $\mathcal{N}(\hat{p}_j, 1/T_j)$

$$j_t^{TS} \leftarrow \arg\max_j \tilde{p}_j \text{ where } \tilde{p}_j \sim \mathcal{N}(\hat{p}_j, 1/T_j)$$

Pros and Cons

Pros

- Widely used as a top level method in recommendation
- Online and not static, learns continuously
- Does not get stale very easily – explores!
- Can utilize user and item features easily (contextual bandits)
- Can add new users and new items very easily

Cons

- No explicit collaboration – does not infer similarity of items/users
- Need powerful contextual models to handle large item sets
- Algorithms more complicated, expensive in contextual bandits

Concluding Thoughts

- In practice, hybrid methods popular
- Recent advances in getting best of all many methodologies
- Matrix factorization where rows and columns have side features
- Collaborative multi-armed bandits – identifies similar items
- Very active area … published literature only way to stay fresh
- Btw … most commercial recommendation systems are massive ensembles with 100s of algorithms running in tandem. Exact details usually a closely guarded secret
- See project page for resources on extreme multi-label classfn
- See course webpage for CS773 for more on bandits

Please give your Feedback

http://tinyurl.com/ml17-18afb