THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP4434 Big Data Analytics

Lab 10 Collaborative Filtering

HUANG Xiao
xiaohuang@comp.polyu.edu.hk

A Dataset for Collaborative Filtering

The last.fm Dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system. We'll focus on two files: **user_artists.dat** — plays counts of artist by user; artists.dat — id, name as they contain all data required to make recommendations for new music artists to a user.

user_artists.dat

userID	artistID	weight
2	51	13883
2	52	11690
…	…	…

artists.dat

id	name
1	MALICE MIZER
2	Diary of Dream
…	…

Loading the Data

Loading the data

```
In [17]: plays = pd.read_csv('data/user_artists.dat', sep='\t')
         artists = pd.read_csv('data/artists.dat', sep='\t', usecols=['id','name'])
         ap = pd.merge(artists, plays,left_on="id",right_on="artistID")
         ap = ap.rename(columns={"weight": "playCount"})
         ap.head()
```

Merge articles and plays

Out[17]:

	id	name	userID	artistID	playCount
0	1	MALICE MIZER	34	1	212
1	1	MALICE MIZER	274	1	483
2	1	MALICE MIZER	785	1	76
3	2	Diary of Dreams	135	2	1021
4	2	Diary of Dreams	257	2	152

Rank the artists based on how much they were played by the users

```
In [18]: artist_rank = ap.groupby(['name']).agg({'userID' : 'count', 'playCount' : 'sum'}) \
           .rename(columns={"userID" : 'totalUniqueUsers', "playCount" : "totalArtistPlays"}) \
           .sort_values(['totalArtistPlays'], ascending=False)

         artist_rank['avgUserPlays'] = artist_rank['totalArtistPlays'] / artist_rank['totalUniqueUsers']
         ap = ap.join(artist_rank, on="name", how="inner").sort_values(['playCount'], ascending=False)
         ap.head()
```

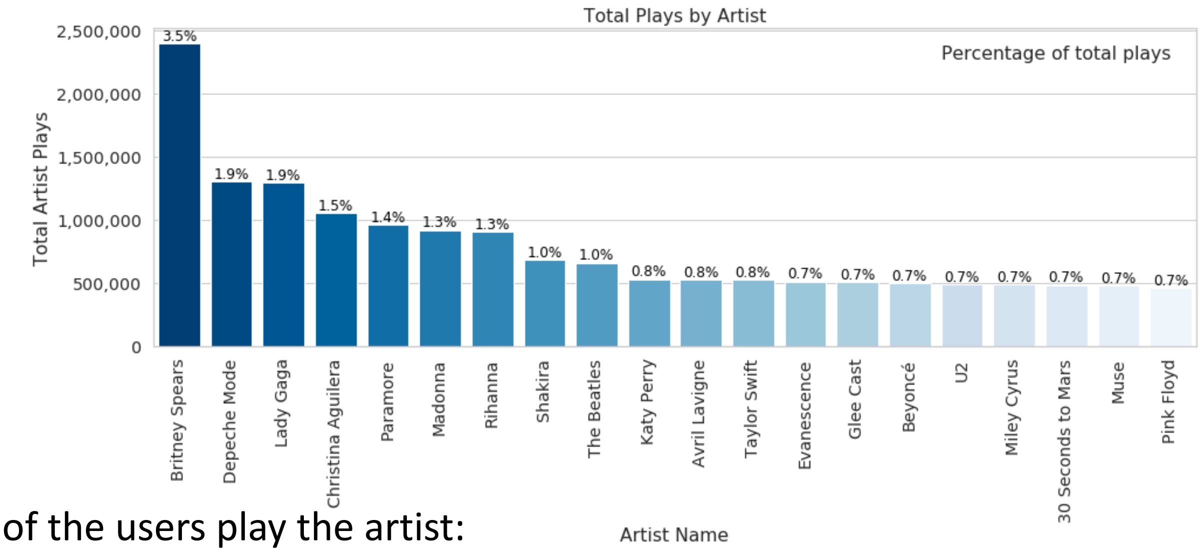
Merge the results with the previous data frame

