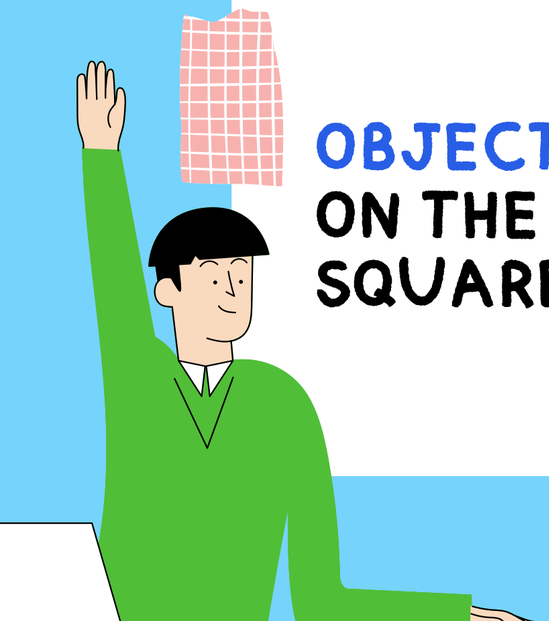
PROJECT2

RECOMMENDATION SYSTEM

BACKGROUND

MOVIELENS IS A MOVIE RECOMMENDATION SYSTEM OPERATED BY GROUPLENS, A RESEARCH GROUP AT THE UNIVERSITY OF MINNESOTA.

OBJECTIVE: BUILD THE RECOMMENDATION SYSTEM BASED ON THE MOVIELENS DATASET TO MINIMIZE THE ROOT-MEAN-SQUARE ERROR



MODEL ARCHITECTURE

1. GLOBAL BIAS
2. ITEM-BASED
3. USER-BASED
4. LATENT FACTOR MODEL

FINAL MODEL

ENSEMBLE RECOMMENDATION SYSTEM

DATASET

ratings_df.head()

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

ratings_train.csv

movies_df.head()

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

movies.csv

UTILITY MATRIX

```
movies_df = pd.read_csv(movie_path)
ratings_df = pd.read_csv(ratings_train_path)

# User-item matrix
utility_matrix = ratings_df.pivot_table(index = "userId", columns = "movieId", values = "rating")

missing_columns = list(set(movies_df["movieId"]) - set(utility_matrix.columns))
for col in missing_columns:
    utility_matrix[col] = np.nan

utility_matrix = utility_matrix[sorted(utility_matrix.columns)]

utility_matrix.head()
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

GLOBAL BIAS

```python
def calculate_user_bias(self, user_id):
    user_rating_val = self.utility_matrix.loc[user_id].dropna().values
    user_rating_count = user_rating_val.shape[0]

    if user_rating_count == 0:
        user_mean = 0
    else:
        user_mean = user_rating_val.sum() / user_rating_count

    user_bias = user_mean - self.overall_mean

    return user_bias

def calculate_movie_bias(self, movie_id):
    movie_rating_val = self.utility_matrix.loc[:,movie_id].dropna().values
    movie_rating_count = movie_rating_val.shape[0]
    if movie_rating_count == 0:
        movie_mean = 0
    else:
        movie_mean = movie_rating_val.sum() / movie_rating_count

    movie_bias = movie_mean - self.overall_mean

    return movie_bias
```

REFERENCE: DES431 LECTURE 7 P. 177

The predicted rating of a user x and item y is given by

$$\hat{r}_{bias}(u_x, I_y) = \mu + b_x + b_y,$$

where,

μ is an overall mean rating (all movies, all users),

b_x is a bias of user x,

b_y is a bias of movie y,

RMSE: 0.888971186618481

```python
def predict(self, user_id, movie_id):
    user_bias = self.calculate_user_bias(user_id)
    movie_bias = self.calculate_movie_bias(movie_id)

