

RecSys

CS771: Introduction to Machine Learning

Purushottam Kar

Precap

2

Recommendation Systems are ubiquitous in various fields

Social networks, e-commerce but also medicine, governance, services

RecSys face severe data imbalance as well as gigantic datasets

Extremely lucrative for monetization

Very active area – RecSys, WSDM, ICML, NeurIPS, KDD, WWW

Will look at two approaches

Collaborative Filtering via Matrix Completion

Extreme Multilabel Classification

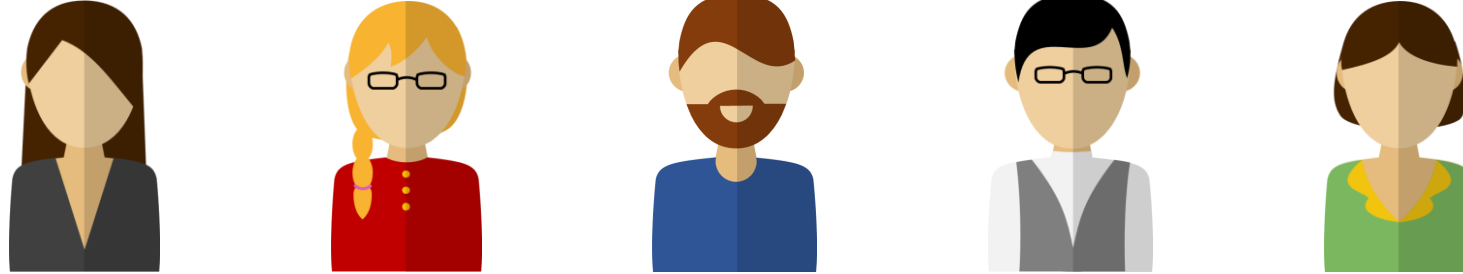
Will also look at how to perform online recommendations



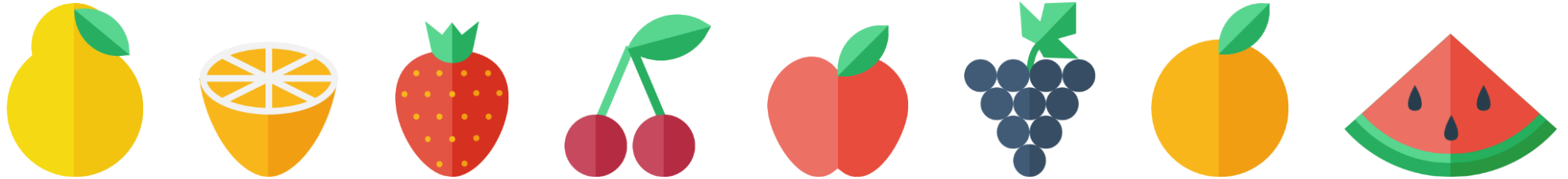
Abstract Problem Statement

3

There are users ...



... and there are items ...



Users come to us one at a time in no particular order

... and we need to recommend, to each user, items they will “like”

User tastes expected to be stable but may vary slowly across time



Applications of RecSys

4

User

sundarbans.com user

reddit.com user

Student

Patient

Search query

Advertiser

Items

sundarbans.com products

Reddit posts

Study material

Medicines

Advertisements

Search query to bid on

- “Users” are in millions and thousands come each second
- “Items” are in millions and each user likes only 5-10 items
- Some items are popular but most items liked by only 5-10 users



The Matrix Completion Problem

5

m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

Some users rate some items

Users rate very rarely

Get to see those entries of X

Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5

m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

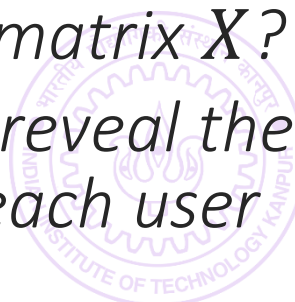
Some users rate some items

Users rate very rarely

Get to see those entries of X

Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5



m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

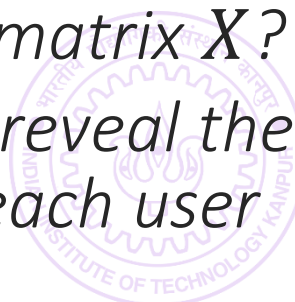
Some users rate some items

Users rate very rarely

Get to see those entries of X

Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5



m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

Some users rate some items

Users rate very rarely

Get to see those entries of X

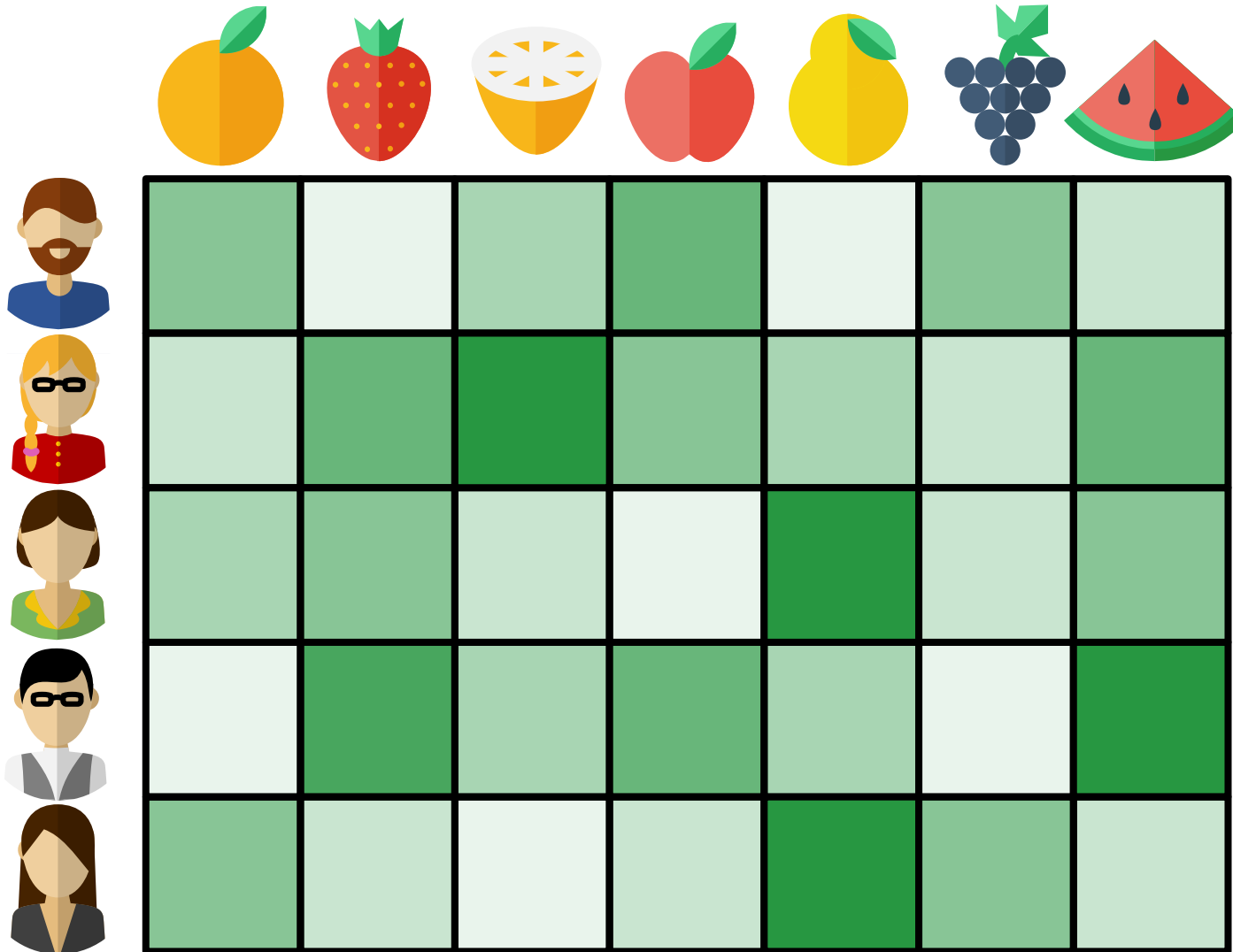
Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5



m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

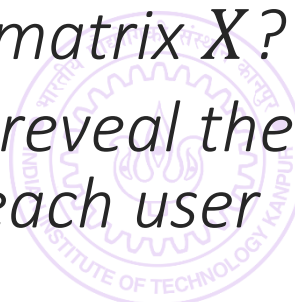
Some users rate some items

Users rate very rarely

Get to see those entries of X

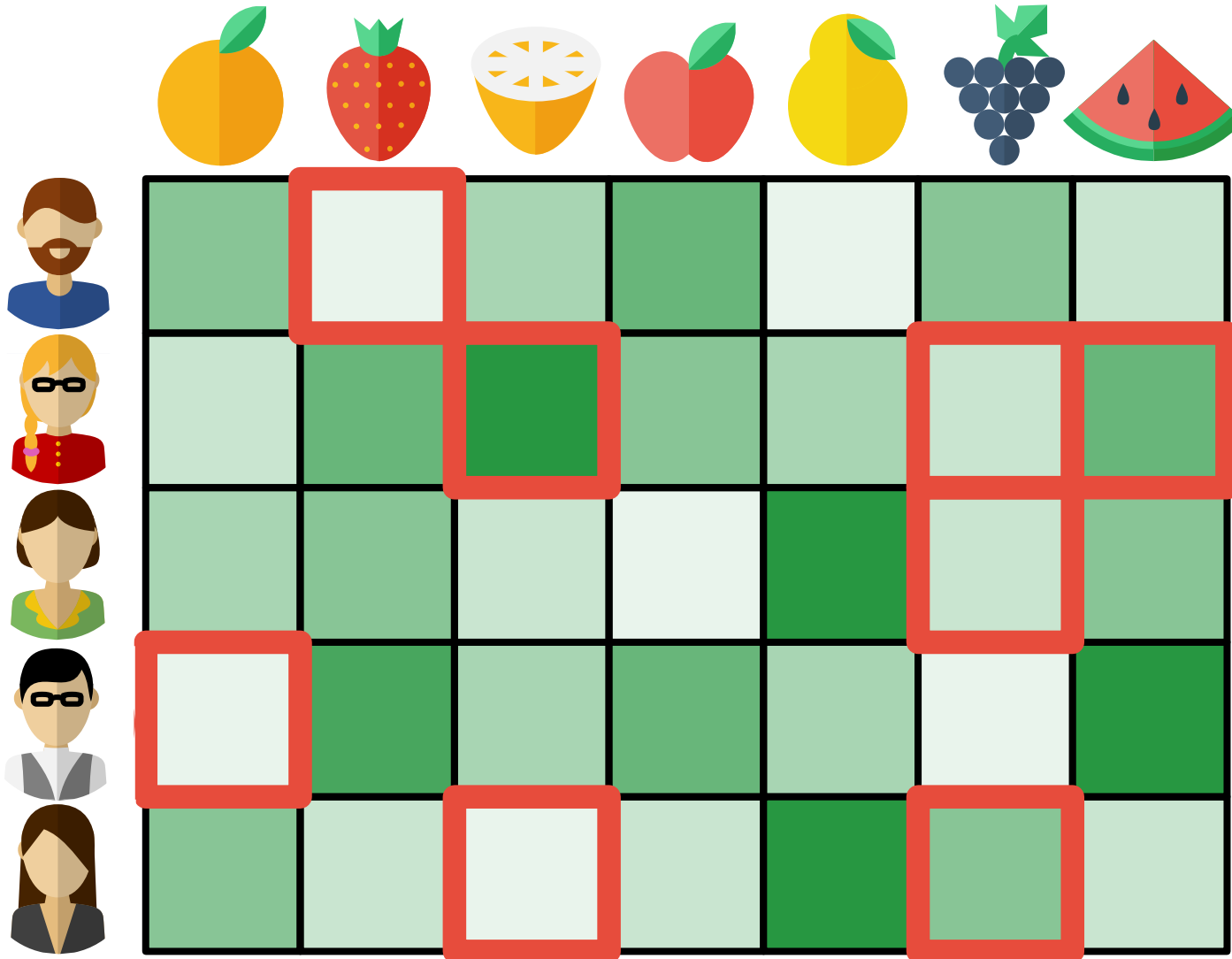
Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5



m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

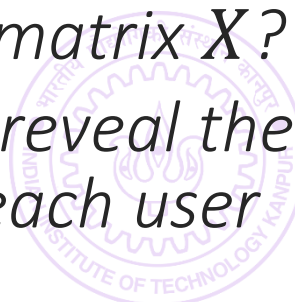
Some users rate some items

Users rate very rarely

Get to see those entries of X

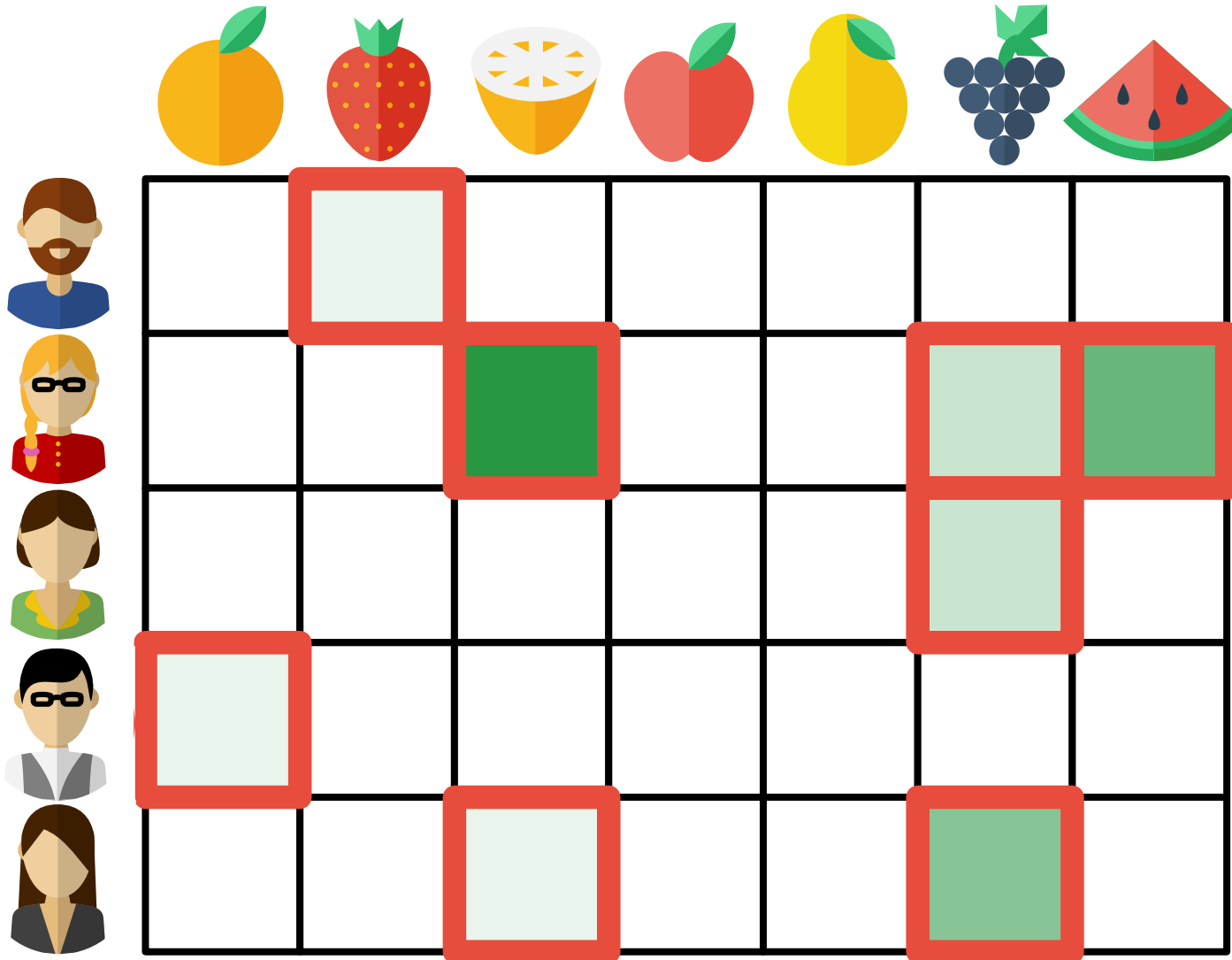
Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5



m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

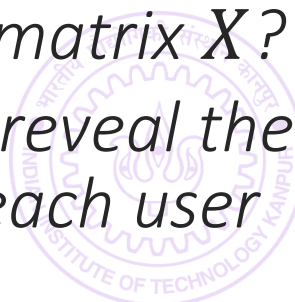
Some users rate some items

Users rate very rarely

Get to see those entries of X

Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix Completion Problem

5

	?		?	?	?	?	?
	?	?		?	?		
	?	?	?	?	?		?
		?	?	?	?	?	?
	?	?		?	?		?

m users n items

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

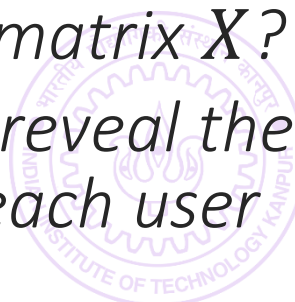
Some users rate some items

Users rate very rarely

Get to see those entries of X


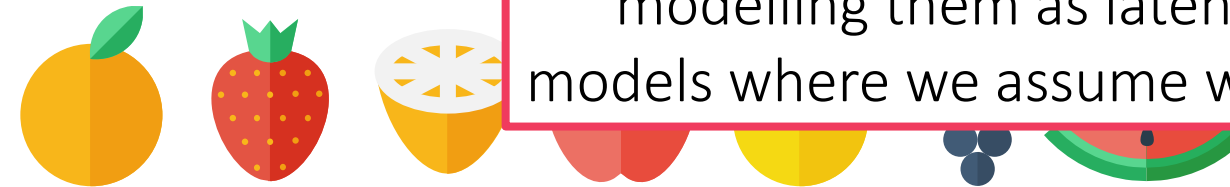
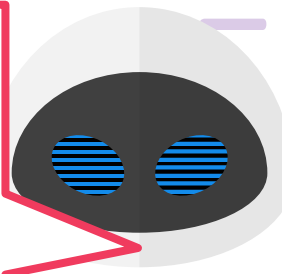
Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



The Matrix

We do not assume we have access to any user/item features in this model (instead we will learn them by modelling them as latent variables). There are other models where we assume we are given user/item features.



