Date No

Week 02

01] Recommended systems

*] A recommendation system predicts what a user will like based on:

- The past behaviour (what they clicked, liked, watched or bought)
- Information about the items and users

Predicting movie ratings

*] User rates movies using zero to five stars

Predicting movie ratings

User rates movies using ~~one~~ zero to five stars

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

n_u = no. of users

n_m = no. of movies

$r(i,j)=1$ if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j)=1$)

$n_u = 4$

$n_m = 5$

$r(1,1) = 1$

$r(3,1) = 0$

$y^{(3,2)} = 4$

02] Using per-item features

What if we have features of the movies?

$n_u = 4$
$n_m = 5$
$n = 2$

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
→ Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

$x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$

$x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$

For user 1: Predict rating for movie i as: $w^{(1)} \cdot x^{(i)} + b^{(1)}$ ← just linear regression

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ $b^{(1)} = 0$ $x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$ $w^{(1)} \cdot x^{(3)} + b^{(1)} = \underline{4.95}$

→ For user j: Predict user j's rating for movie i as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

Cost Function

*) $r(i,j) = 1$ if user j has rated movie i (0 otherwise)

*) $y^{(i,j)}$ = rating given by user j on movie i (if defined)

*) $w^{(j)}, b^{(j)}$ = parameters of user j

*) $x^{(i)}$ = feature vector for movie i

*) For user j and movie i, predict rating:

$$w^{(j)} \cdot x^{(i)} + b^{(j)}$$

*) m_j = no. of movies rated by user j

Date No

To learn $w^{(j)}, b^{(j)}$

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

n = no. of features

m_j = cut off

To learn parameters $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, w^{(n_u)}, b^{(n_u)}$ for all users

$$J\left(\begin{matrix} w^{(1)}, \ldots, w^{(n_u)} \\ b^{(1)}, \ldots, b^{(n_u)} \end{matrix}\right) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \underbrace{\left(w^{(j)} \cdot x^{(i)} + b^{(j)}\right)}_{f(x)} - y^{(i,j)}\Big)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

03] Collaborative filtering algorithm

*) Uses user-behaviour pattern — not item features

*) Idea → Users who behaved like you also liked these items.

a) User - User CF = Find similar uses

b) Item - Item CF = Find similar items based

Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0 ←	?	? $x^{(2)}$
Cute puppies of love	?	4	0	? ←	?	? $x^{(3)}$
Nonstop car chases	0	0	5	4 ←	?	?
Swords vs. karate	0	0	5	? ←	?	?

$w^{(1)} = \begin{bmatrix}5\\0\end{bmatrix}$, $w^{(2)} = \begin{bmatrix}5\\0\end{bmatrix}$, $w^{(3)} = \begin{bmatrix}0\\5\end{bmatrix}$, $w^{(4)} = \begin{bmatrix}0\\5\end{bmatrix}$

$b^{(1)} = 0$, $b^{(2)} = 0$, $b^{(3)} = 0$, $b^{(4)} = 0$ ←

using $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$w^{(1)} \cdot x^{(1)} \approx 5$
$w^{(2)} \cdot x^{(1)} \approx 5$
$w^{(3)} \cdot x^{(1)} \approx 0$
$w^{(4)} \cdot x^{(1)} \approx 0$

→ $x^{(1)} = \begin{bmatrix}1\\0\end{bmatrix}$ ↑

Cost function

Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots w^{(n_u)}, b^{(n_u)}$

to learn $x^{(i)}$:

$$J(x^{(i)}) = \frac{1}{2} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2$$

$$+ \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$$

To learn $x^{(1)}, x^{(2)}, \ldots x^{(n_m)}$:

$$J(x^{(1)}, x^{(2)}, \ldots x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Put → Cost function to learn $w^{(1)}, b^{(1)}, \ldots w^{(n_u)}, b^{(n_u)}$.