    # Find the predited rating given userId and movieId
    predicted_rating = self.overall_mean + user_bias + movie_bias

    return predicted_rating
```

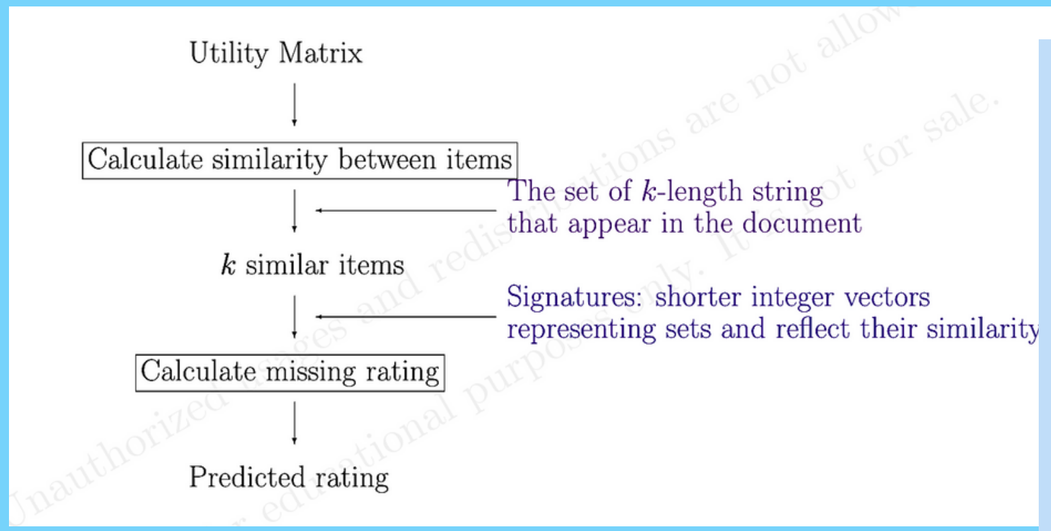
COLLABORATIVE FILTERING

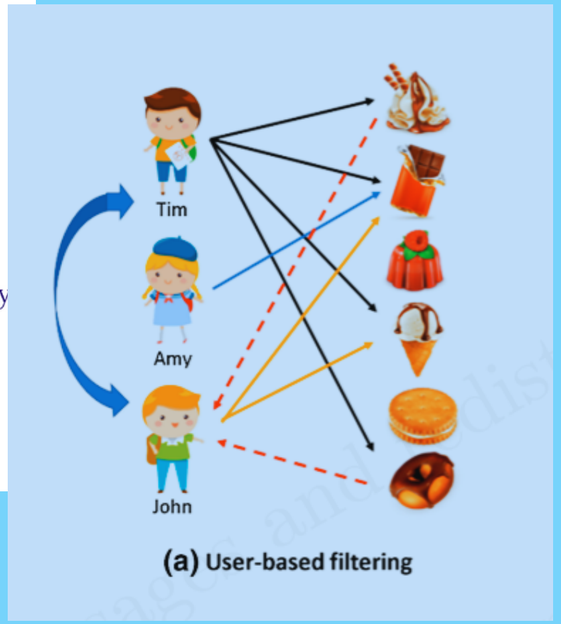
ITEM-BASED & USER-BASED

ITEM-BASED

$$\hat{r}_{item-based}(u_x, I_y) = \frac{\sum_{I_j \in \Omega} sim(I_y, I_j) \times r(u_x, I_j)}{\sum_{I_j \in \Omega} sim(I_y, I_j)},$$

	userId	movieId	predicted_rating
0	4	45	3.646502
1	4	52	3.483849
2	4	58	3.387581
3	4	222	3.891997
4	4	247	3.898270





(a) User-based filtering

```
user_ratings = self.utility_matrix.loc[user_id]
similar_movies = self.item_similarity_matrix[movie_id][user_ratings.notna()]
top_similar_movies = similar_movies.nlargest(self.k+1)[1:]
user_ratings_updated = user_ratings[top_similar_movies.index]
```

```
predicted_rating = (user_ratings_updated * top_similar_movies).sum() / top_similar_movies.sum()
```

RMSE: 0.8468

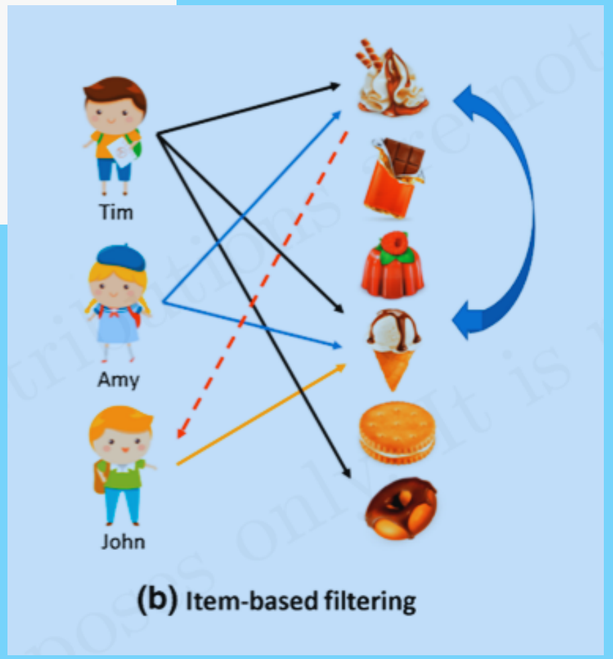
USER-BASED

$$\hat{r}_{user-based}(u_x, I_y) = \bar{r}_x + \frac{\sum_{u_i \in \Phi} sim(u_x, u_i) \times (r(u_i, I_y) - \bar{r}_i)}{\sum_{u_i \in \Phi} sim(u_x, u_i)},$$