?		?	?	?	?	?
?	?		?	?		
?	?	?	?	?		?
	?	?	?	?	?	?
?	?		?	?		?

Rating matrix $X \in \mathbb{R}^{m \times n}$

X_{ij} : rating indicating how much user i likes item j

Deeper shade \Rightarrow more like

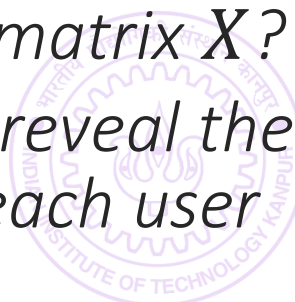
Some users rate some items

Users rate very rarely

Get to see those entries of X

Can we recover the unseen entries i.e. complete matrix X ?

Completing X would reveal the most liked items for each user



Low-rank Matrix Completion

14

Let us make some assumptions about the ratings

If two users u_1, u_2 are “similar”, they would rate all items similarly

If u_1 likes an item, u_2 would also like it, if u_1 dislikes some item, u_2 would also dislike it

If two items v_1, v_2 are “similar”, all users would rate them similarly

If a user likes v_1 , they like v_2 as well, if some user dislikes v_1 , they would also dislike v_2

Some such assumption is necessary to solve the problem

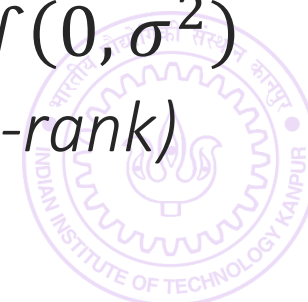
No other reason why observed entries should tell us anything about unobserved entries

Popular assumption – latent user and item features and linear model

Assume for every user u_i , exists a vector $\mathbf{u}^i \in \mathbb{R}^k$ and for every item v_j , exists a vector $\mathbf{v}^j \in \mathbb{R}^k$ such that rating $r_{ij} = \langle \mathbf{u}^i, \mathbf{v}^j \rangle + \epsilon_{ij}$ where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$

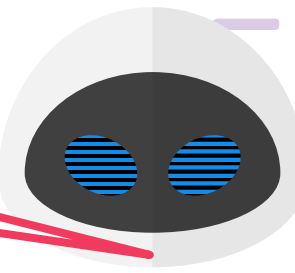
Equivalent to saying that the ratings matrix is almost rank k (i.e. low-rank)

Let $U = [\mathbf{u}^1 \dots \mathbf{u}^m]^\top$, $V = [\mathbf{v}^1 \dots \mathbf{v}^n]^\top$ then $X = UV^\top + E$



Low-rank Matrix Com

k is a hyperparameter in this model



Let us make some assumptions about the ratings

If two users u_1, u_2 are “similar”, they would rate all items similarly

If u_1 likes an item, u_2 would also like it, if u_1 dislikes some item, u_2 would also dislike it

If two items v_1, v_2 are “similar”, all users would rate them similarly

If a user likes v_1 , they like v_2 as well, if some user dislikes v_1 , they would also dislike v_2

Some such assumption is necessary to solve the problem

No other reason why observed entries should tell us anything about unobserved entries

Popular assumption – latent user and item features and linear model

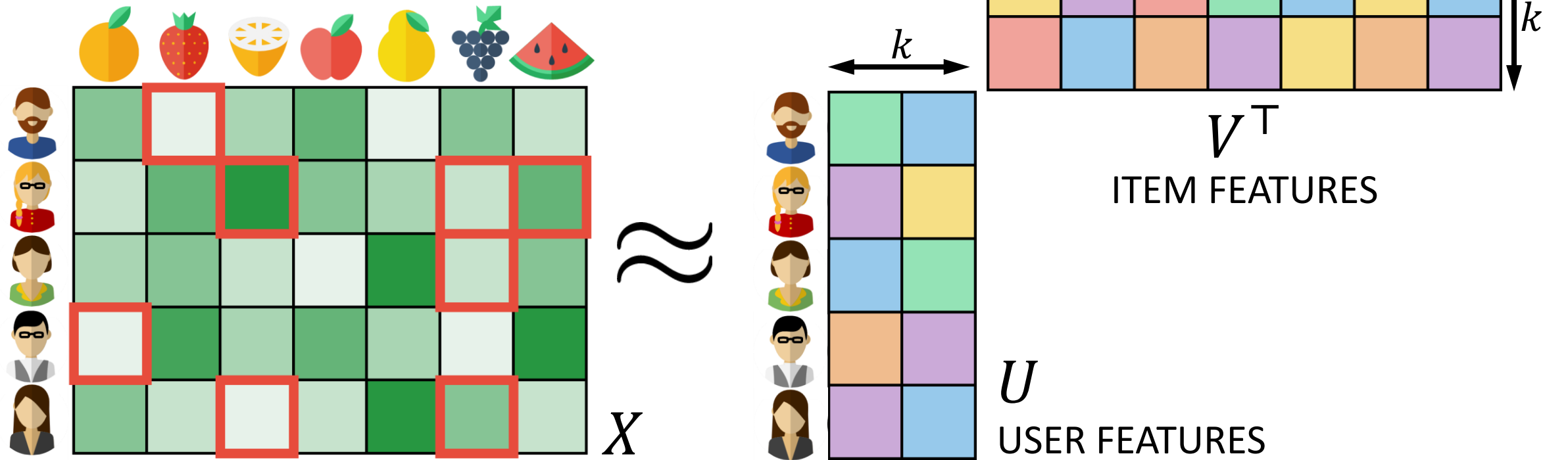
Assume for every user u_i , exists a vector $\mathbf{u}^i \in \mathbb{R}^k$ and for every item v_j , exists a vector $\mathbf{v}^j \in \mathbb{R}^k$ such that rating $r_{ij} = \langle \mathbf{u}^i, \mathbf{v}^j \rangle + \epsilon_{ij}$ where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$

Equivalent to saying that the ratings matrix is almost rank k (i.e. low-rank)

Let $U = [\mathbf{u}^1 \dots \mathbf{u}^m]^\top$, $V = [\mathbf{v}^1 \dots \mathbf{v}^n]^\top$ then $X = UV^\top + E$



Low-rank Matrix Completion



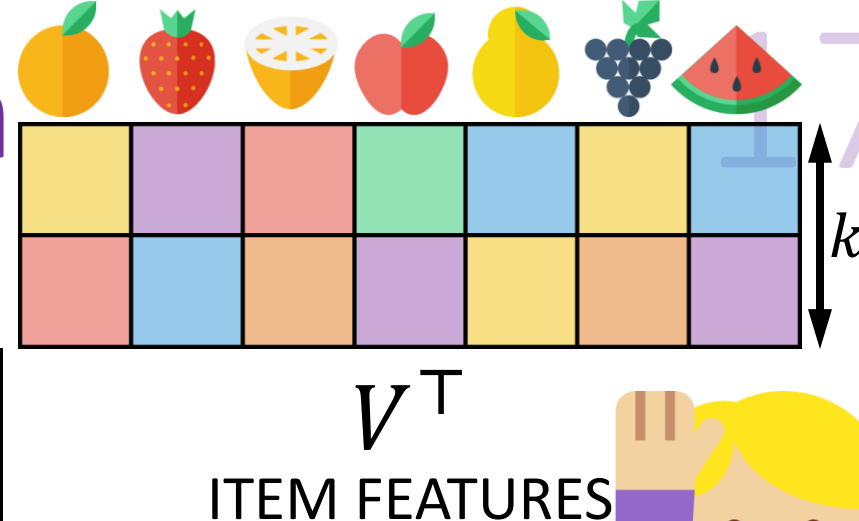
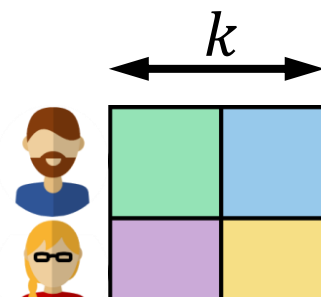
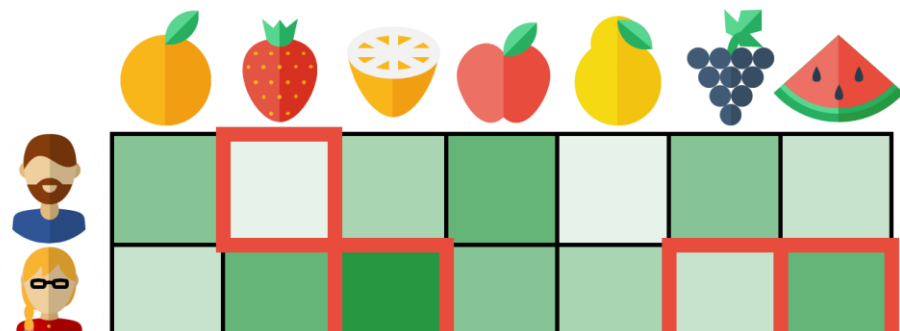
$$X_{ij} \approx \langle \mathbf{u}^i, \mathbf{v}^j \rangle \text{ where } U = [\mathbf{u}^1, \dots, \mathbf{u}^m]^T \text{ and } V = [\mathbf{v}^1, \dots, \mathbf{v}^n]^T$$

As a side effect, vector representations of all users and items

If two users rate similarly, then the method will learn similar “features” for them!
 These “features” can be useful in making future recos but can also violate privacy if they reveal details users didn’t tell us e.g. if one of the features is correlated with age



Low-rank Matrix Completion



Suppose $\Omega \subset [m] \times [n]$ are the locations observed by us. The MAP problem (with Gaussian priors on $\mathbf{u}^i, \mathbf{v}^j$) reduces to the following optimization problem

$$\arg \min_{\substack{U \in \mathbb{R}^{m \times k} \\ V \in \mathbb{R}^{n \times k}}} \sum_{(i,j) \in \Omega} (X_{ij} - \langle \mathbf{u}^i, \mathbf{v}^j \rangle)^2 + \lambda \cdot \left(\sum_{i=1}^m \|\mathbf{u}^i\|_2^2 + \sum_{j=1}^n \|\mathbf{v}^j\|_2^2 \right)$$

$X_{ij} \approx \langle \mathbf{u}^i, \mathbf{v}^j \rangle$ where $U = [\mathbf{u}^1 \dots \mathbf{u}^m]^T$ and $V = [\mathbf{v}^1 \dots \mathbf{v}^n]^T$

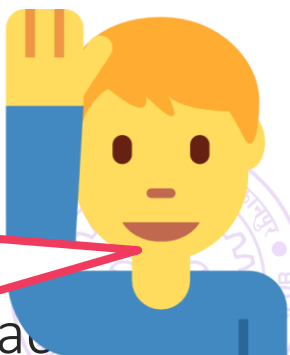
As a side effect,

If two users rate

These “features”

can be useful in making future recommendations but can also violate privacy by revealing details users didn't tell us e.g. if one of the features is correlated with age

Alternating minimization is currently one of the fastest ways to solve this problem since if we fix U , solving for V is just a bunch of ridge regression problems and vice versa!



ALTERNATING OPTIMIZATION

1. Observed locations Ω and values $\{X_{ij} : (i, j) \in \Omega\}$
2. Initialize $\mathbf{v}^{1,0}, \mathbf{v}^{2,0}, \dots, \mathbf{v}^{n,0}$
3. For $t = 1, 2, \dots$

1. Update the user vectors for $i = 1, \dots, m$

$$\mathbf{u}^i \leftarrow \left(\sum_{j:(i,j) \in \Omega} \mathbf{v}^j (\mathbf{v}^j)^\top + \lambda \cdot I \right)^{-1} \left(\sum_{j:(i,j) \in \Omega} X_{ij} \cdot \mathbf{v}^j \right)$$

2. Update the item vectors for $j = 1, \dots, n$

$$\mathbf{v}^j \leftarrow \left(\sum_{i:(i,j) \in \Omega} \mathbf{u}^i (\mathbf{u}^i)^\top + \lambda \cdot I \right)^{-1} \left(\sum_{i:(i,j) \in \Omega} X_{ij} \cdot \mathbf{u}^i \right)$$

4. Repeat until convergence



ALTERNATING OPTIMIZATION

1. Observed locations Ω and values $\{X_{ij}: (i, j) \in \Omega\}$
2. Initialize $\mathbf{v}^{1,0}, \mathbf{v}^{2,0}, \dots, \mathbf{v}^{n,0}$
3. For $t = 1, 2, \dots$

1. Update the user vectors for $i = 1, \dots, n$

$$\mathbf{u}^i \leftarrow \left(\sum_{j:(i,j) \in \Omega} \mathbf{v}^j (\mathbf{v}^j)^\top + \lambda \cdot I \right)^{-1} \left(\sum_{j:(i,j) \in \Omega} X_{ij} \cdot \mathbf{v}^j \right)$$

Note: sum only over items that user i rated (small for most users)

2. Update the item vectors for $j = 1, \dots, m$

$$\mathbf{v}^j \leftarrow \left(\sum_{i:(i,j) \in \Omega} \mathbf{u}^i (\mathbf{u}^i)^\top + \lambda \cdot I \right)^{-1} \left(\sum_{i:(i,j) \in \Omega} X_{ij} \cdot \mathbf{u}^i \right)$$

Note: sum only over users that rated item j (small for most items)

4. Repeat until convergence



Interesting Extensions

20

Can speed AltMin by using SGD to update user and item latent vectors

Would take only $\mathcal{O}(k)$ time per iteration

Suppose ratings are not real numbers but counts or Boolean

Can model ratings using a generalized linear models instead (see lec 25)

$$\mathbb{P}[r_{ij} | \mathbf{u}^i, \mathbf{v}^j] \propto \exp(r_{ij} \cdot \langle \mathbf{u}^i, \mathbf{v}^j \rangle - A(\langle \mathbf{u}^i, \mathbf{v}^j \rangle) - h(r_{ij}))$$

The model we studied becomes a special case where $\mathbb{P}[r_{ij} | \mathbf{u}^i, \mathbf{v}^j] = \mathcal{N}(\langle \mathbf{u}^i, \mathbf{v}^j \rangle, \sigma^2)$

Instead of the alternations being ridge regression problems, they would now become GLM problems e.g. logistic regression problems (Boolean ratings)

There also exist non-linear extensions to these

Dziugaite and Roy, Neural network matrix factorization, 2015

He et al, Neural collaborative Filtering, WWW 2017



Collaborative Filtering

21

Pros

- Relies purely on behavioural data
- Does not require users to submit personal information
- Does not require sellers to submit item information
- Collaborative method: lazy users benefit from active users

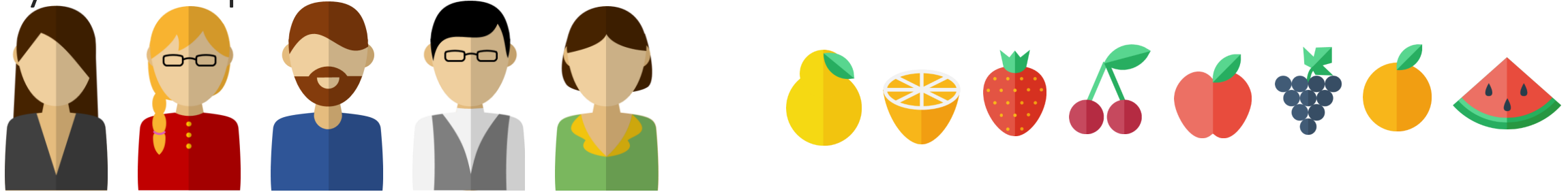
Cons

- Relies purely on behavioural data
- Cannot utilize user and item information even if present
- Expensive to recompute matrix factorization again and again
- Adding new users and items not straightforward (cold start problem)