&

Cost function to learn $x^{(1)}, \ldots x^{(n_m)}$

together

$$J(w,b,x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w_j^{(i)} \cdot x^{(i)} + b^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Goal is to minimize $w^{(1)} \ldots w^{(n_u)}$, $b^{(1)} \ldots b^{(n_u)}$, $x^{(1)} \ldots x^{(n_m)}$

Date No

04] Binary labels

*] In many modern systems, we don't have explicit ratings (like 1 to 5 stars). Instead we have implicit feedback

ex:- ·) Fav/likes = positive feedback
·) View/Click = possible interest
·) purchase = strong positive signal

These are binary labels

1 = If user interacted
0 = otherwise

*) "0" doesn't always mean dislike (It could just mean not seen yet)

Cost function

Predict that the probability of $y^{(i,j)} = 1$ is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

where $g(z) = \frac{1}{1+e^{-z}}$

Date No

Loss for binary labels

$y^{(i,j)}$: $f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

Loss for single example

$$J_{(w,b,x)} = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

$$g(w^{(j)} \cdot x^{(i)} + b^{(j)})$$

for all examples

05] Mean Normalization

Users who have not rated any movies

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	
Love at last	5	5	0	0	?	0
Romance forever	5	?	?	0	?	0
Cute puppies of love	?	4	0	?	?	0
Nonstop car chases	0	0	5	4	?	0
Swords vs. karate	0	0	5	?	?	0

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{\substack{w^{(1)},\dots,w^{(n_u)} \\ b^{(1)},\dots,b^{(n_u)} \\ x^{(1)},\dots,x^{(n_m)}}} \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)}\right)^2+\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2+\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2$$

$w^{(5)}=\begin{bmatrix}0\\0\end{bmatrix}$ $b^{(5)}=0$ $w^{(5)}\cdot x^{(i)}+b^{(5)}$

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \begin{matrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{matrix} \qquad \mu=\begin{bmatrix}2.5\\2.5\\2\\2.25\\1.25\end{bmatrix}$$

$$\begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user j, on movie i predict:

$w^{(j)}\cdot x^{(i)}+b^{(j)}+\mu_i$

$y^{(i,j)}$

$w^{(j)}\cdot b^{(j)}\cdot x^{(i)}$

User 5 (Eve):

$w^{(5)}=\begin{bmatrix}0\\0\end{bmatrix}$ $b^{(5)}=0$ $\underbrace{w^{(5)}\cdot x^{(i)}+b^{(5)}}_{0}+\mu_i=2.5$

06] Implementation in Tensor Flow

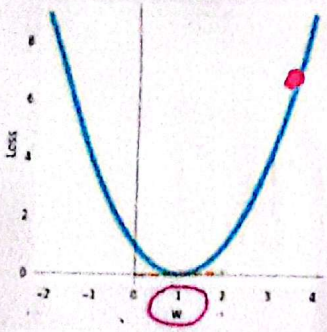
$J = (\overbrace{wx}^{f(x)} - \overbrace{1}^{y})^2$

Gradient descent algorithm

Repeat until convergence

$w = w - \alpha \frac{d}{dw} J(w,b)$

Fix b = 0 for this example



Custom Training Loop

```
w = tf.Variable(3.0)
x = 1.0
y = 1.0 # target value
alpha = 0.01

iterations = 30
for iter in range(iterations):
    # Use TensorFlow's Gradient tape to record the steps
    # used to compute the cost J, to enable auto differentiation.
    with tf.GradientTape() as tape:
        fwb = w*x
        costJ = (fwb - y)**2

    # Use the gradient tape to calculate the gradients
    # of the cost with respect to the parameter w.
    [dJdw] = tape.gradient( costJ, [w] )

    # Run one step of gradient descent by updating
    # the value of w to reduce the cost.
    w.assign_add(-alpha * dJdw)
```

Tf.variables are the parameters we want to optimize

Auto Diff

Auto Grad

$\frac{\partial}{\partial w} J(w)$

tf.variables require special function to modify

Implementation in TensorFlow

Gradient descent algorithm

Repeat until convergence

$w = w - \alpha \frac{d}{dw} J(w,b,x)$

$b = b - \alpha \frac{d}{db} J(w,b,x)$

$X = X - \alpha \frac{d}{dx} J(w,b,x)$

```
# Instantiate an optimizer.
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200
for iter in range(iterations):
    # Use TensorFlow's GradientTape
    # to record the operations used to compute the cost
    with tf.GradientTape() as tape:

        # Compute the cost (forward pass is included in cost)
        cost_value = cofiCostFuncV(X, W, b, Ynorm, R,
                                   num_users, num_movies, lambda)
                                   # n_u       # n_m

    # Use the gradient tape to automatically retrieve
    # the gradients of the trainable variables with respect to
    # the loss
    grads = tape.gradient( cost_value, [X,W,b] )

    # Run one step of gradient descent by updating
    # the value of the variables to minimize the loss.
    optimizer.apply_gradients( zip(grads, [X,W,b]) )
```

Dataset credit: Harper and Konstan. 2015. The MovieLens Datasets: History and Context

Date No

07) Finding Related Items

*) The features $x^{(i)}$ of item i are quite hard to interpret.

