Graded Activity - Case Study 4.1

Build your own Recommendation System for Movies

Instructions

1. Save this template on your computer and change the name of the file as follows: YourLastName_FirstName_CS_4

*Note: You will not be able to retrieve this document from the edX platform after you submit it. Please, save this document in a central location for future reference.

2. This is an individual activity. A scoring rubric can be downloaded from the CS4.1 section on the wiki. This rubric will help you to successfully complete your activity and grade others.

3. Read the CS self-help documentation and follow the steps and answer the questions on this activity sheet.

3. If you have any questions, feel free to ask the TAs in the forum space created for this activity.

4. Upload your document as a .pdf or as a .ppt / .pptx

*Note: edX has a **10MB** file size limit for document submission. If you have used large image(s), you may need to resize before submitting, OR you may simply include a web URL for the image in the image location on this activity sheet. Be sure to submit your assignment at least one hour before the deadline to provide time for troubleshooting.

5. The edX system will automatically assign you a Case Study from one of your peers. Follow the rubric and assess your peer.

Case Study 4.1

- **1. What programming language did you use?**

Python

- **2.1 What does your data look like? Share a screenshot of your plotted data.**

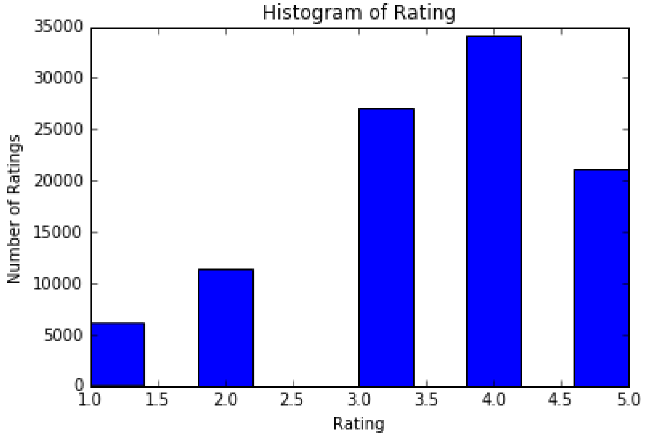

Figure 1

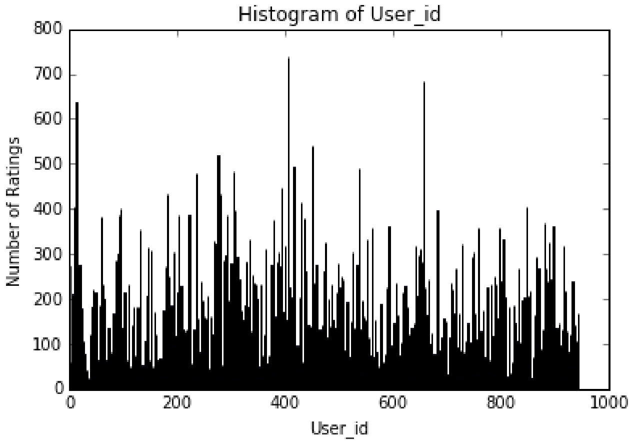

Figure 2

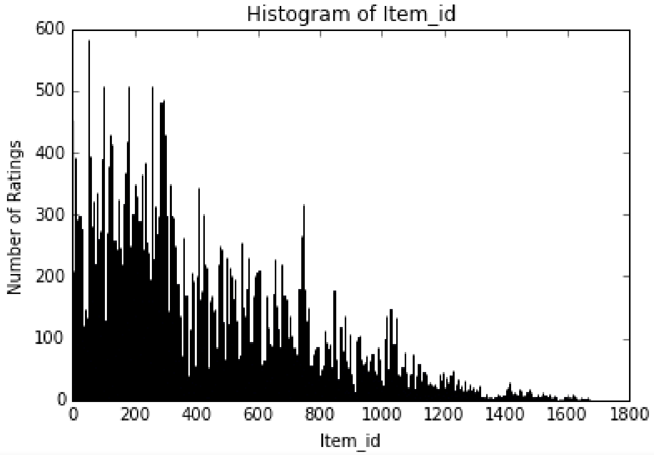

Figure 3

Comments:

- Figure 1: a larger number of ratings with ratings of 3, 4, and 5 rather than rating of 1 and 2
- Figure 2: certain users (user_id) have number of rating as low as about 10, and as high as 700. A large amount of users has number of ratings from 50 to 200
- Figure 3: Movies with Item_id from 0-400 have more rating than the rest.