Recommendation as Multi-label Learning 22

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 $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1, \dots, n}$ where $\mathbf{y}^i \in \{0, 1\}^L$

Learn DT/LwP/deep models to predict \mathbf{y}^i using \mathbf{x}^i

Recall: often, available info is one sided i.e. $\mathbf{y}_j^i = 1$ means user i does like item j . However, users mostly do not tell us anything about items that they do not like i.e. $\mathbf{y}_k^i = 0$ does not necessarily mean means user i doesn't like item k

Also influences what performance measures can be used. Mostly “precision” based performance measures used since recall based performance measures would require us to know the entire set of items a user likes which is unreasonable to expect



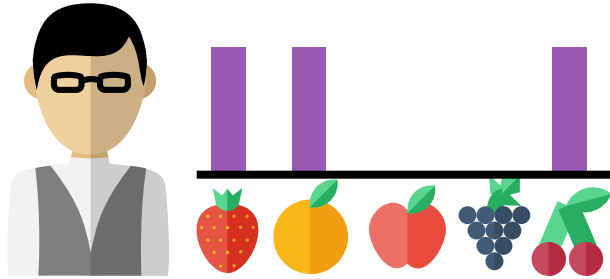
Recommendation using kNN

23



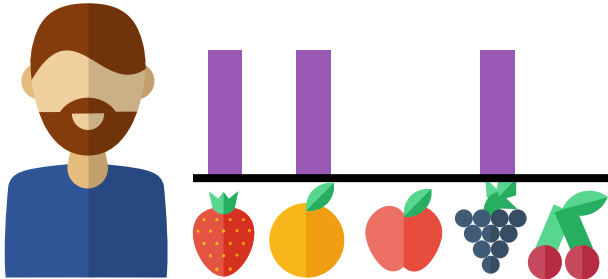
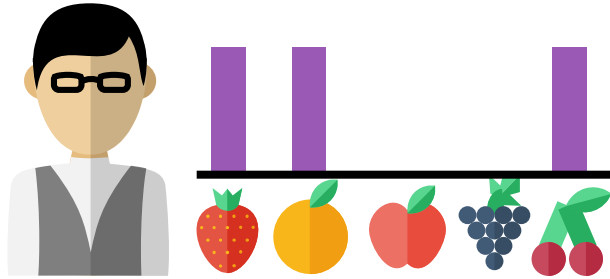
Recommendation using kNN

23



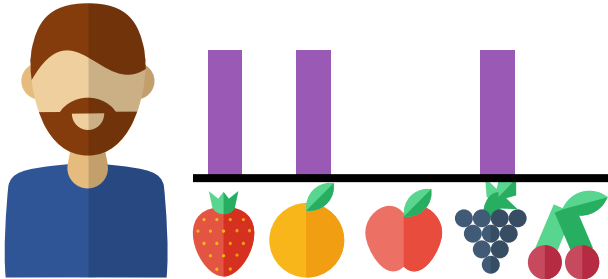
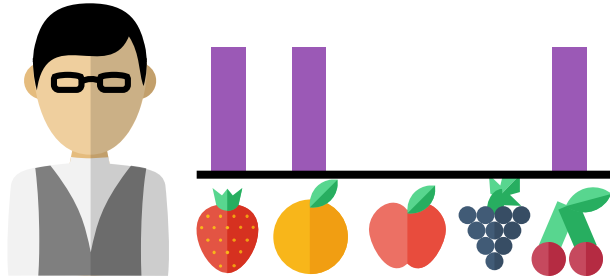
Recommendation using kNN

23



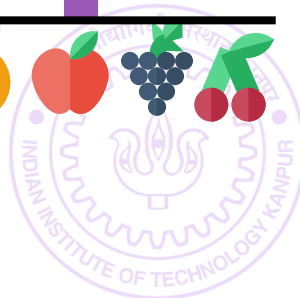
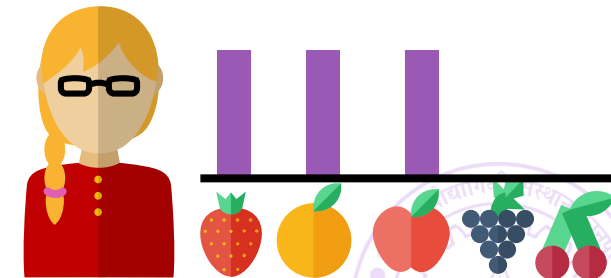
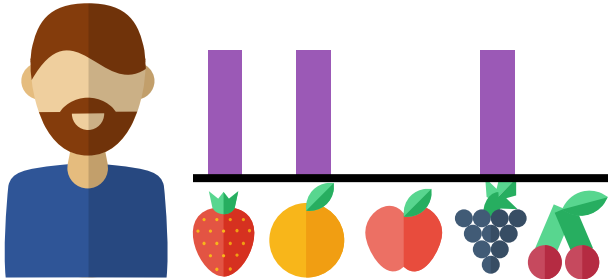
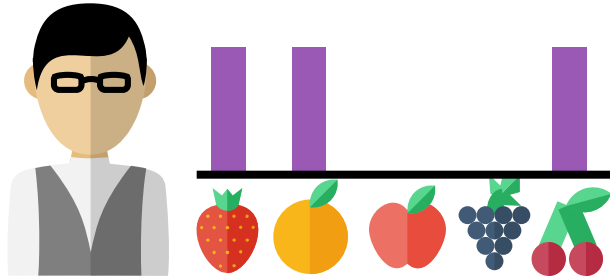
Recommendation using kNN

23



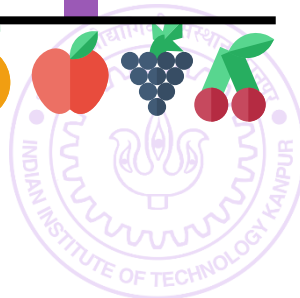
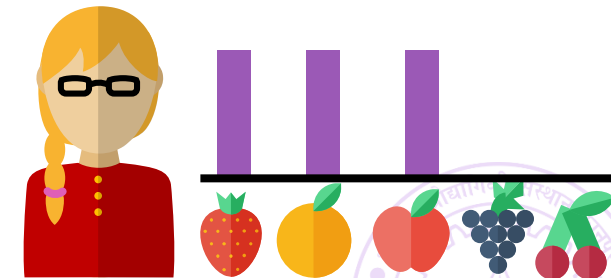
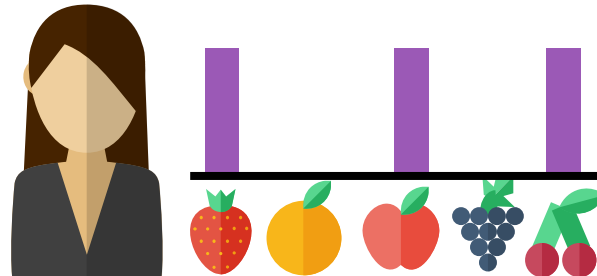
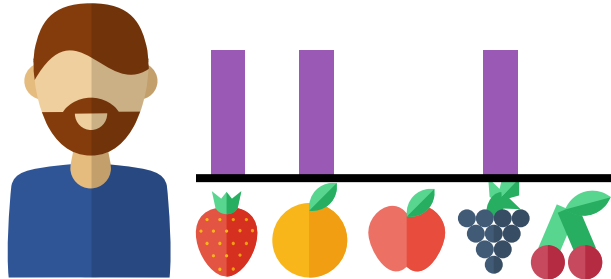
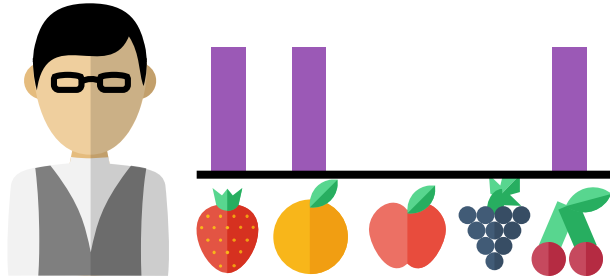
Recommendation using kNN

23



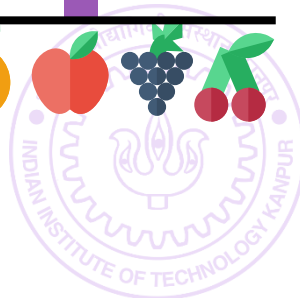
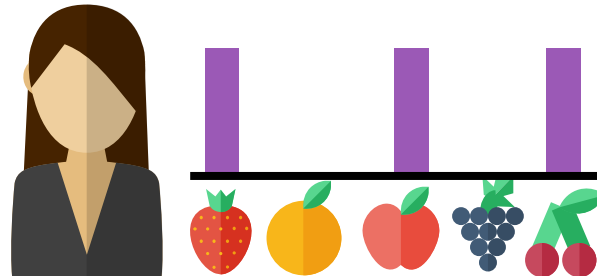
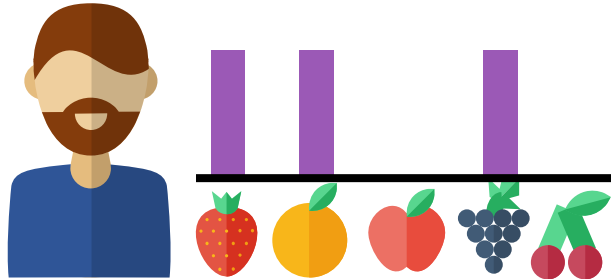
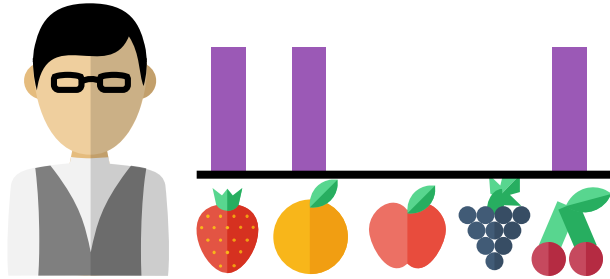
Recommendation using kNN

23



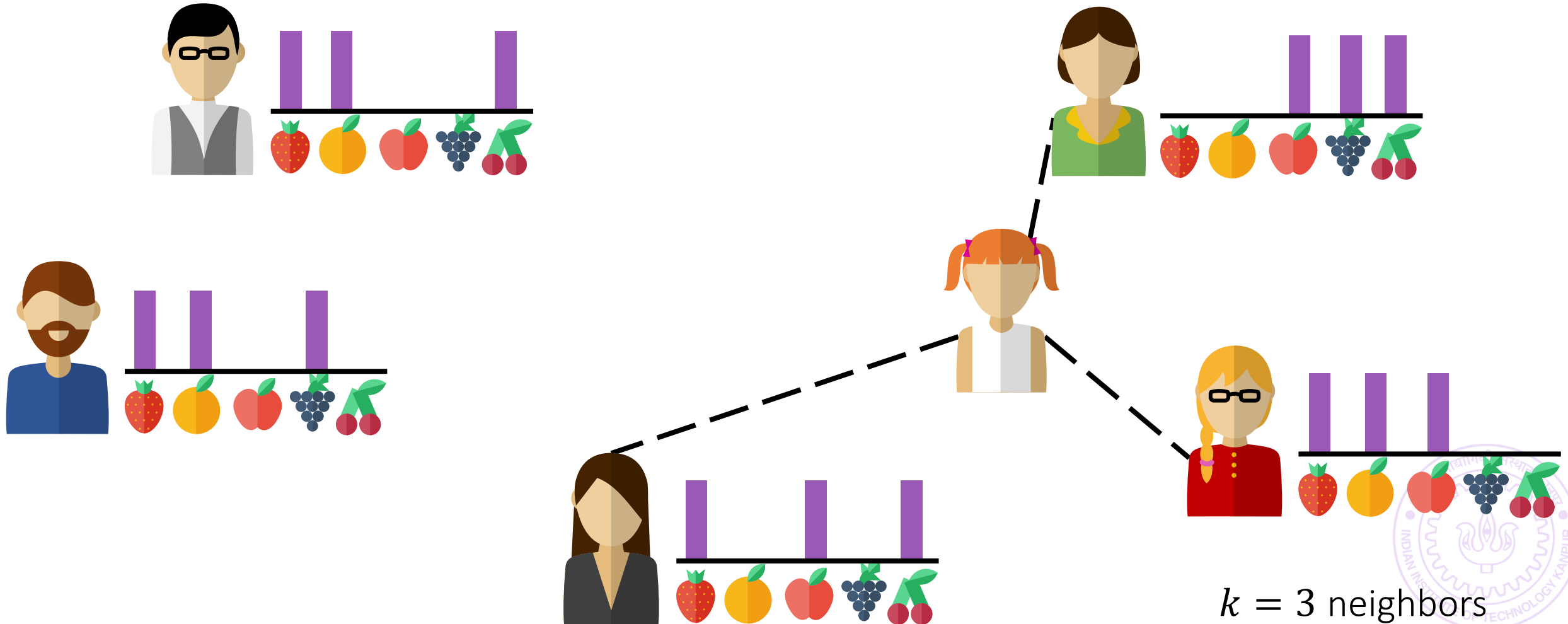
Recommendation using kNN

23



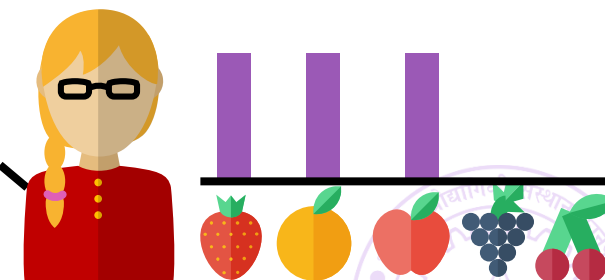
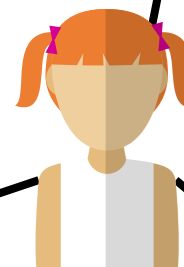
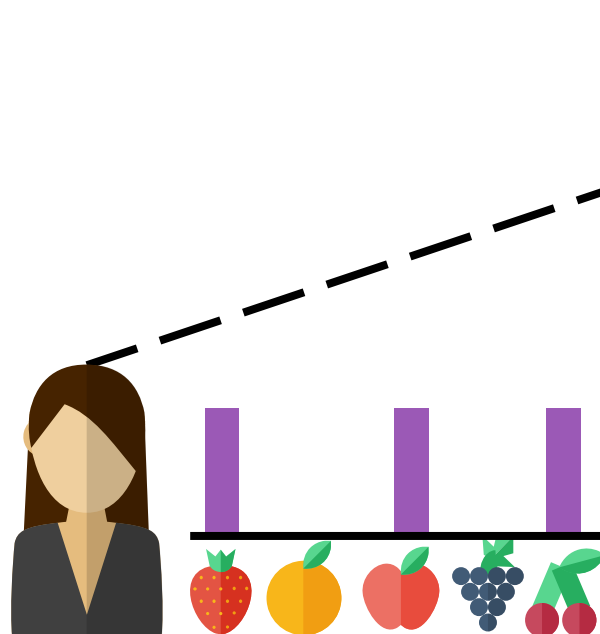
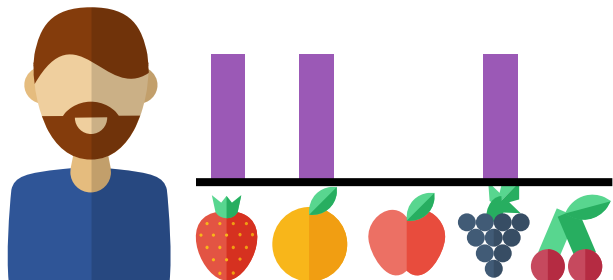
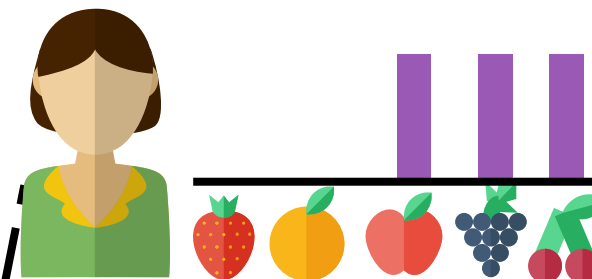
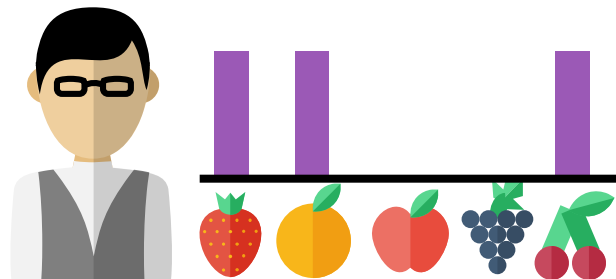
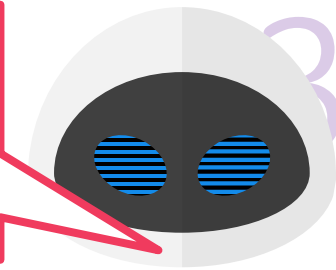
Recommendation using kNN