*) To find other items related to it,

find item k with $x^{(k)}$ similar to $x^{(i)}$

i.e with smallest Distance

$$\sum_{l=1}^{n} \left(x_l^{(k)} - x_l^{(i)}\right)^2$$

$$\| x^{(k)} - x^{(i)} \|^2$$

Limitations of Collaborative Filtering

1) Cold start problem

*) How to rank new items that few users have rated ?

*) How to show something resonable to new users who have rated few items

2) Use side information about items or users

*) Item: Genre, movie stars, studio, . . .

*) User: Demographics (age, gender, location), expressed pereferences ; - --

08) Collaborative Filtering vs Content-based Filtering

Collaborative Filtering

*) It recommends items based on user behaviour (ratings, cliks, likes) - not on features

*) There are two major types

1) Memory based

These methods use the entire collection of user-item interaction data to compute similarities and make predictions. There are two main categories

User-Based Collaborative Filtering

*) Identifies users who are similar to target users (uses who have rated items similarly) and then recommends items that those "neighboring"

users have liked but the target user has not yet experienced.

Item-Based Collaborative Filtering

*) Identifies and calculate in similarity between items (items that tend to be rated similarly by many users). when making a recommendation for a user, it looks at items the user has liked and suggest other items that are highly similar to them.

2] Model-Based

*) These methods use the user-item interaction data to train a model (ML or DL) to learn underlying patterns, which is then used to make predictions

Matrix Factorization:

*) Techiniques like Singular Value Decomposition (SVD) or Alternating Least Squares (ALS) that decompose the user-item iteraction matrix into two lower dimensional matrices

*) One represent users and other represent items

*) The dot product of user's vector and item's vector gives the predicted preferences

Clustering Models

*) Grouping similar users or items into clusters and then using the cluster's characteristics to make recommendations

Deep Learning Models

*) Using neural networks (Neural Collaborative Filtering) to learn complex non-linear interactions between users and items.

Content-Based

*) It recommends items similar to what user liked before, based on item features (metadata)

How it works =)

*) Each item (ex- movie) is represented by a vector of features

*) Each user has a preference vector - learned from the items they rated / liked

*) The system computes similarity (Cosine / dot product) between
   -> User preference vector
   -> Item feature vectors

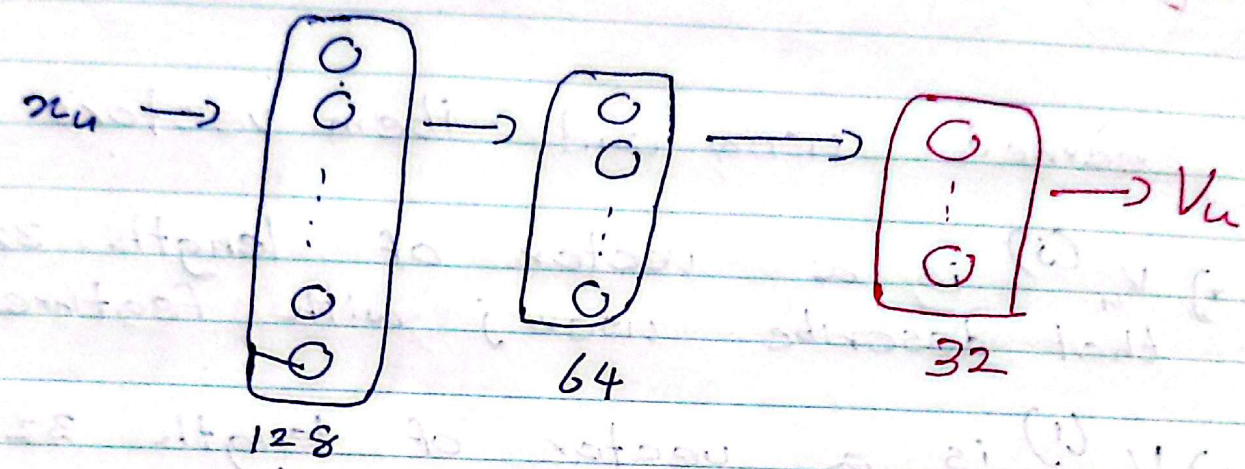
*) Then recommends items with highest similarity scores.

