Baseline_knn_nmf_bias

December 19, 2019

[4]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from surprise import Dataset
from surprise import accuracy
from surprise import Reader
from surprise.model_selection import KFold
from surprise.model_selection import cross_validate
import time
import sklearn
from sklearn.model_selection import train_test_split
from random import shuffle
from surprise.prediction_algorithms import NMF
from surprise.prediction_algorithms import KNNWithMeans
import seaborn as sns
```

0.0.1 Read train and test datast

[5]:
```
train_ratings = pd.read_csv("train_final.csv")
test_ratings = pd.read_csv("test_final.csv")
```

[6]:
```
print('train_minimum_rating',min(train_ratings['rating']))
print('train_maximum_rating',max(train_ratings['rating']))
print('test_minimum_rating',min(train_ratings['rating']))
print('test_maximum_rating',max(train_ratings['rating']))
```

```
train_minimum_rating 1
train_maximum_rating 5
test_minimum_rating 1
test_maximum_rating 5
```

[7]:
```
reader = Reader(rating_scale=(1.0, 5.0))
ratings = Dataset.load_from_df(train_ratings, reader)
```

[8]:
```
raw_ratings = ratings.raw_ratings
random.seed(42)
```

```
shuffle(raw_ratings)
```

[9]:
```
ratings.raw_ratings = raw_ratings
```

[10]:
```
copy_ratings=ratings
```

First we run a 5-fold cv on the train set to see some preliminary results

[14]:
```
method = KNNWithMeans()
cross_val = cross_validate(method, ratings, measures=['RMSE', 'MAE'], cv=5,␣
 ↪verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3562  1.3540  1.3591  1.3677  1.3668  1.3608  0.0055
MAE (testset)     1.0358  1.0356  1.0393  1.0434  1.0456  1.0399  0.0040
Fit time          10.48   11.25   10.56   9.68    9.71    10.34   0.59
Test time         2.58    2.33    2.46    2.13    2.01    2.30    0.21
```

0.0.2 KNN-Grid_search: cross-validation on similarity measure

[15]:
```
def try_similarity(options):
    method = KNNWithMeans(sim_option=options)
    cross_val = cross_validate(method, ratings, measures=['RMSE', 'MAE'],␣
 ↪return_train_measures=True,cv=5, verbose=False)

    train_mae=np.mean(cross_val['train_mae'])
    train_rmse=np.mean(cross_val['train_rmse'])

    val_mae=np.mean(cross_val['test_mae'])
    val_rmse=np.mean(cross_val['test_rmse'])


    return train_rmse,train_mae,val_rmse,val_mae
```

[16]:
```
train_rmse_list2=[]
train_mae_list2=[]
```

```
val_rmse_list2=[]
val_mae_list2=[]



similarity_measure = {
    'cosine': {
         'name': 'cosine',
         'user_based': False
         },
    'pearson_baseline': {
         'name': 'pearson_baseline',
         'user_based': False
         },
    'pearson': {
         'name': 'pearson',
         'user_based': False
         }
         }

_min=float('inf')

for k, v in similarity_measure.items():


    print('similarity measure is', k)


    train_rmse,train_mae,val_rmse,val_mae=try_similarity(v)
    train_rmse_list2.append(train_rmse)
    train_mae_list2.append(train_mae)

    val_rmse_list2.append(val_rmse)
    val_mae_list2.append(val_mae)
    if val_rmse<_min:
        _min=val_rmse
        _similarity_measure=v




print("similarity  measure that produce the smallest rmse␣
 ↪are",_similarity_measure)
```