23

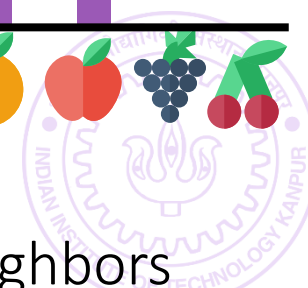


R

Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels

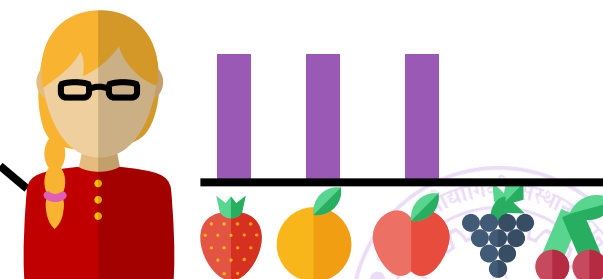
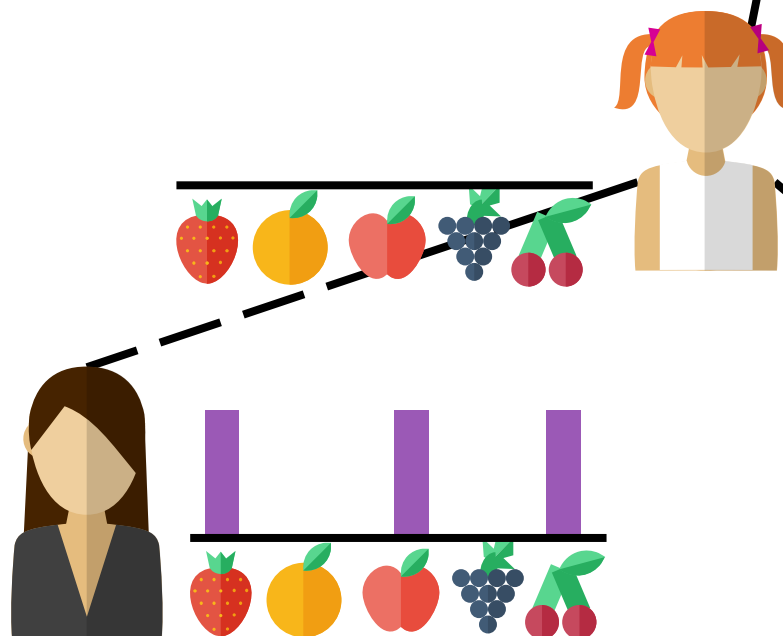
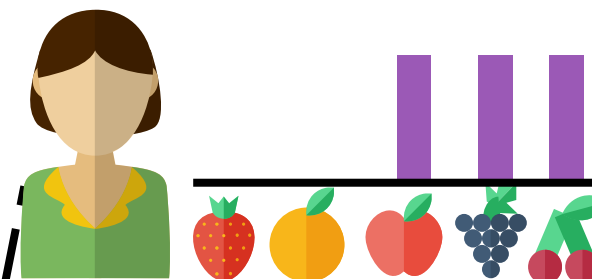
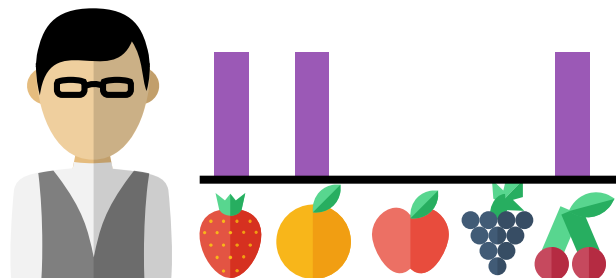
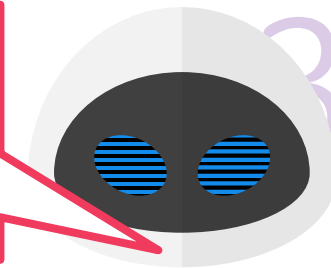


$k = 3$ neighbors

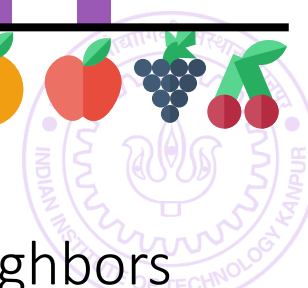


R

Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels

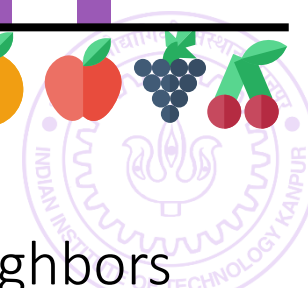
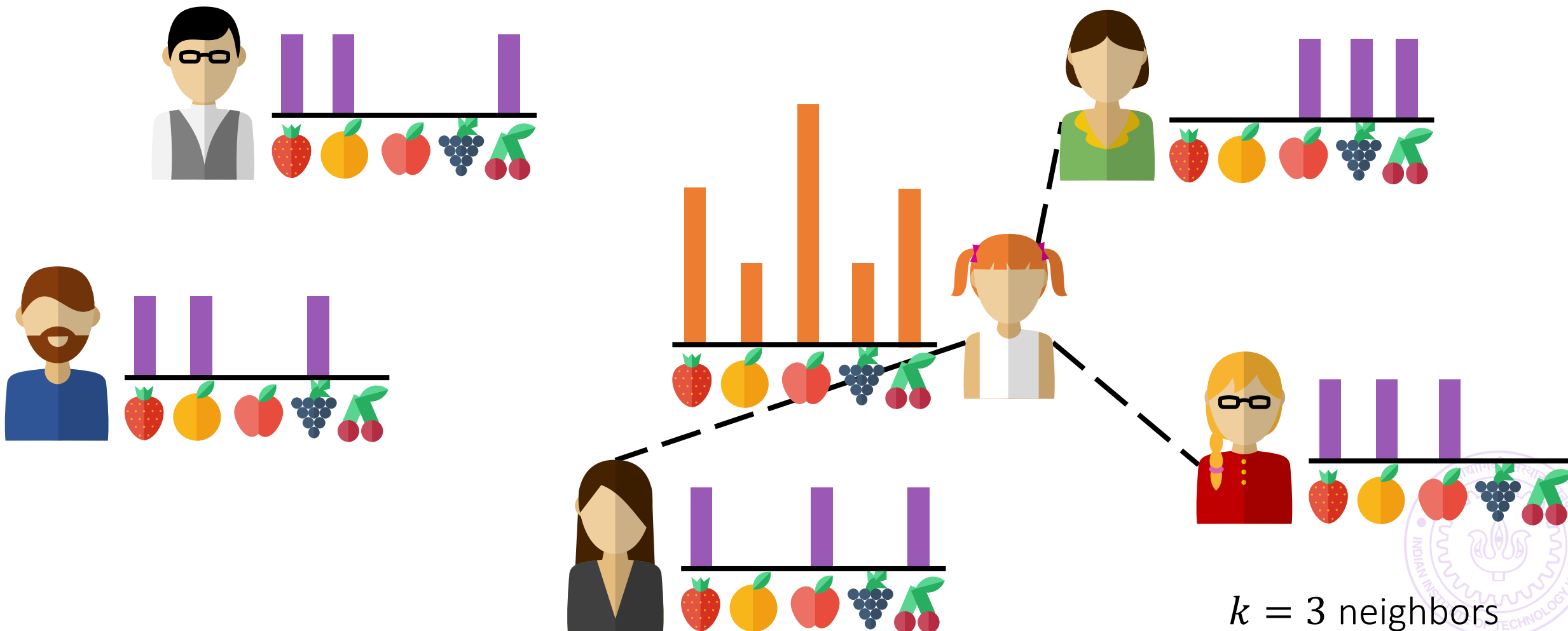
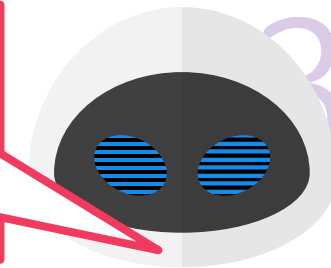


$k = 3$ neighbors



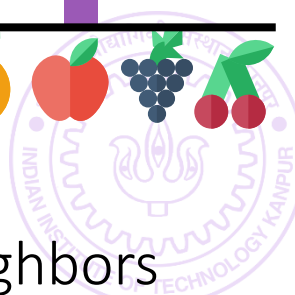
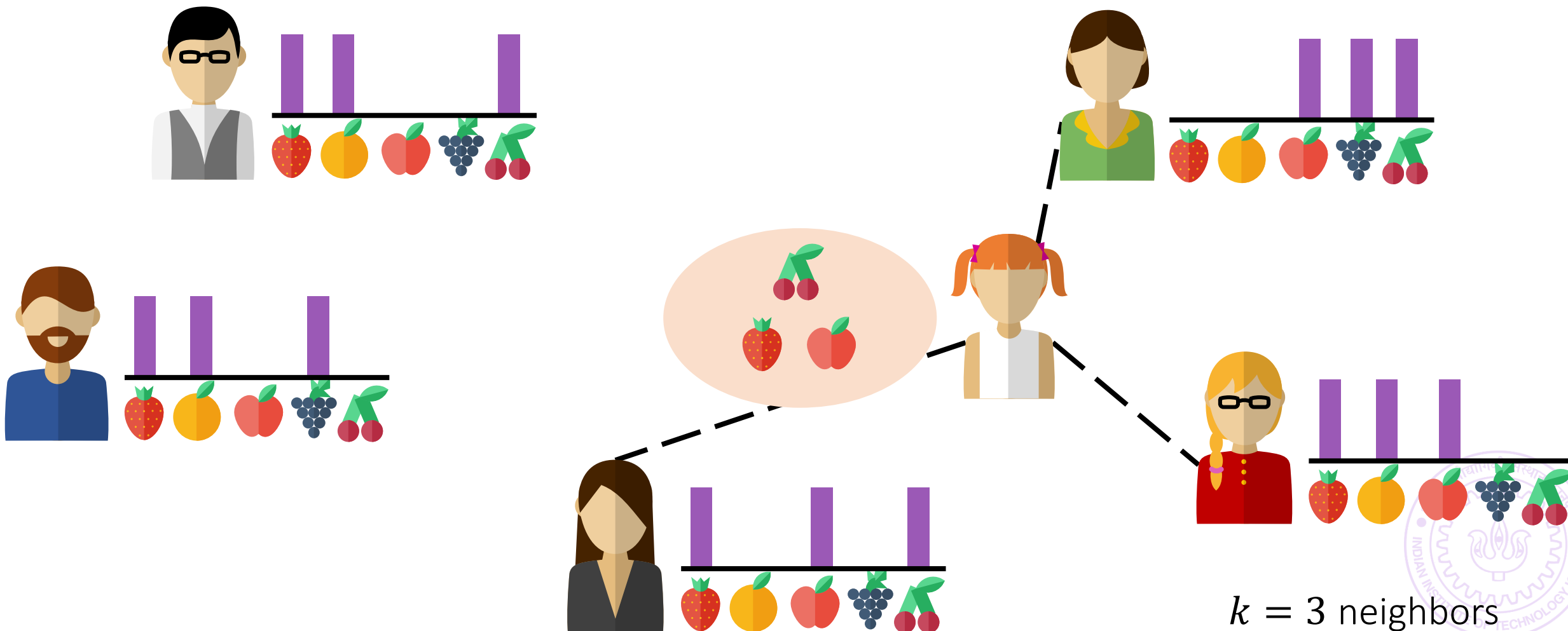
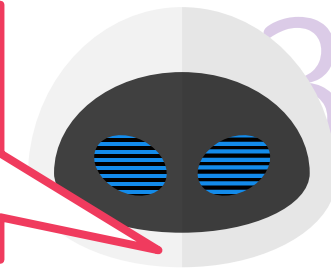
R

Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels



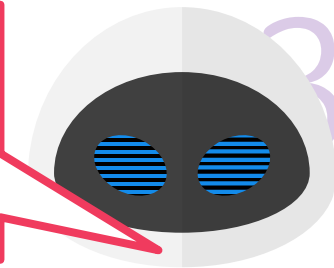
R

Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels

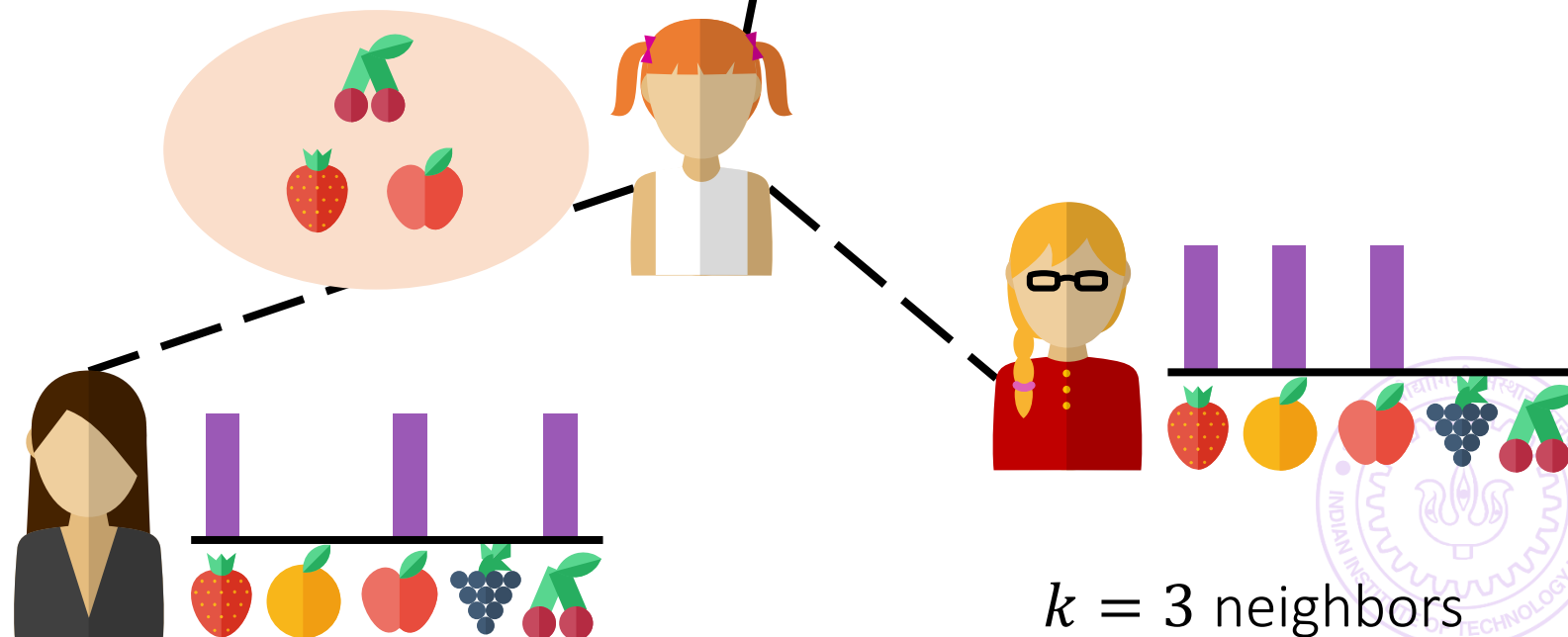
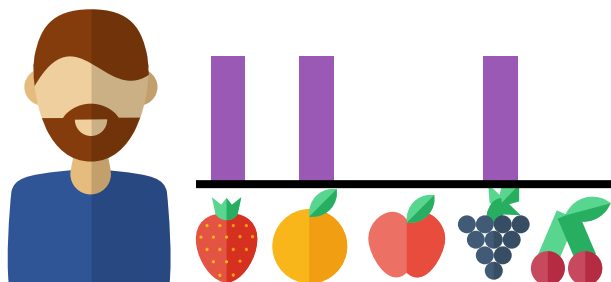


R

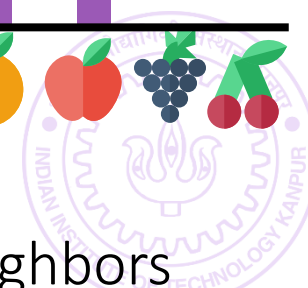
Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels



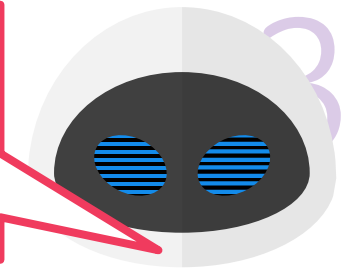
May similarly use LwP methods for recommendation too. Average the feature vectors of all users who like a label as the “prototype” of that label – total of L prototypes. At test time, recommend whichever labels are such that their prototypes are close to the test feature vector