Summary table

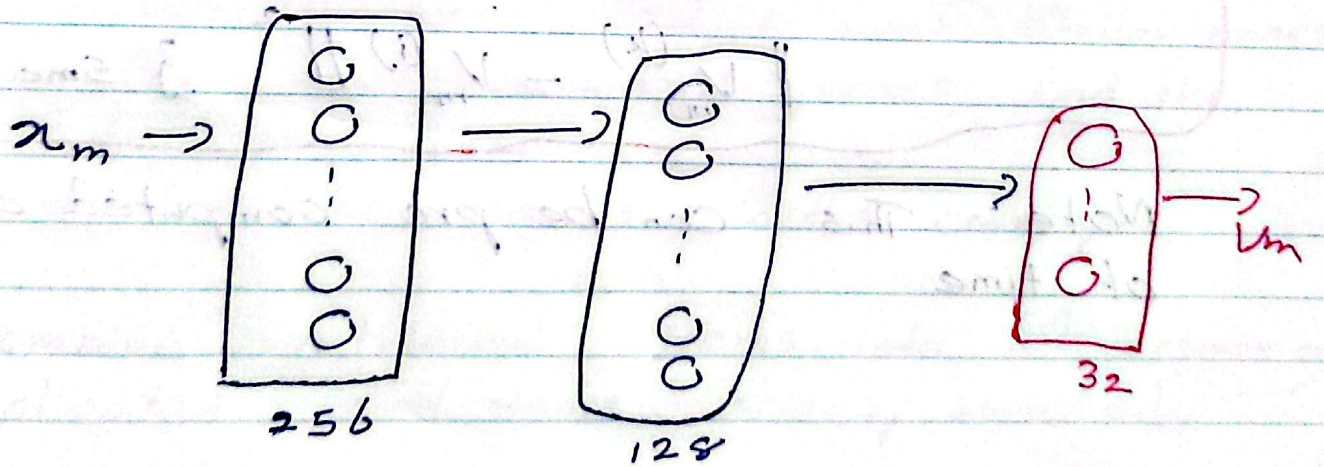
Feature	Content - Based	Collaborative
Based on	Item metadata/ feature	User - item interaction
Need user ratings?	Not necessiraly	Yes
Can handle new items easily?	Yes (if item features exist)	No
Can handle new users easily?	No	No
Learns hidden patterns	No	Yes
Explainability	Easy	Harder

09] Deep learning for content-based filtering

*] $x_u \rightarrow v_u$ (User Network)



*] $x_m \rightarrow v_m$ (movie network)



Prediction : $v_u^{(j)} \cdot v_m^{(i)}$

$g(v_u^{(j)} \cdot v_m^{(i)})$ to predict the probability that is $y^{(i,j)}$ is 1

Cost Function =) $J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2$

+ NN Regularization term

Learned user and item vectors:

*) $v_u^{(j)}$ is a vector of length 32 that describe user j with features $x_u^{(j)}$

*) $v_m^{(i)}$ is a vector of length 32 that describes movie i with features $x_m^{(i)}$

To find ^ Similar movies to movie i:

$$\| v_m^{(k)} - v_m^{(i)} \|^{-2} \text{ small}$$

Note: This can be pre computed ahead of time

10] Recommending from a large catalogue

*) When you have millions of items, you can't compute similarity with every user in real time (Too slow)

*) We use a two stage pipeline for this

1) Retrieval

*) Goal → Quickly narrow down from millions to hundred of likely items

*) Generate large list of plausible item candidates (ANN, FAISS, ScaNN)

ex:- 1) For each of the last 10 movies watched by the user, find 10 most similar movies
2) For most viewed 3 genres, find the top 10 movies
3) Top 20 movies in the country

*) Combine retrieved items into list, removing duplicates and items already watched/purchased.

Ranking

*) Take list retrieved and rank using learned model (MLP)

*) Display ranked items to user

Retrieval Step

*) Retrieving more items results in better performance but slower recommendations

*) To analyze / optimize the trade off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e, $p(y^{(i,j)}) = 1$ of items displayed to user are higher)