```
# Select rating that similar users have given to the target movie
similar_users_rating = target_movie_rating[similar_users_simscore.index]

# Compute average rating of similar user
similar_users_avrrating = ratings_df[ratings_df["userId"].isin(similar_users_simscore.index.tolist())] \
                                    .groupby("userId").mean()["rating"] \
                                    [similar_users_simscore.index]

offset = (similar_users_simscore * (similar_users_rating - similar_users_avrrating)).sum() \
                / similar_users_simscore.sum()

predicted_rating = target_user_avrrating + offset
```

RMSE: 0.8860

	userId	movieId	predicted_rating
0	4	45	3.342061
1	4	52	3.523440
2	4	58	3.650240
3	4	222	3.714852
4	4	247	4.114008



(b) Item-based filtering

LATENT FACTOR MODEL

Matrix Factorization-based algorithms

class surprise.prediction_algorithms.matrix_factorization.SVD(*n_factors=100, n_epochs=20, biased=True, init_mean=0, init_std_dev=0.1, lr_all=0.005, reg_all=0.02, lr_bu=None, lr_bi=None, lr_pu=None, lr_qi=None, reg_bu=None, reg_bi=None, reg_pu=None, reg_qi=None, random_state=None, verbose=False*)

Bases: `AlgoBase`

The famous *SVD* algorithm, as popularized by Simon Funk during the Netflix Prize. When baselines are not used, this is equivalent to Probabilistic Matrix Factorization [SM08] (see note below).

GLOBAL MEAN = SUM OF ALL RATINGS / TOTAL NUMBER OF RATINGS

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

If user u is unknown, then the bias b_u and the factors p_u are assumed to be zero. The same applies for item i with b_i and q_i.

For details, see equation (5) from [KBV09]. See also [RRSK10], section 5.3.1.

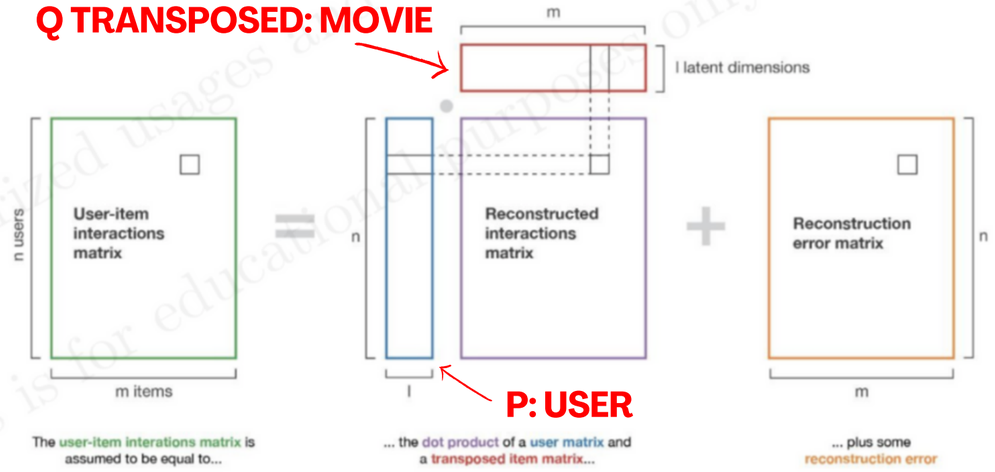
To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$$

```python
from surprise import Dataset, Reader, SVD
from sklearn.metrics import mean_squared_error

# Create a Reader object
reader = Reader(rating_scale=(0.5, 5))

# Load the data into a Surprise Dataset object
ratings_train = ratings_df.copy()
data = Dataset.load_from_df(ratings_train[['userId', 'movieId', 'rating']], reader)
trainset = data.build_full_trainset()