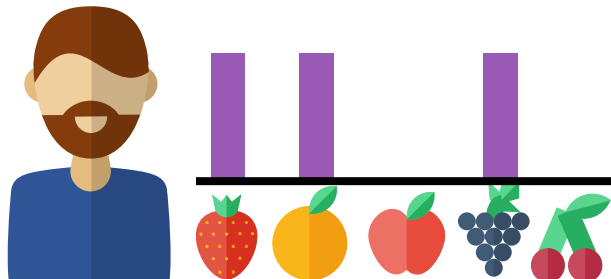
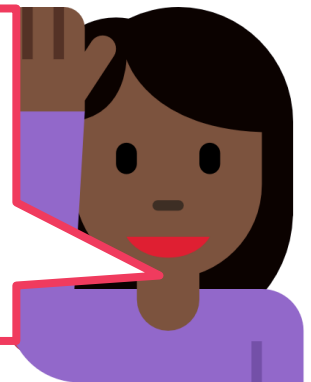
$k = 3$ neighbors



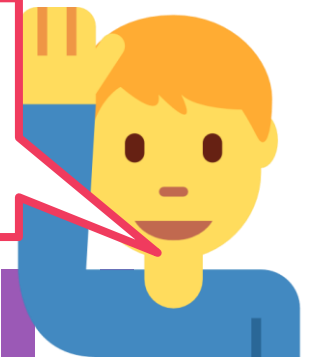
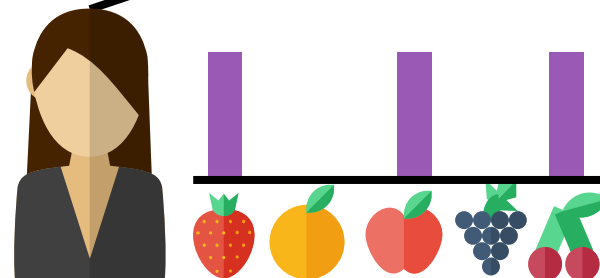
R Find neighbors using the feature vectors corresponding to the users i.e. \mathbf{x}^i e.g. Euclidean/Mahalanobis distance. Once neighbors are known, find out which labels are popular in the neighborhood and recommend top blah labels



May similarly use LwP methods for recommendation too. Average the feature vectors of all users who like a label as the “prototype” of that label – total of L prototypes. At test time, recommend whichever labels are such that their prototypes are close to the test feature vector



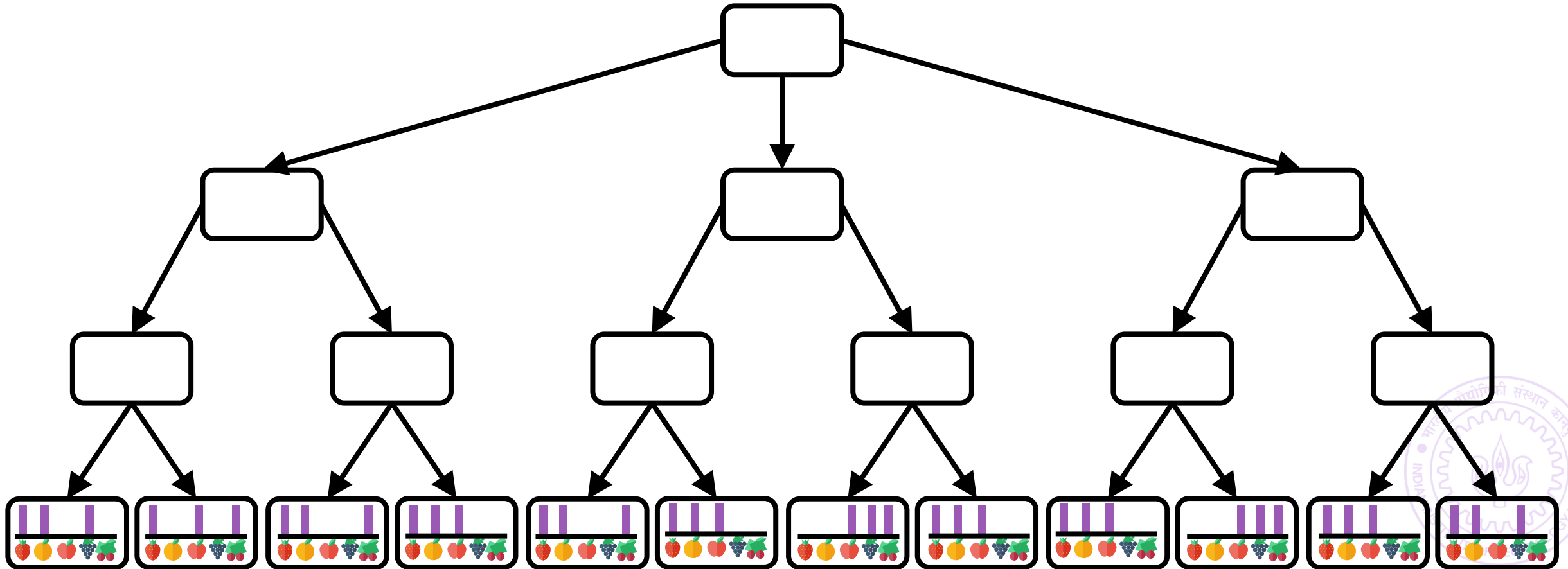
LwP/kNN methods may give better performance for rare labels which have extremely few users liking them



$k = 3$ neighbors

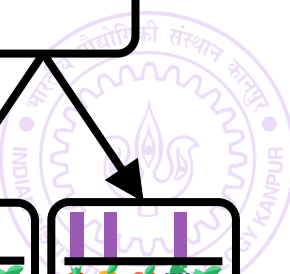
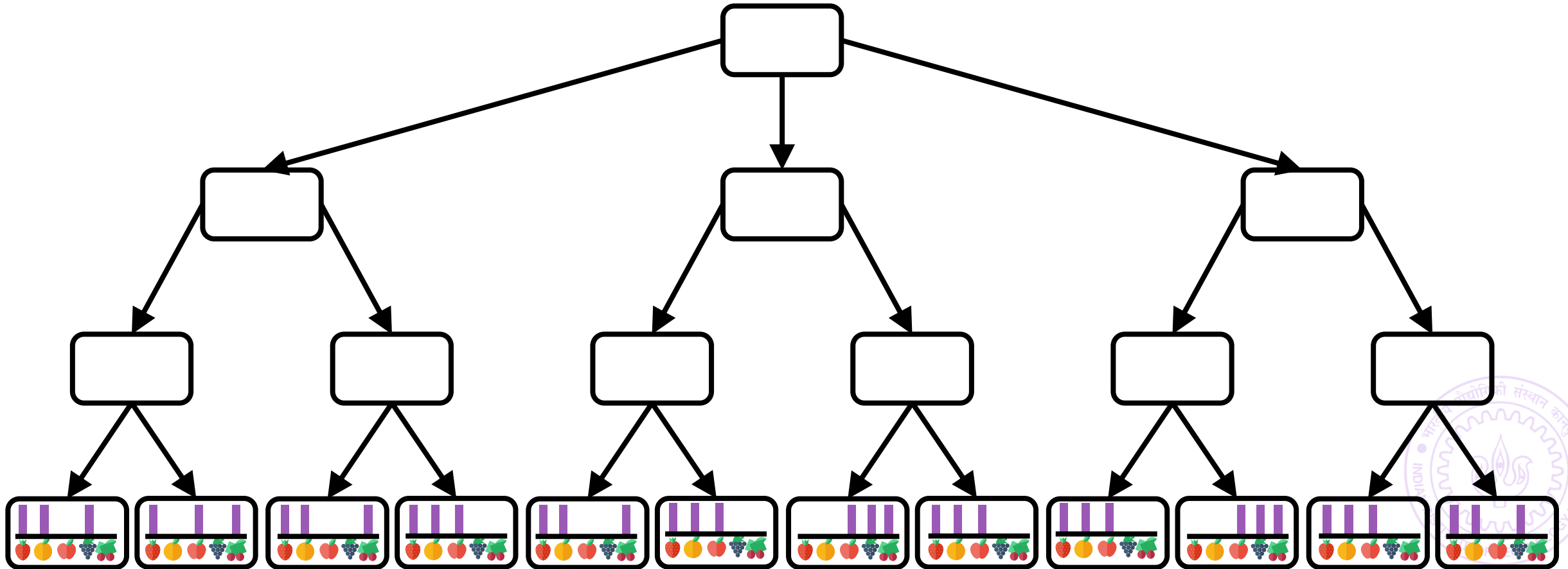
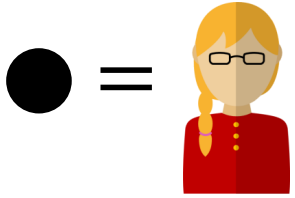
Recommendation using DT

37



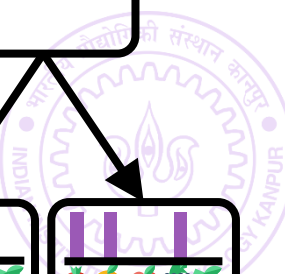
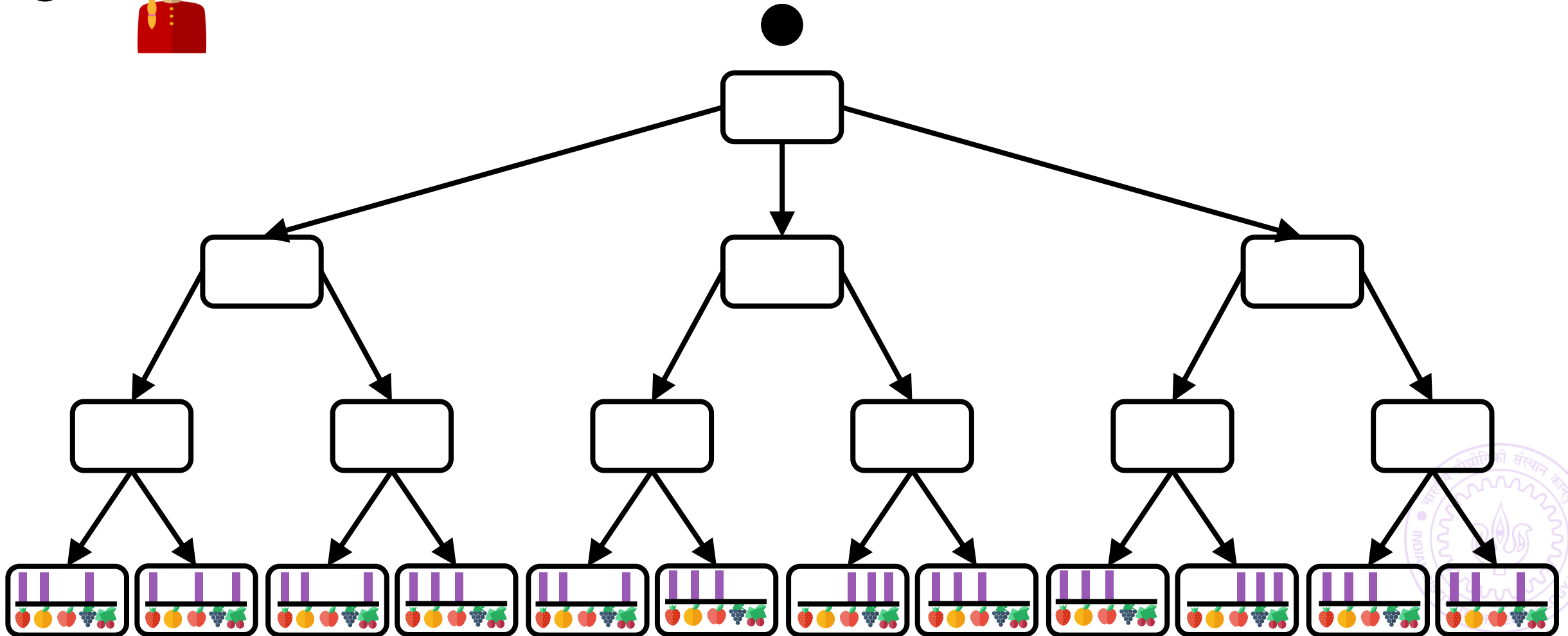
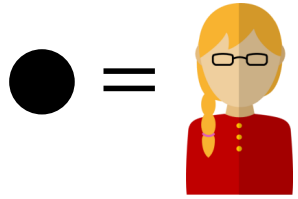
Recommendation using DT

37



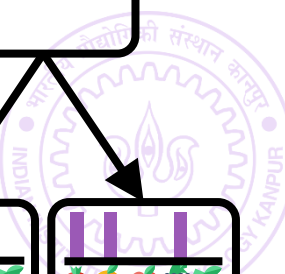
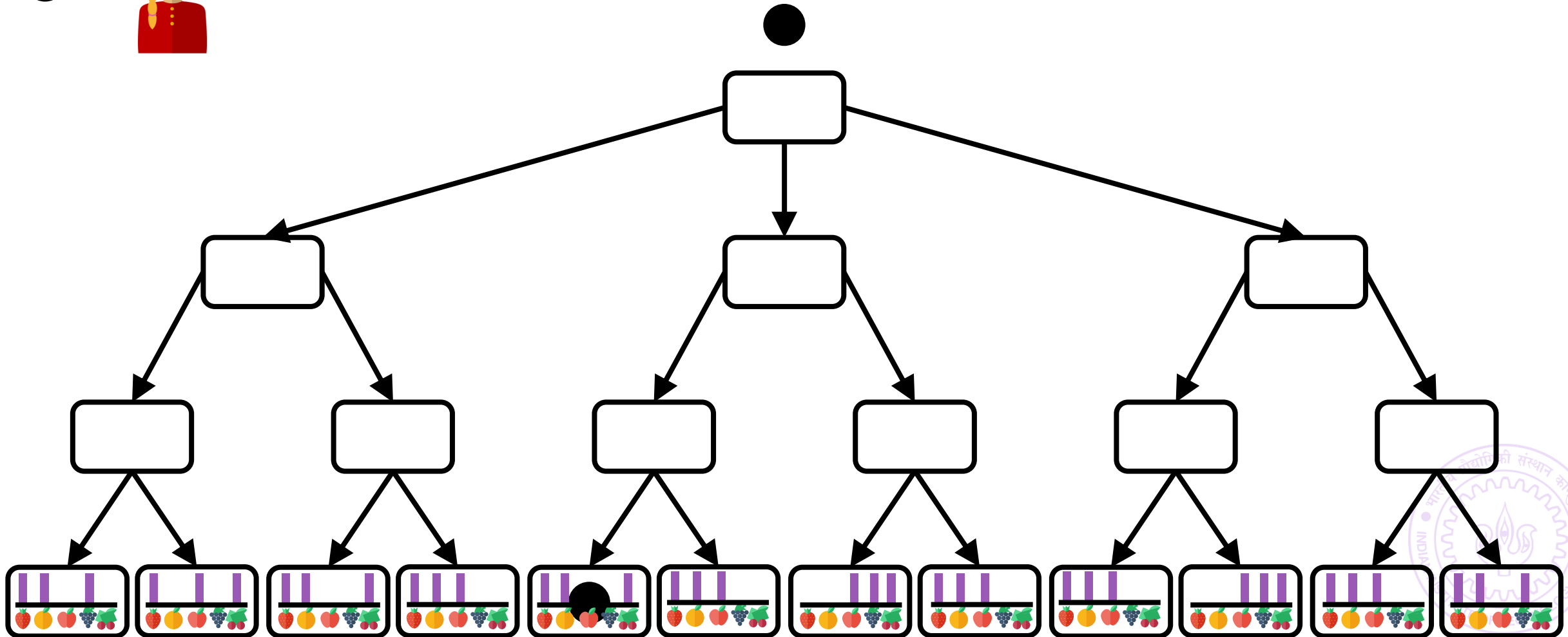
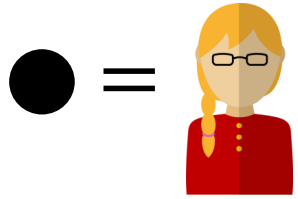
Recommendation using DT