Out[18]:

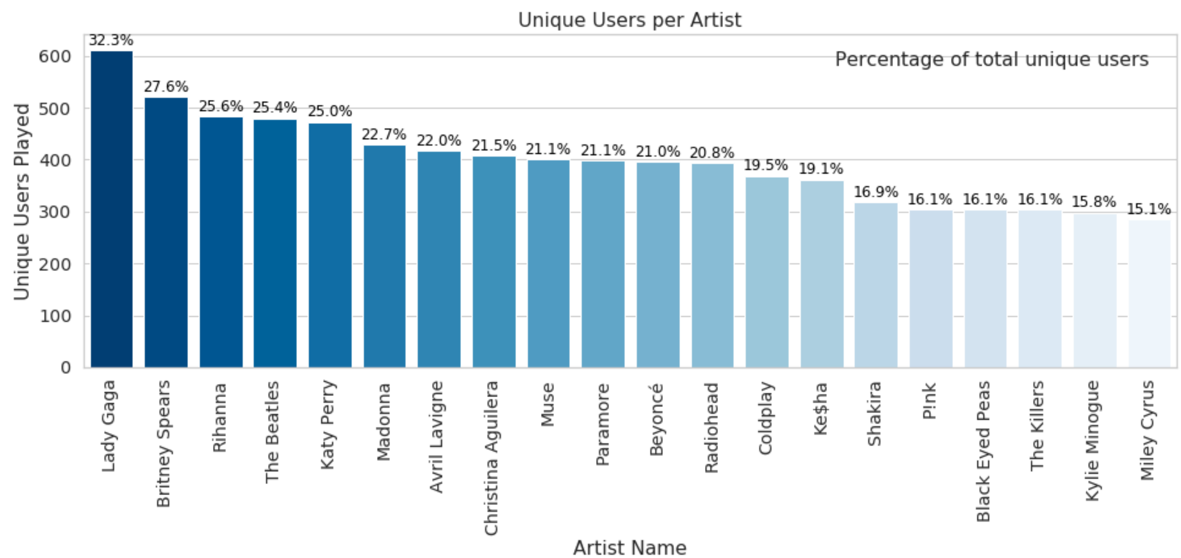
	id	name	userID	artistID	playCount	totalUniqueUsers	totalArtistPlays	avgUserPlays
2800	72	Depeche Mode	1642	72	352698	282	1301308	4614.567376
35843	792	Thalía	2071	792	324663	26	350035	13462.884615
27302	511	U2	1094	511	320725	185	493024	2664.994595
8152	203	Blur	1905	203	257978	114	318221	2791.412281
26670	498	Paramore	1664	498	227829	399	963449	2414.659148

Exploration

The names of the artists that were played most:



How much of the users play the artist:



Preprocessing

Data scaling

```
In [78]: pc = ap.playCount
         play_count_scaled = (pc - pc.min()) / (pc.max() - pc.min())

         ap = ap.assign(playCountScaled=play_count_scaled)

         ratings_df = ap.pivot(index='userID', columns='artistID', values='playCountScaled')
         ratings = ratings_df.fillna(0).values

         sparsity = float(len(ratings.nonzero()[0]))
         sparsity /= (ratings.shape[0] * ratings.shape[1])
         sparsity *= 100
         print('{:.2f}%'.format(sparsity))

0.28%
```

Squish the play counts in the [0,1] range and add a new column

```
In [77]: train, val = train_test_split(ratings)
         train.shape

Out[77]: (1892, 17632)
```

Directly Learn latent factors

$$\mathbf{R} \approx Q \cdot P^T$$

R (rows: items; columns: users)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

Q (rows: items; columns: factors)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

P^T (rows: factors; columns: users)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

- Ratings can be recovered by latent factors (low-dimensional features)