Case Study 4.1

Reducing Sparsity…

- **2.2 What criteria did you use for subsetting data?**

not include a user if they have fewer than 50 ratings. Increasing sparity from 6.30% to 9.27% (more ratings filling up the rating matrix after cutting off the dataset)

Code for cutting off the dataset

```
users = data["user_id"]
ratings_count = {}
for user in users:
    if user in ratings_count:
        ratings_count[user] += 1
    else:
        ratings_count[user] = 1
RATINGS_CUTOFF = 50 # A user is not included if they have fewer than "RATINGS_CUTOFF" ratings
remove_users = []
for user,num_ratings in ratings_count.iteritems():
    if num_ratings < RATINGS_CUTOFF:
        remove_users.append(user)
data = data.loc[~data['user_id'].isin(remove_users)]
```

Code for calculating sparsity

```
import numpy as np

Number_Ratings = float(len(data))
Number_Movies = float(len(np.unique(data["item_id"])))
Number_Users = float(len(np.unique(data["user_id"])))
Sparsity = (Number_Ratings/(Number_Movies*Number_Users))*100.0
```

- **3. (Optional) Share the academic literature you consulted for implementing this Case Study (if any).**

Click here to add text

Case Study 4.1

- **4. Share the code you used for creating a data split**

```
#Spitting (% of the dataset): 70% for training and 30% for testing

import graphlab #import Graphlab

sf = graphlab.SFrame(data)  #Load dataset "data" with a graphlab Sframe
sf_train, sf_test = sf.random_split(.7)  #split dataset "data": 70% for train & 30% for test
```

Case Study 4.1

5. Share a screenshot of your popularity recommender code

Code

```
#use training set (70% of dataset) to create the model
#the model computes the mean rating for each item and uses this to rank items for recommendations
popularity_recommender = graphlab.recommender.popularity_recommender.create(sf_train,target='rating')

#use test set (30% of dataset) to test the model
#by evaluating prediction error for each user-item pair in the given data set
popularity_recommender.evaluate_rmse(sf_test,'rating')
```

Output: showing the average RMSE for some movies (item_id) in Fig. a, and some users (user_id) in Fig. b.
Note that, count for user_id might be smaller than RATINGS_CUTOFF (=40) , which is specified in Q. 2.2 and applicable for dataset.
Since the results are produced from test set, which is 30% of the data, count for user_id might be smaller than RATINGS_CUTOFF.

```
Recsys training: model = popularity

Preparing data set.

    Data has 62172 observations with 568 users and 1632 items.

    Data prepared in: 0.094077s

62172 observations to process; with 1632 unique items.

'rmse_overall': 1.0241501867105367
```

item_id	count	rmse
118	75	1.16696698538
1029	3	1.03386194601
435	57	0.984195972784
1517	3	1.29099444874
537	7	1.27915631796
526	40	0.893828439396
232	28	0.929062872017
310	19	0.821564830298
49	29	1.0388903263
13	55	1.06583878002

[1476 rows x 3 columns]

Figure a

user_id	count	rmse
118	19	1.14012872956
435	120	0.875786996556
537	151	1.02859990121
526	20	1.17668927106
232	29	0.948251205943
49	65	1.33945136091
13	194	1.27344366113
363	89	1.17415296549
60	74	0.694181258021
738	53	0.663889198481

[568 rows x 3 columns]

Figure b

Case Study 4.1

6. Collaborative Filtering - Validation set

- **6.1 Paste the Train/Validation split code you used for 75% - 25%**

```
#Spitting (% of the dataset): (0.75*70%) 52.5% for training, (0.25*70%) 17.5% for validating, and 30% for testing

import graphlab #import Graphlab

#Load dataset "data" with a graphlab Sframe
sf = graphlab.SFrame(data)

#split dataset "data": 70% for train & 30% for test
sf_train, sf_test = sf.random_split(.7)

#further split train set: 75% for training & 25% for validating
sf_train, sf_validate = sf_train.random_split(.75)
```

- **6.2 What values did you use for training the different models?**

Different Regularization terms are used for training the different models: 1e-5, 1e-4, 1e-3, 1e-2, and 1e-1

Case Study 4.1

MIT xPRO

6. Collaborative Filtering - Validation set

- **6.3 Which model resulted in the lowest RMSE?**

The model with regularization_term=0.001

- **6.3 How did you determine the best model?**

Based on the validation RMSE achieved. Using train data set with each regularization_term creates a model, and then running the model with validate data set to find RMSE. Choose the model with a regularization_term which give the lowest RMSE value on the validate data set

- **Share a screenshot of your code with the best parameters**

```
sf_train, sf_validate = sf_train.random_split(.75) #further split train set: 75% for training & 25% for validating

#different Regularization terms are used for training the different models
#assign regularization_terms to find an optimal value for the dataset
regularization_terms = [10**-5,10**-4,10**-3,10**-2,10**-1]

#initialize best_regularization_term and best_RMSE
best_regularization_term=0
best_RMSE = np.inf

for regularization_term in regularization_terms:
    #create a model with regularization_term and the train set
    factorization_recommender = graphlab.recommender.factorization_recommender.create(sf_train,
                                                                                      target='rating',
                                                                                      regularization=regularization_term)

    #eveluate the model with the validate set
    evaluation = factorization_recommender.evaluate_rmse(sf_validate,'rating')

    #update best_RMSE and best_regularization_term if overal RMSE on the validate data set is lower than the previous one
    if evaluation['rmse_overall'] < best_RMSE:
        best_RMSE = evaluation['rmse_overall']
        best_regularization_term = regularization_term

print "Best Regularization Term", best_regularization_term
print "Best Validation RMSE Achieved", best_RMSE
```