37



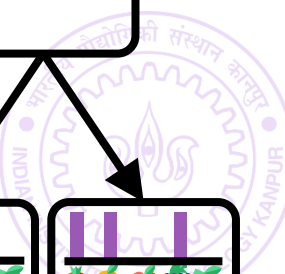
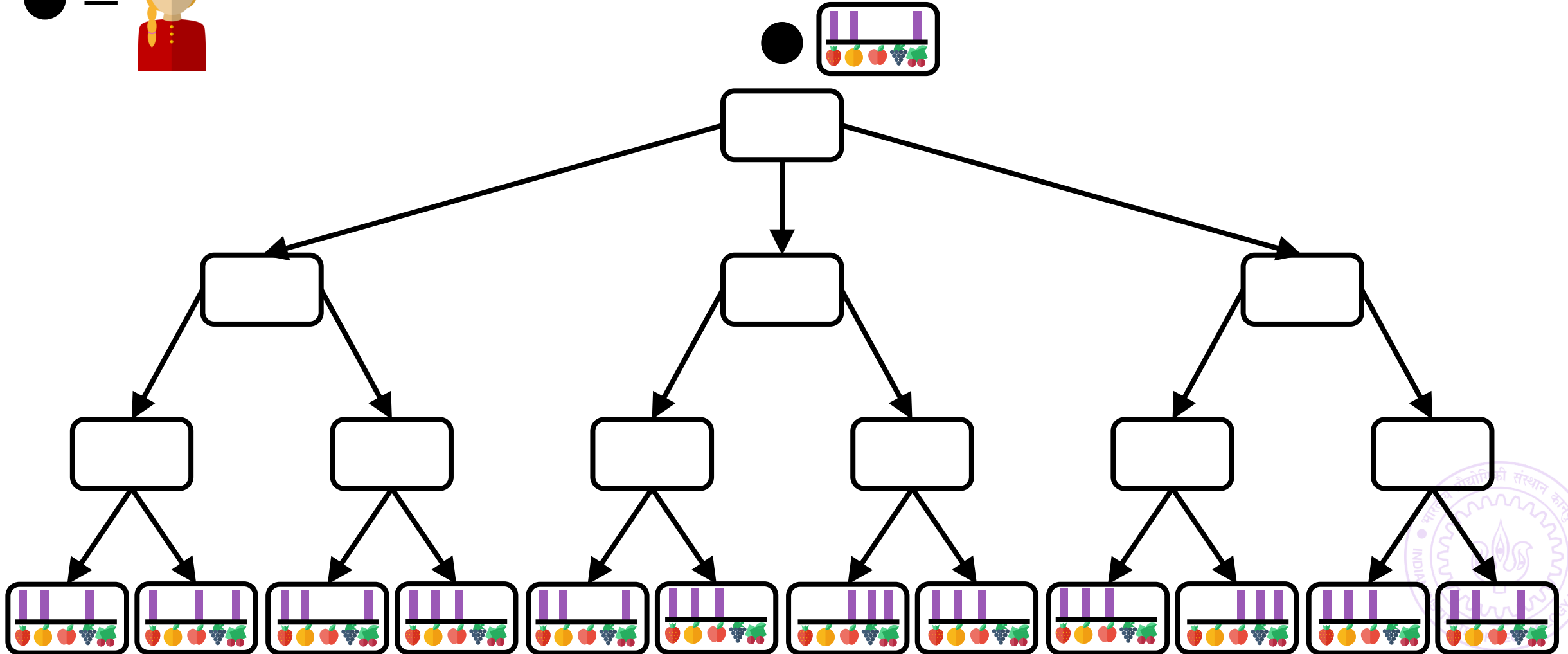
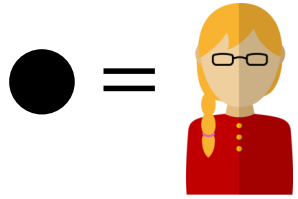
Recommendation using DT

37



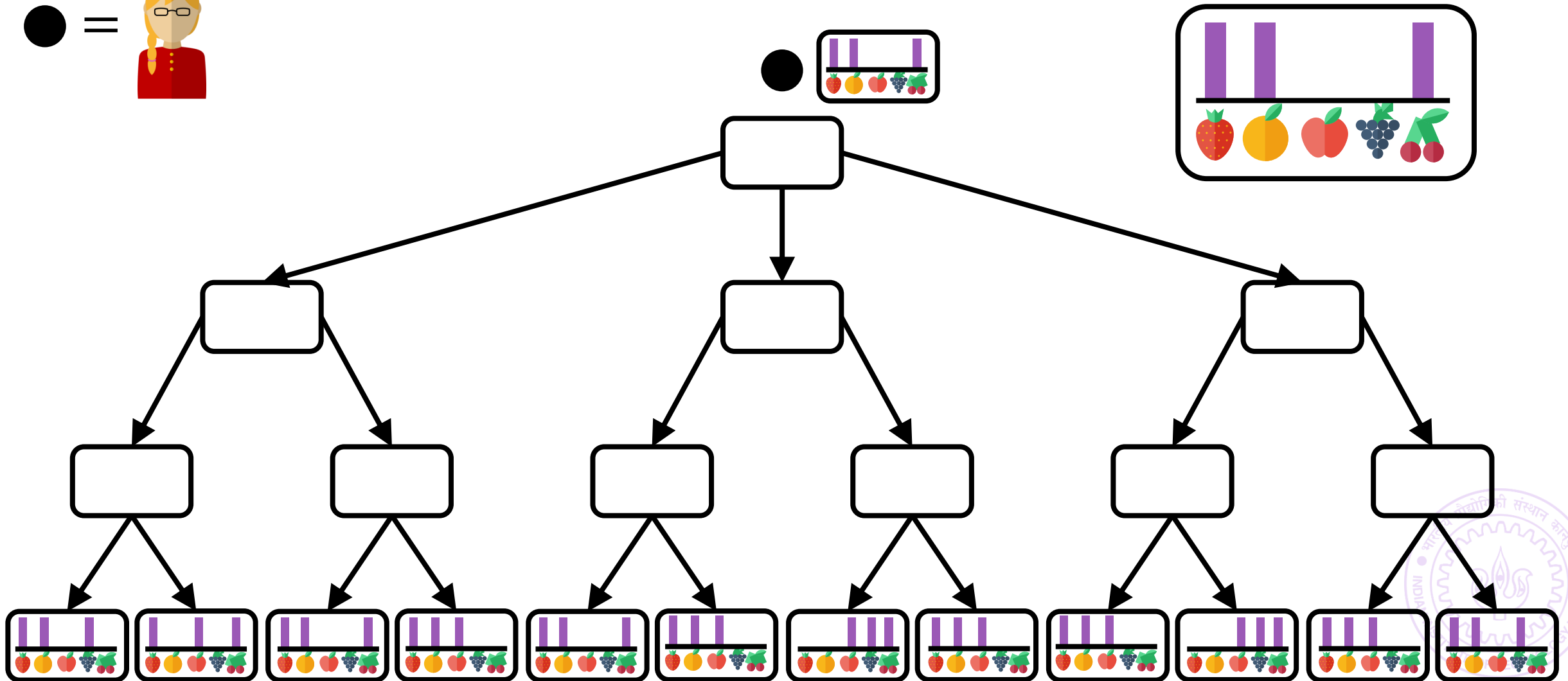
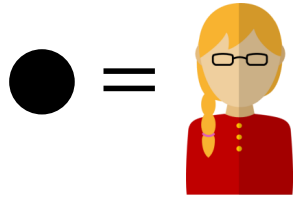
Recommendation using DT

37



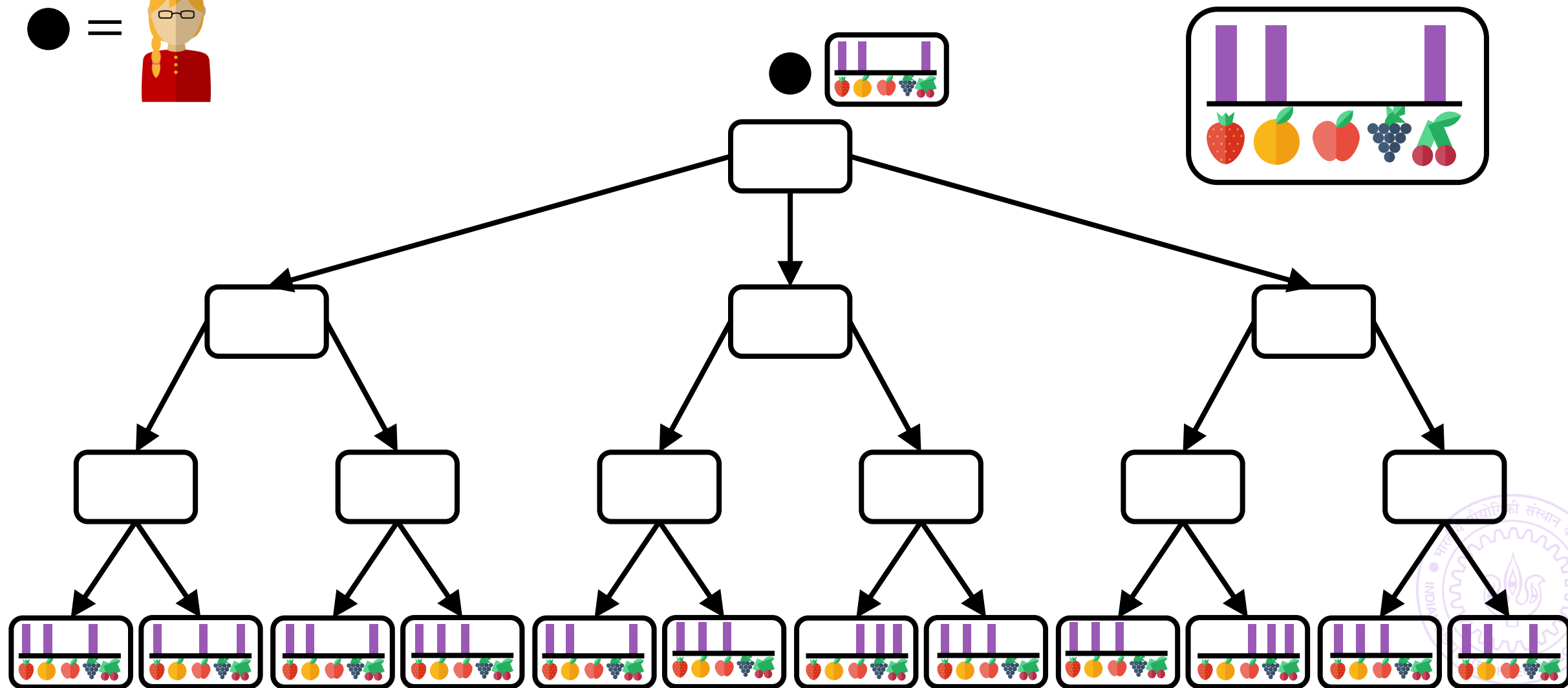
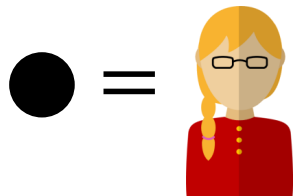
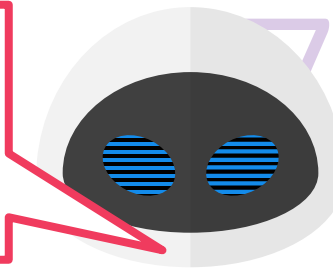
Recommendation using DT

37



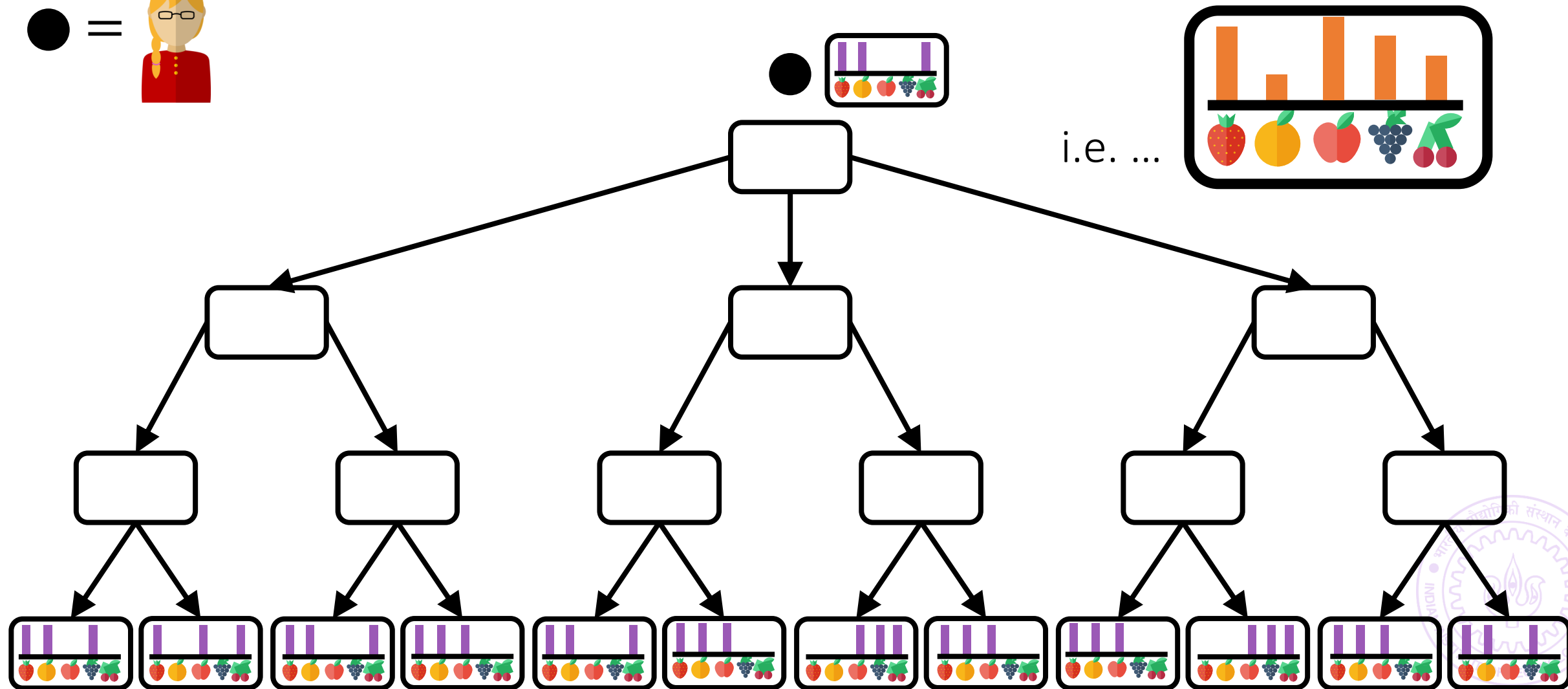
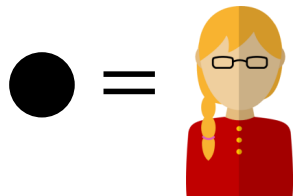
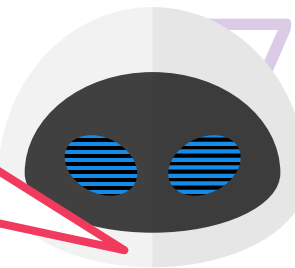
Re

Instead of just storing a few active labels at leaves, can store how popular those labels are as well (e.g. count how many times each active label occurred in train points that reached that leaf) – can allow us to rank labels



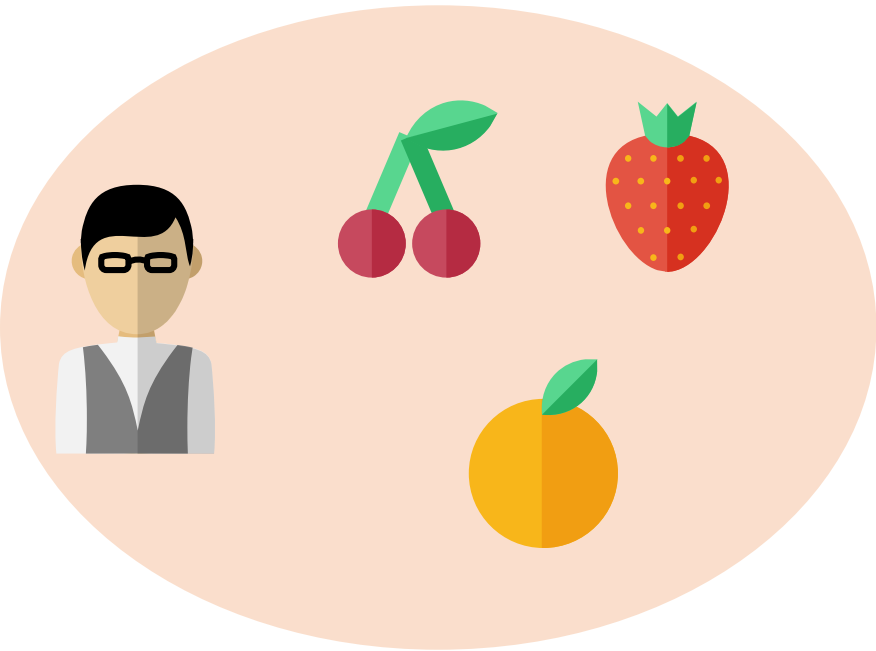
Re

Instead of just storing a few active labels at leaves, can store how popular those labels are as well (e.g. count how many times each active label occurred in train points that reached that leaf) – can allow us to rank labels



Recommendation using Linear Models

45



Treat each label as independent binary classification problem

*This approach is known as **binary relevance** – gives good accuracies but not scalable if L is large ☹*