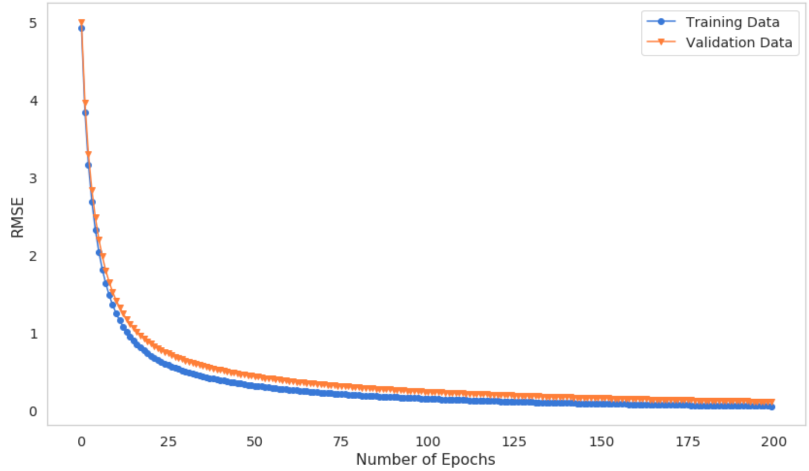
$$h_{i,j}(x,\theta) = \left(\theta^{(j)}\right)^T x^{(i)} = \theta_0^{(j)} x_0^{(i)} + \theta_1^{(j)} x_1^{(i)} + \theta_2^{(j)} x_2^{(i)} + \theta_3^{(j)} x_3^{(i)}$$

Training by SGD

```
def fit(self, X_train, X_val):
  m, n = X_train.shape

  self.P = 3 * np.random.rand(self.n_latent_features, m)
  self.Q = 3 * np.random.rand(self.n_latent_features, n)

  self.train_error = []
  self.val_error = []

  users, items = X_train.nonzero()

  for epoch in range(self.n_epochs):
      for u, i in zip(users, items):
          error = X_train[u, i] - self.predictions(self.P[:,u], self.Q[:,i])
          self.P[:, u] += self.learning_rate * (error * self.Q[:, i] - self.lmbda * self.P[:, u])
          self.Q[:, i] += self.learning_rate * (error * self.P[:, u] - self.lmbda * self.Q[:, i])

      train_rmse = rmse(self.predictions(self.P, self.Q), X_train)
      val_rmse = rmse(self.predictions(self.P, self.Q), X_val)
      self.train_error.append(train_rmse)
      self.val_error.append(val_rmse)

  return self
```

CF Gradient Decent Update

```
def predict(self, X_train, user_index):
  y_hat = self.predictions(self.P, self.Q)
  predictions_index = np.where(X_train[user_index, :] == 0)[0]
  return y_hat[user_index, predictions_index].flatten()
```



Making Recommendations

```
In [91]: user_id = 1236
         user_index = ratings_df.index.get_loc(user_id)
         predictions_index = np.where(train[user_index, :] == 0)[0]

         rating_predictions = recommender.predict(train, user_index)
```

Make recommendations for user 1236

```
In [94]: create_artist_ratings(artists, predictions_index, rating_predictions)
```

Out[94]:

	id	name	rating
0	5014	Towers of London	0.506186
1	11792	Burn Down Rome	0.499209
2	12380	Symphony X - "V" The New Mitology Suite	0.495009
3	12815	auncia	0.493578
4	13827	Thavius Beck	0.491191
5	13843	Rock Kills Kid	0.482907
6	15312	Aphex Twin, Drukqs	0.475207
7	16136	Jason Anderson	0.474715
8	17010	Neil Patrick Harris	0.472820
9	17737	Chá de Abu	0.472218

Recommendation list

Further Practice

Further tasks:

- Implement Content based recommender system using project dataset
- Implement Collaborative Filtering based recommended system using project dataset

Further readings:

- https://realpython.com/build-recommendation-engine-collaborative-filtering/
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf#:~:text=which%20it%20occurs.-,Definition,document%20or%20a%20web%20page.
- https://www.kdnuggets.com/2019/09/machine-learning-recommender-systems.html
- https://github.com/grahamjenson/list_of_recommender_systems
- An academic Survey