Best model

```
Best Regularization Term 0.001
Best Validation RMSE Achieved 0.931788174092
```

```
#running the test data set witht the best model
factorization_recommender = graphlab.recommender.factorization_recommender.create(sf_train,
                                                                                  target='rating',
                                                                                  regularization=best_regularization_term)
print "Test RMSE on best model", factorization_recommender.evaluate_rmse(sf_test,'rating')['rmse_overall']
```

```
Test RMSE on best model 0.943706616837
```

Case Study 4.1

- **7. Describe how you did the Item Similarity Filtering**

Code

```
sf = graphlab.SFrame(data) #Load dataset "data" with a graphlab SFrame
#redefine the dataset: 70% for training and 30% for test
sf_train_item, sf_test_item = sf.random_split(.7)

item_similarity_recommender = graphlab.recommender.item_similarity_recommender.create(sf_train_item,target='rating')
print "Test RMSE on model", item_similarity_recommender.evaluate_rmse(sf_test_item,'rating')['rmse_overall']
```

Result

```
Test RMSE on model 3.6597122327
```

RMSE in similarity filtering is much higher than popularity recommender and collaborative filtering

- **8. Share a screenshot of the code for how you got the top K recommendations for one of the models**

```
k=5

#top k recommendations from Popularity Recommender
popularity_top_k = popularity_recommender.recommend(k=k)

#top k recommendations from the Collaborative Filtering model
factorization_top_k = factorization_recommender.recommend(k=k)

#top k recommendations from Item-Item Similarity Recommender
item_similarity_top_k = item_similarity_recommender.recommend(k=k)

#print the Collaborative Filtering model
print item_similarity_top_k
```

Result Top 5 recommendations of the Collaborative Filtering model

user_id	item_id	score	rank
244	178	4.69396442393	1
244	483	4.69184857825	2
244	114	4.66390985468	3
244	408	4.64419347266	4
244	318	4.63560754279	5
298	64	4.69501933348	1
298	114	4.6440998007	2
298	169	4.6380979706	3
298	408	4.62438341867	4
298	12	4.5827265669	5

```
[2840 rows x 4 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.
```

Case Study 4.1

- **10. Share your precision/recall, confusion Matrix**

Code

```
models = [popularity_recommender,factorization_recommender,item_similarity_recommender]
model_names = ['popularity_recommender','factorization_recommender','item_similarity_recommender']
precision_recall = graphlab.recommender.util.compare_models(sf_test,models,metric='precision_recall',model_names=model_names)
```

PROGRESS: Evaluate model popularity_recommender

Precision and recall summary statistics by cutoff

cutoff	mean_precision	mean_recall
1	0.0	0.0
2	0.0	0.0
3	0.000586854460094	0.000125754527163
4	0.00044014084507	0.000125754527163
5	0.000352112676056	0.000125754527163
6	0.000293427230047	0.000125754527163
7	0.000503018108652	0.000133981458847
8	0.000660211267606	0.000160258524224
9	0.000782472613459	0.000168485455907
10	0.00105633802817	0.00020315842176

PROGRESS: Evaluate model factorization_recommender

Precision and recall summary statistics by cutoff

cutoff	mean_precision	mean_recall
1	0.161971830986	0.00340268857593
2	0.12764084507	0.00533711835484
3	0.118544600939	0.00761945366216
4	0.103433098592	0.00876494784352
5	0.101056338028	0.0108703857913
6	0.106807511737	0.0135953573377
7	0.11569416499	0.0178900108791
8	0.116197183099	0.0207420083033
9	0.113654147105	0.0228071895557
10	0.111443661972	0.024663728299

PROGRESS: Evaluate model item_similarity_recommender

Precision and recall summary statistics by cutoff

cutoff	mean_precision	mean_recall
1	0.573943661972	0.0156045294534
2	0.543133802817	0.0289464127846
3	0.533450704225	0.0430832902986
4	0.516285211268	0.0550399361212
5	0.508450704225	0.0672602884222
6	0.49882629108	0.0785304620578
7	0.484406438632	0.0878912603022
8	0.473591549296	0.0977090443051
9	0.466549295775	0.10764380375
10	0.461795774648	0.118284684131

- **Any comments?**

Look at top recommendations from each model, item similarity recommender provides highest number of recommendations which belong to true top selections from users; while popularity recommender provides less number of recommendations which belong to true top selections by users.