Total of L binary problems – learn a separate linear model for each

$$\text{E.g. } \mathbf{y}_j^i = \text{sign}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

May use any binary classfn method to learn $\mathbf{w}^j, j = 1 \dots L$ e.g. SVMs

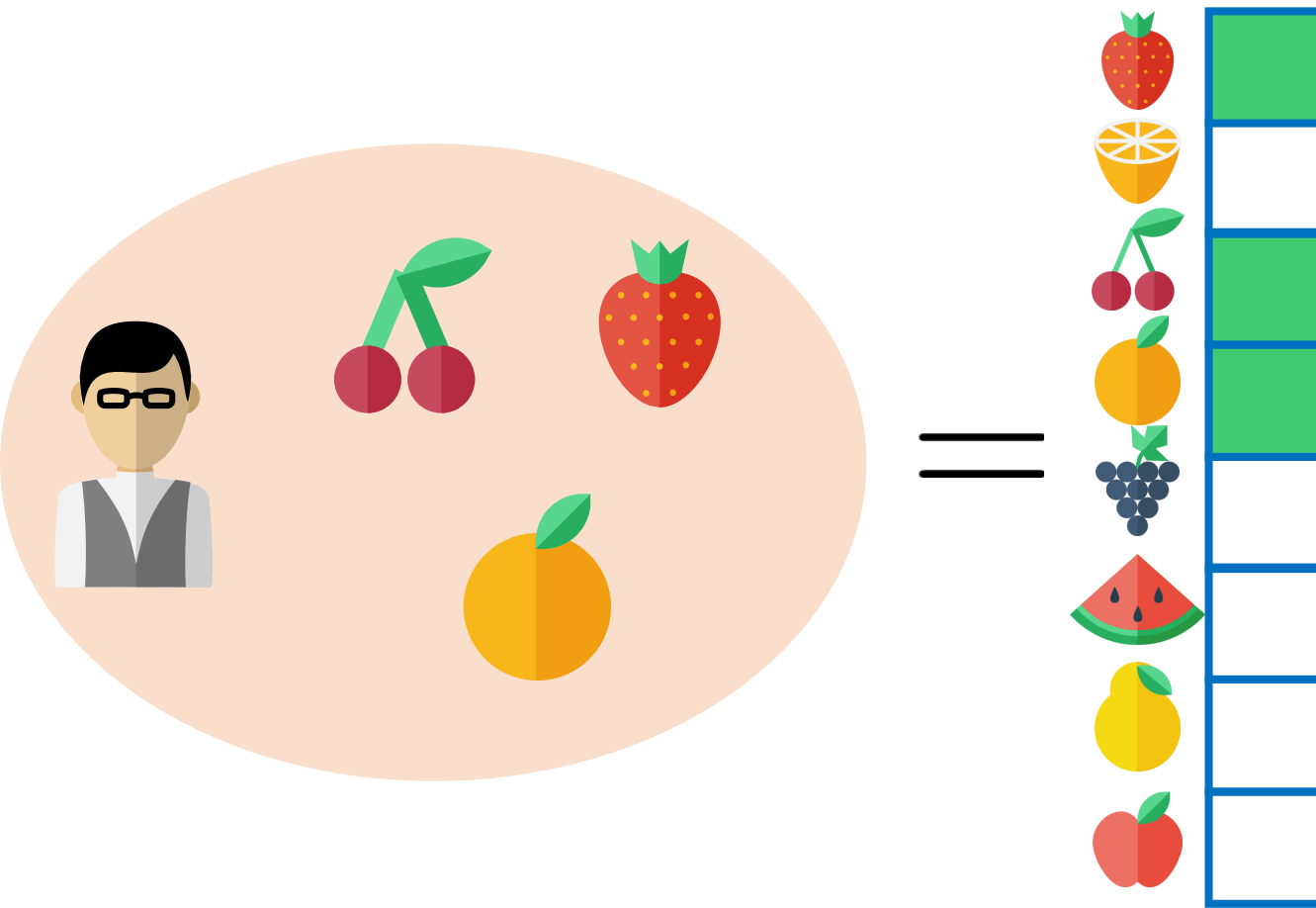
May also use probabilistic methods

E.g. assume user i likes item j with probability $\boldsymbol{\eta}_j^i = \text{sigmoid}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$

Use logistic regression to learn \mathbf{w}^j

Recommendation using Linear Models

45



Treat each label as independent binary classification problem

*This approach is known as **binary relevance** – gives good accuracies but not scalable if L is large ☹*

Total of L binary problems – learn a separate linear model for each

$$\text{E.g. } \mathbf{y}_j^i = \text{sign}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

May use any binary classfn method to learn $\mathbf{w}^j, j = 1 \dots L$ e.g. SVMs

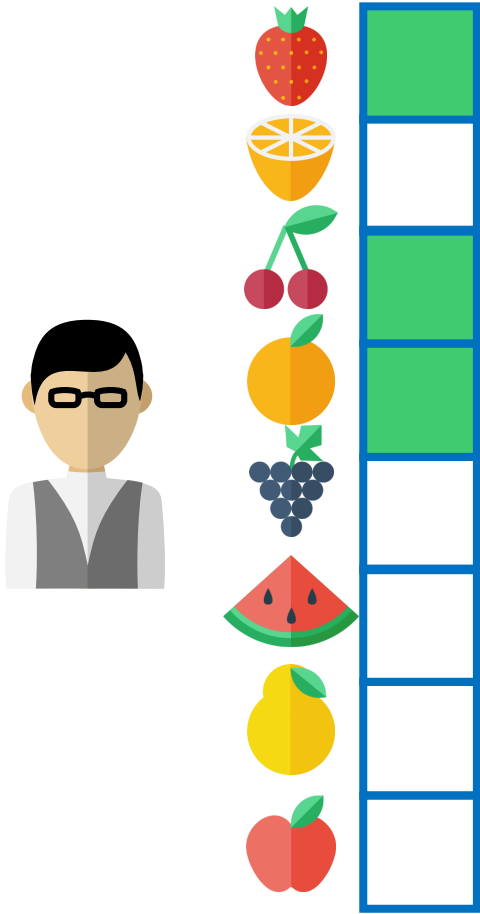
May also use probabilistic methods

E.g. assume user i likes item j with probability $\boldsymbol{\eta}_j^i = \text{sigmoid}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$

Use logistic regression to learn \mathbf{w}^j

Recommendation using Linear Models

45



Treat each label as independent binary classification problem

*This approach is known as **binary relevance** – gives good accuracies but not scalable if L is large ☹*

Total of L binary problems – learn a separate linear model for each

$$\text{E.g. } \mathbf{y}_j^i = \text{sign}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

May use any binary classfn method to learn $\mathbf{w}^j, j = 1 \dots L$ e.g. SVMs

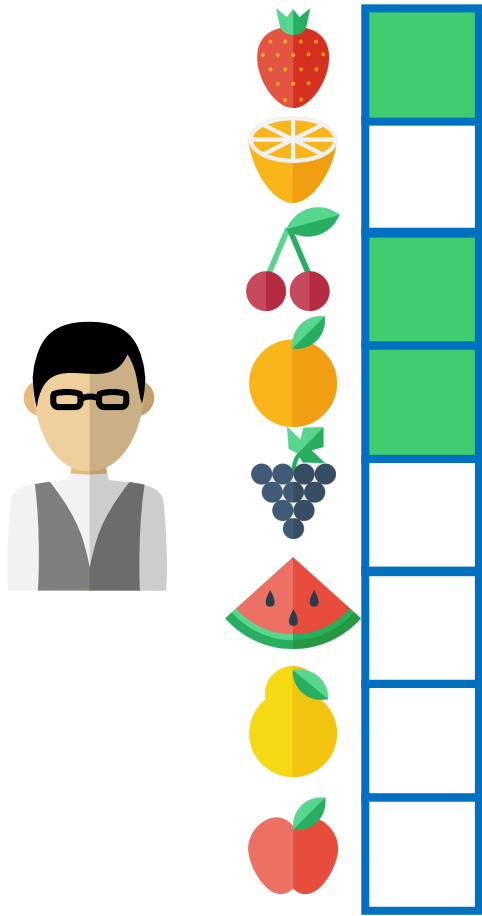
May also use probabilistic methods

E.g. assume user i likes item j with probability $\boldsymbol{\eta}_j^i = \text{sigmoid}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$

Use logistic regression to learn \mathbf{w}^j

Recommendation using Linear Models

45



$$\mathbf{x}^i \in \mathbb{R}^d \quad \mathbf{y}^i \in \{0,1\}^L$$

Treat each label as independent binary classification problem

*This approach is known as **binary relevance** – gives good accuracies but not scalable if L is large ☹*

Total of L binary problems – learn a separate linear model for each

$$\text{E.g. } \mathbf{y}_j^i = \text{sign}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

May use any binary classfn method to learn $\mathbf{w}^j, j = 1 \dots L$ e.g. SVMs

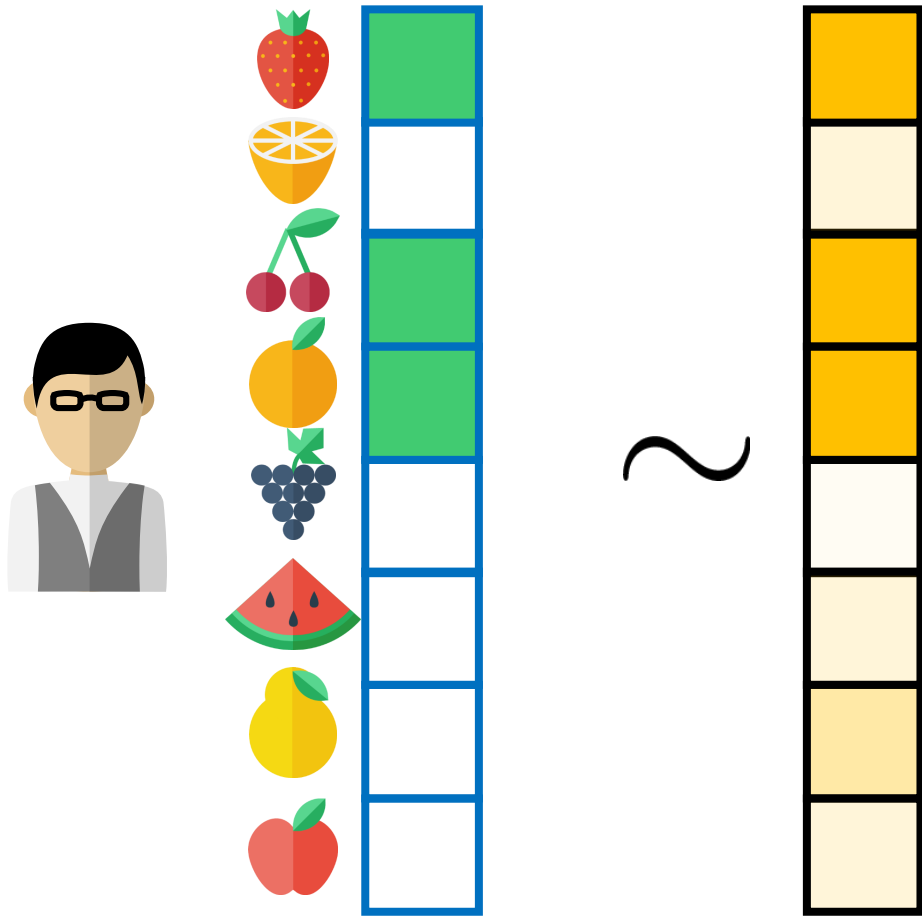
May also use probabilistic methods

$$\text{E.g. assume user } i \text{ likes item } j \text{ with probability } \boldsymbol{\eta}_j^i = \text{sigmoid}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

Use logistic regression to learn \mathbf{w}^j

Recommendation using Linear Models

45



$$\mathbf{x}^i \in \mathbb{R}^d \quad \mathbf{y}^i \in \{0,1\}^L \quad \boldsymbol{\eta}^i \in [0,1]^L$$

Treat each label as independent binary classification problem

*This approach is known as **binary relevance** – gives good accuracies but not scalable if L is large ☹*

Total of L binary problems – learn a separate linear model for each

$$\text{E.g. } \mathbf{y}_j^i = \text{sign}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$$

May use any binary classfn method to learn $\mathbf{w}^j, j = 1 \dots L$ e.g. SVMs

May also use probabilistic methods

E.g. assume user i likes item j with probability $\boldsymbol{\eta}_j^i = \text{sigmoid}(\langle \mathbf{w}^j, \mathbf{x}^i \rangle)$

Use logistic regression to learn \mathbf{w}^j

Recommendation using Latent Variables 50

Very popular concept in recommendation systems

Part of the explosion in number of users and items may be an illusion

Even though there are 1 million users, there may not be 1 million types of users

Even though there are 1 billion items, there may not be 1 billion types of items

This means that there may very well exist a

Small set of item types s.t. prediction of any item possible as combination of these types

One simple idea to exploit this: force the predictors $\mathbf{w}^j \in \mathbb{R}^d$ to be linear combinations of a few, say k meta predictors say $\mathbf{z}^1, \dots, \mathbf{z}^k \in \mathbb{R}^d$

We have to learn both $\mathbf{z}^1, \dots, \mathbf{z}^k$ as well as how to combine them to get \mathbf{w}^j i.e. a combination vector $\mathbf{h}^j \in \mathbb{R}^k$ s.t. $\mathbf{w}^j = \mathbf{h}_1^j \cdot \mathbf{z}^1 + \dots + \mathbf{h}_k^j \cdot \mathbf{z}^k$

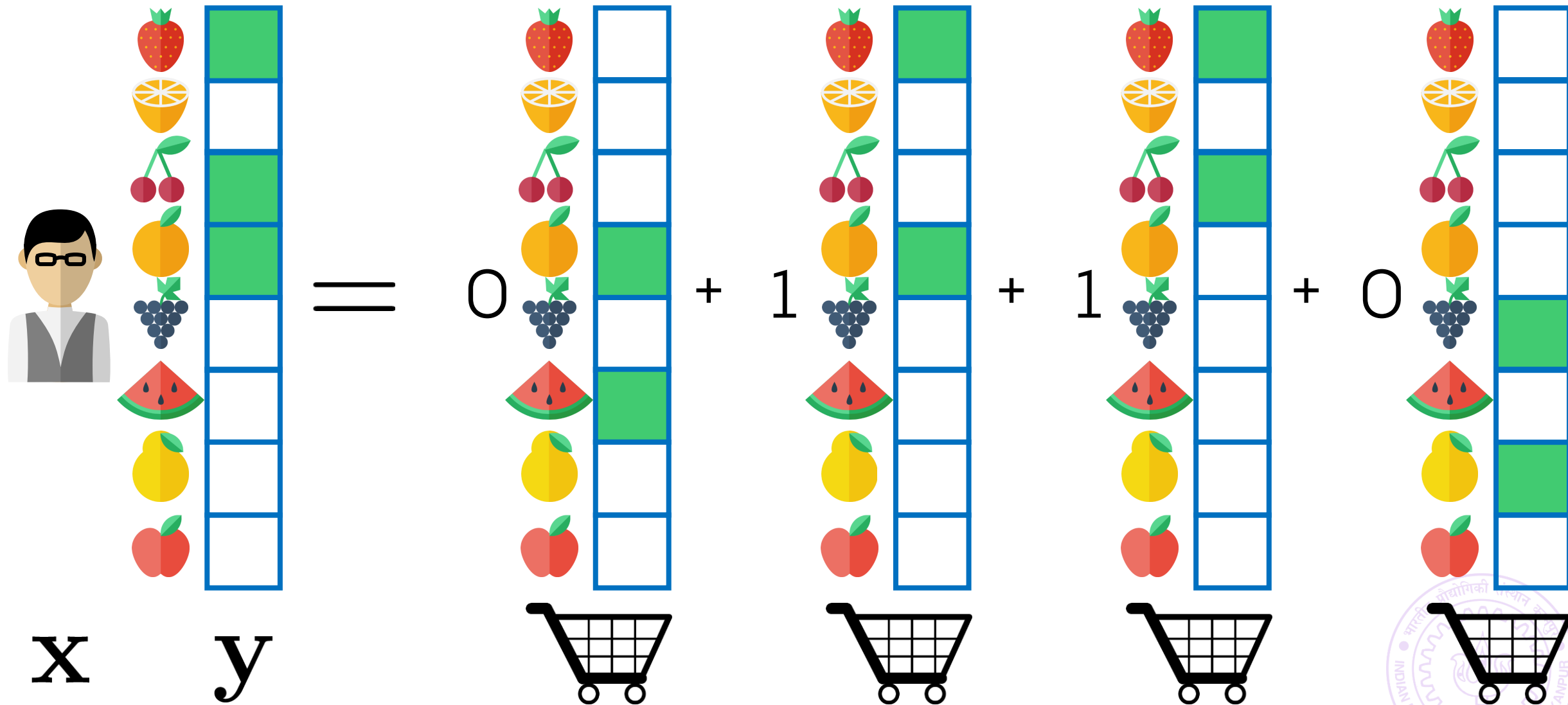
This is equivalent to assuming that $W = [\mathbf{w}^1, \dots, \mathbf{w}^L] \in \mathbb{R}^{d \times L}$ is rank k