# Define the SVD model and train it on the training set
svdmodel = SVD(n_factors=100, lr_all=0.01, reg_all=1)
svdmodel.fit(trainset)
```

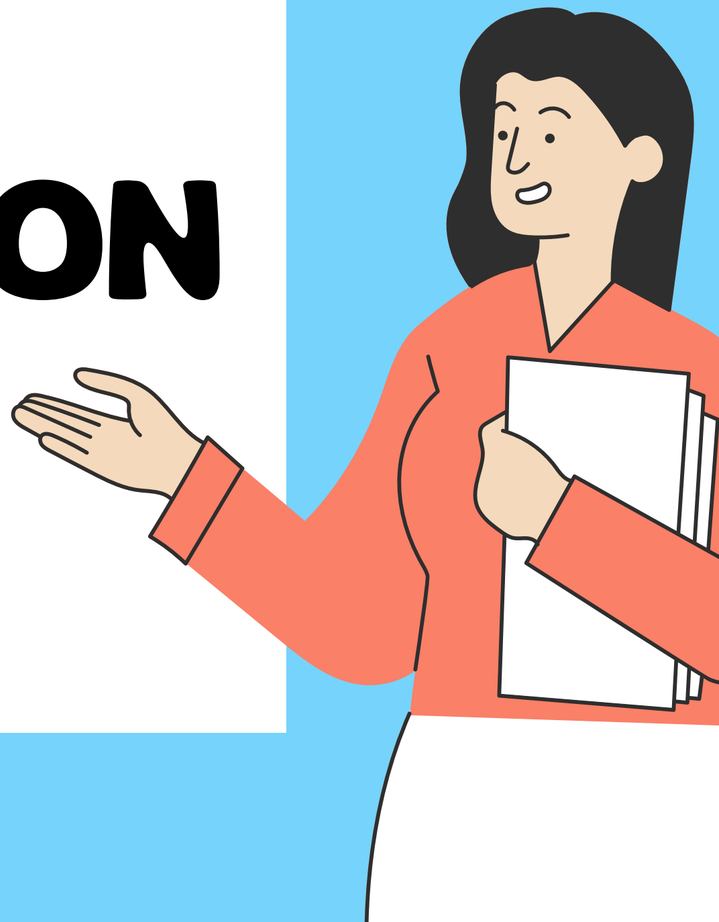
```
RMSE:  0.8697917376963271
```

Q TRANSPOSED: MOVIE



REFERENCE: HTTPS://SURPRISE.READTHEDOCS.IO/ (OFFICIAL SURPROSE DOCUMENTATION WEBSITE + DES431 LECTUER 7 P 167

ENSEMBLE RECOMMENDATION SYSTEM



MODEL ARCHITECTURE

1. GLOBAL BIAS
2. ITEM-BASED
3. USER-BASED
4. LATENT FACTOR MODEL

FINAL MODEL

MODEL PREDICTION

```
def predict_rating(df):
    '''
    Input:
        df = a dataframe with two columns: userId, movieId
    Output:
        a dataframe with three columns: userId, movieId, rating
    '''
```

```
################################ Model Construction ###################################
# Global Bias Model
global_bias_model = GlobalBias(utility_matrix)

# Collaborative Filtering
# Item-based
itembased_cf = ItemBasedCollaborativeFiltering()
itembased_cf.fit(utility_matrix)

# User-based
userbased_model = UserBasedCollaborativeFiltering()
userbased_model.fit(utility_matrix)

# Laten Factor Model
svdmodel = SVD(n_factors=100, lr_all=0.01, reg_all=1)
svdmodel.fit(trainset)
```

MODEL CONSTRUCTION

```
############################## Model Prediction ###################################
# Define list so storing the predicted values
gb_pred = []
itcf_pred = []
ucf_pred = []
svdm_pred = []

# Predict the rating for each pair of userId and movieId
for index, (user_id, movie_id) in enumerate(zip(r["userId"], r["movieId"])):
    # Global Bias
    gb = global_bias_model.predict(user_id, movie_id)
    gb_pred.append(gb)

    # Item-based
    itcf = itembased_cf.predict(user_id, movie_id)
    itcf_pred.append(itcf)

    # User-based
    ucf = userbased_model.predict(user_id, movie_id)
    ucf_pred.append(ucf)

    # Latent Factor Model
    svd = svdmodel.predict(user_id, movie_id).est
    svdm_pred.append(svd)
```

CONSTRUCT FINAL RATING

```
############################## Construct Final Rating ###################################
df["gb_pred"] = gb_pred
df["itcf_pred"] = itcf_pred
df["ucf_pred"] = ucf_pred
df["svdm_pred"] = svdm_pred

# Calculate the weighted rating
df["rating"] = 0.15*df["gb_pred"] + 0.4*df["itcf_pred"] + 0.2*df["ucf_pred"] + 0.25*df["svdm_pred"]

return df[["userId","movieId","rating"]]
```

RATING VALIDATION

```python
from sklearn.metrics import mean_squared_error

r_true = ratings_valid["rating"].to_numpy()
r_pred = r["rating"].to_numpy()

rmse = mean_squared_error(r_true, r_pred, squared=False)
print(f"RMSE = {rmse:.4f}")
```

```
RMSE = 0.8182
```

RESULT


THANK
YOU!

Team Member

1. 6322770064 Pauruetai Kobsahai
2. 6322770114 Thanakrit Loetpricha
3. 6322770692 Chanon Charuchinda
4. 6322772367 Rawikarn Keitiwattanapong