Yu et al. Large-scale Multi-label Learning with Missing Labels, ICML 2014

Xu et al. Robust Extreme Multi-label Learning, KDD 2016



Recommendation using Latent Variables 51



Recommendation using Latent Variables 52

Very popular concept in recommendation systems

Part of the explosion in number of users and items may be an illusion

Even though there are 1 million users, there may not be 1 million types of users

Even though there are 1 billion items, there may not be 1 billion types of items

This means that there may very well exist a

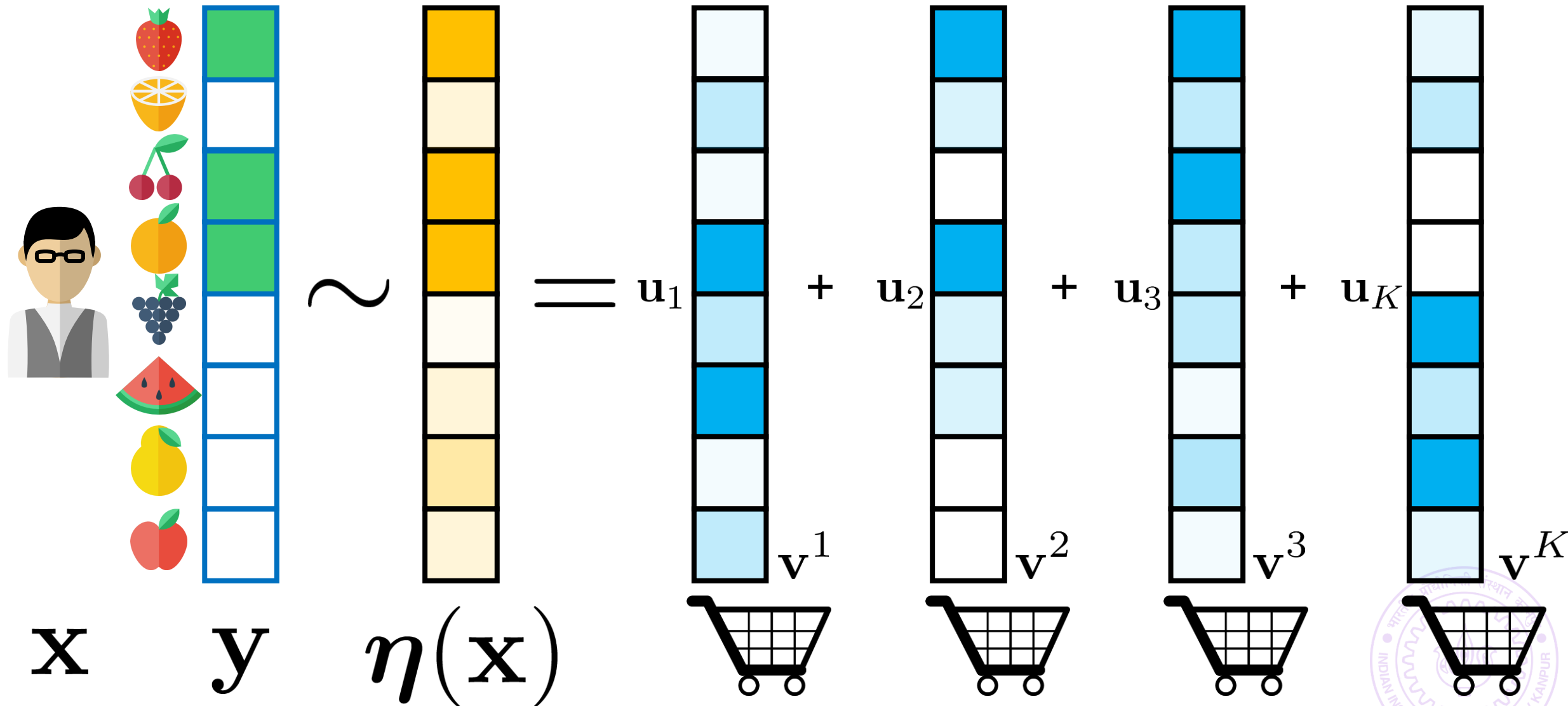
Small set of item types s.t. prediction of any item possible as combination of these types

Small set of item combinations (say sets of items that are frequently purchased together) that every purchase profile can be explained in their terms

One simple idea to exploit this: assume that the like probability vector of every user i.e. $\boldsymbol{\eta}^i \in [0,1]^L$ is a linear combination of a few prototype probability vectors – each prototype corresponding to a purchase profile. Refer to the reference for more technical details



Recommendation using Latent Variables 53



Extreme Multi-label Classification

54

Pros

Very powerful – can extend to millions of items and users

State-of-the-art accuracies and very fast training and predictions

Indications that several major commercial systems use this in some form

Can utilize user and item features easily

Can add new users easily

Cons

No explicit collaboration – does not infer similarity of items/users

May pose an issue if user features are not very informative

Not straightforward to add new items

Recommendations can get stale with time

Common trick – retrain models frequently e.g. nightly, monthly



Recommendation via Online Learning

55

Need for explore exploit techniques

Useful in discovering user tastes of which we are not aware, of which even the user themselves may not be aware

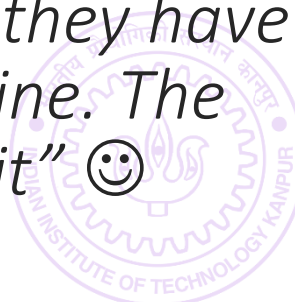
The Multi-armed Bandit Framework

Widely used in recommendation systems to assess utility of a recommendation

Also used in ensemble learning systems and “flighting” systems to assess the utility of various algorithms in offering good performance

Also forms a basis of more powerful models like Reinforcement Learning

The name derives from “one-armed bandit”, a colloquial name for slot machines in casinos – they were called so because of the single lever they have (the single “arm”) which has to be pulled in order to bet on the machine. The machines frequently rob users of their money hence the name “bandit” 😊



The Need for Exploration

56

Collaborative Filtering and Extreme Classification are static techniques

Even if they learn a good set of items for a user, they 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 they liked!

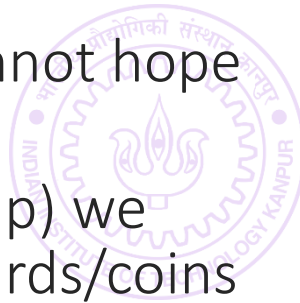
No exploration – usually a small set of items get recommended to everybody - other items get starved and sellers of those items may leave marketplace ☹

Bandit algorithms remedy this by using explore-exploit techniques

Bandit algorithms work with “bandit” feedback. They get feedback only for the action they perform, not for any action that they did not perform

E.g., we can only get clicks on items that we actually recommend to users. Cannot hope to get clicks if an item has not even been displayed on the website as an ad

E.g., if playing Mario, can only get points (rewards) for actions (e.g. jump/stomp) we actually perform. No way to propose an action hypothetically and expect rewards/coins



The Need for Exploration

Collaborative Filtering

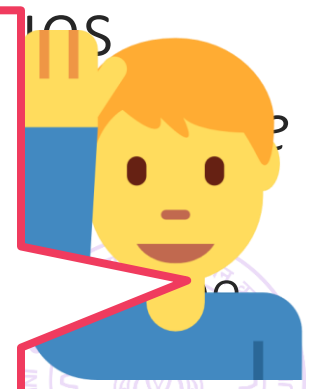
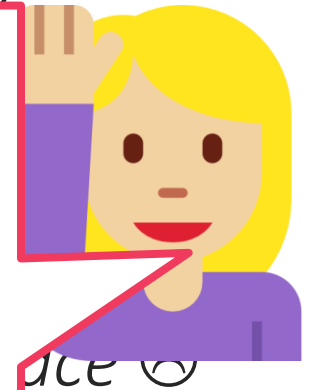
The opposite of bandit setting is “full information” setting where we can reliably predict the reward we would get for this action or that action even without performing that action

Even if they learn a good set of items for a user, they will keep making the

A good example is simple linear regression. We have data $\{(\mathbf{x}^i, y^i)\}_{i=1}^n$ and an “action” is proposing a linear model \mathbf{w} . Since we have all the labels and features with us, we can predict how a certain linear model would perform in our minds without actually committing to that model. We exploit this and conveniently perform offline optimization to propose the best linear model using $\hat{\mathbf{w}}_{\text{MLE}}$ etc

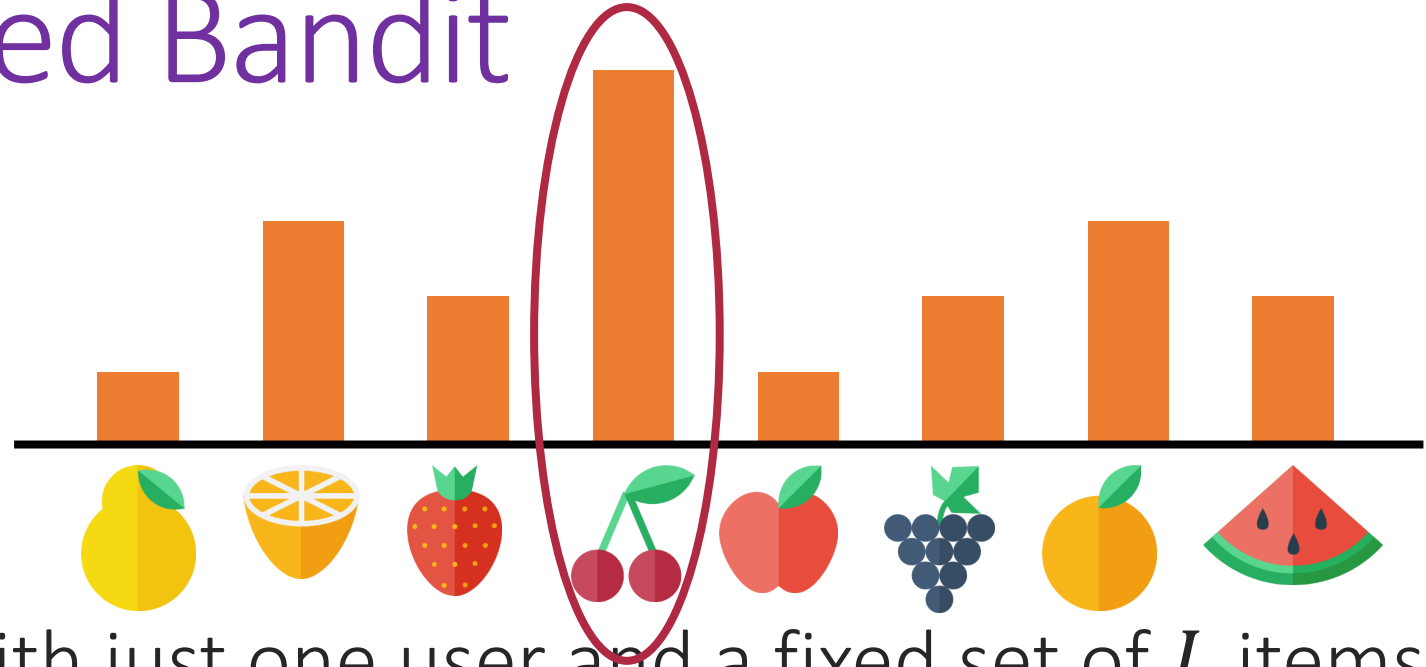
Had this been the bandit setting, data would be hidden from us completely. We would propose a model, say $\hat{\mathbf{w}}^1$ and we would be told the least squares error that $\hat{\mathbf{w}}^1$ incurs on a hidden dataset. Using that information, we may propose a new model $\hat{\mathbf{w}}^2$ and we would be told its performance and so on. In this setting it is not simple to predict performance of models other than those we have proposed.

E.g., if playing Mario, can only get points (rewards) for actions (e.g. jump/stomp) we actually perform. No way to propose an action hypothetically and expect rewards/coins



The Multi-armed Bandit

58



Will use a toy model with just one user and a fixed set of L items

To handle multiple users we need something called contextual bandits – CS773

Each item $j \in [L]$ has an associated prob p_j of user buying it if recommended

The probability values are unknown to us – we need to discover them using RecSys itself

We assume user cannot buy items that have not been recommended to them

Want to maximize total number of purchases made by the user

If we knew the values of p_j then we would just keep recommending item $j^ = \arg \max p_j$ again and again and get the maximum expected purchase rate*

PURE EXPLORATION

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

Pros

Simple algorithm

Gives us idea of p_j for 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

60

Main drawback of PureExp is that it keeps recommending an item even if after, say $\frac{T_0}{2}$ recommendations of that item, it is still 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 \hat{p}_j is a very good estimate of p_j

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

Give item j some form of benefit of doubt if T_j is small



The UCB Algorithm

61

UPPER CONFIDENCE BOUND

1. One user, L items
2. For $t = 1, 2, 3, \dots$,
 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 $1/\sqrt{T_j}$

T_j # times item j was recommended

n_j # times item j was purchased

If an item is starved i.e. T_j is small, it gets large benefit of doubt

Thus, starved items will get recommended eventually

However, if $T_{j'}$ is large for item j' , then tiny benefit

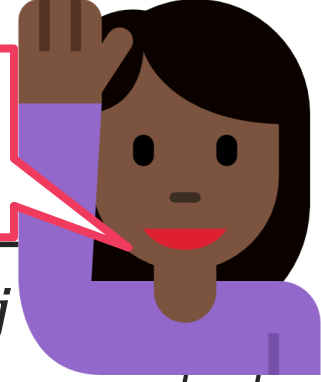
In that case, item j' recommended only if $\hat{p}_{j'}$ itself is large

UCB gives every item a fighting chance but ditches losers quickly



The UCB Algorithm

Initially, may want to recommend all items at least once to avoid divide-by-zero errors



UPPER CONFIDENCE BOUND

1. One user, L items
2. For $t = 1, 2, 3, \dots$,
 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 $1/\sqrt{T_j}$

T_j # times item j was recommended

n_j # times item j was purchased

If an item is starved i.e. T_j is small, it gets large benefit of doubt

Thus, starved items will get recommended eventually

However, if $T_{j'}$ is large for item j' , then tiny benefit

In that case, item j' recommended only if $\hat{p}_{j'}$ itself is large

UCB gives every item a fighting chance but ditches losers quickly



Explore Exploit

63

Pros

- Online and not static, learns continuously (even if user tastes change)
- Does not get stale very easily – explores!
- Can utilize user and item features easily (contextual bandits)
- Can add new users and new items easily

Cons

- No explicit collaboration – does not infer similarity of items/users
 - Modifications do exist for “collaborative contextual bandits” e.g. Li et al ICML 2017
 - Need powerful contextual models to handle large item sets
 - Algorithms more complicated, expensive in contextual bandits
- Explore exploit algorithms are a prominent RecSys tool



In practice, ensemble methods popular in RecSys

Most commercial recsys are massive ensembles with 100s of algorithms running in tandem. Exact details usually closely guarded secrets

Recent advances focus on getting best of all many methodologies

E.g. matrix factorization where rows and columns have side features

E.g. collaborative multi-armed bandits – identifies similar items/users

Reinforcement Learning generalizes notion of bandit algorithms

Bandit algorithms usually assume that any action/proposal only generates a reward but does not change the reward function

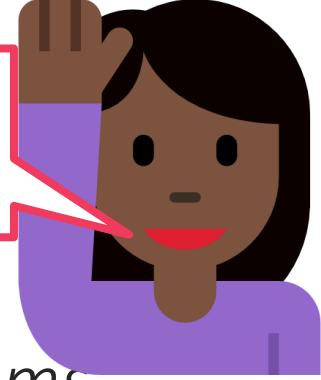
E.g. recommending an item (whether liked or disliked) not expected to change user's tastes drastically

RL removes this assumption – not only do our actions generate feedback from user, they may also fundamentally change the way the user gives future feedback – need to be much more careful in this model



Comm

Think of a game of Mario – not only may a false move give less reward right now, it may also make it impossible to win the game overall ☹



In practice, ensemble methods popular in RecSys

Most commercial recsys are massive ensembles with 100s of algorithms running in tandem. Exact details usually closely guarded secrets

Recent advances focus on getting best of all many methodologies

E.g. matrix factorization where rows and columns have side features

E.g. collaborative multi-armed bandits – identifies similar items/users

Reinforcement Learning generalizes notion of bandit algorithms

Bandit algorithms usually assume that any action/proposal only generates a reward but does not change the reward function

E.g. recommending an item (whether liked or disliked) not expected to change user's tastes drastically

RL removes this assumption – not only do our actions generate feedback from user, they may also fundamentally change the way the user gives future feedback – need to be much more careful in this model