```
similarity measure is cosine
Computing the msd similarity matrix...
```

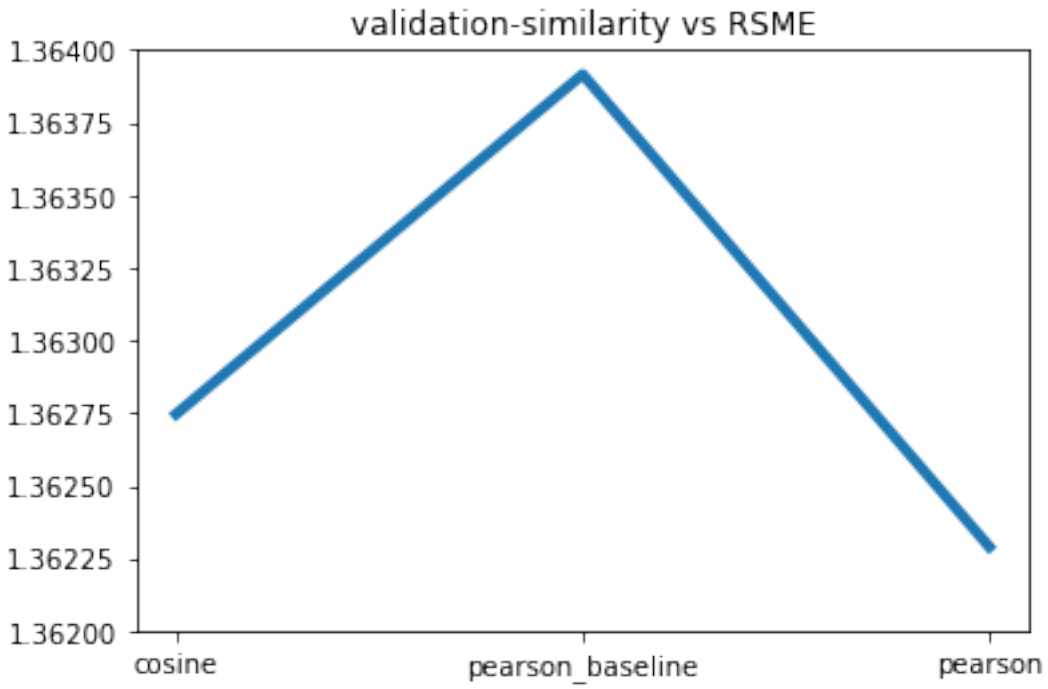
```
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
similarity measure is pearson_baseline
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
similarity measure is pearson
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
similarity  measure that produce the smallest rmse are {'name': 'pearson',
'user_based': False}
```

[25]:
```
val_rmse_list2
```

[25]:
```
[1.3627465682522808, 1.3639160079259403, 1.3622895245161495]
```

[40]:
```
measure_list=['cosine', 'pearson_baseline', 'pearson']
lines = plt.plot(measure_list,val_rmse_list2)
plt.setp(lines[0], linewidth=4)
#plt.setp(lines[1], linewidth=2)
#plt.setp(lines[2], markersize=10)
plt.ylim(1.362, 1.364)
# plt.legend(('train_mae', 'test_mae'),
#            loc='upper right')
plt.title('validation-similarity vs RSME')
plt.savefig('validation-similarity vs RSME.jpg')
```

```
plt.show()
```



0.0.3 KNN-Grid_search: cross-validation on neighborhood size

[21]:
```
def try_neighbor(kNeighbor,_similarity_measure):
    method = KNNWithMeans(min_k=kNeighbor,sim_option=_similarity_measure)
    cross_val = cross_validate(method, ratings, measures=['RMSE', 'MAE'],␣
 ↪return_train_measures=True,cv=5, verbose=False)

    train_mae=np.mean(cross_val['train_mae'])
    train_rmse=np.mean(cross_val['train_rmse'])

    val_mae=np.mean(cross_val['test_mae'])
    val_rmse=np.mean(cross_val['test_rmse'])


    return train_rmse,train_mae,val_rmse,val_mae
```

[29]:
```
train_rmse_list=[]
train_mae_list=[]
val_rmse_list=[]
val_mae_list=[]
```

```
neighbor_num = np.arange(5, 50 , 10)

_min=float('inf')
for num in neighbor_num:


    print('number of neighbor', num)



    train_rmse,train_mae,val_rmse,val_mae=try_neighbor(num,{'name':␣
 ↪'pearson_baseline', 'user_based': False})
    train_rmse_list.append(train_rmse)
    train_mae_list.append(train_mae)

    val_rmse_list.append(val_rmse)
    val_mae_list.append(val_mae)
    if val_rmse<_min:
        _min=val_rmse
        _num=num







print("number of neighbor that produce the smallest rmse are", _num)
```

```
number of neighbor 5
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
number of neighbor 15
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
number of neighbor 25
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
number of neighbor 35
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
number of neighbor 45
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
number of neighbor that produce the smallest rmse are 15
```

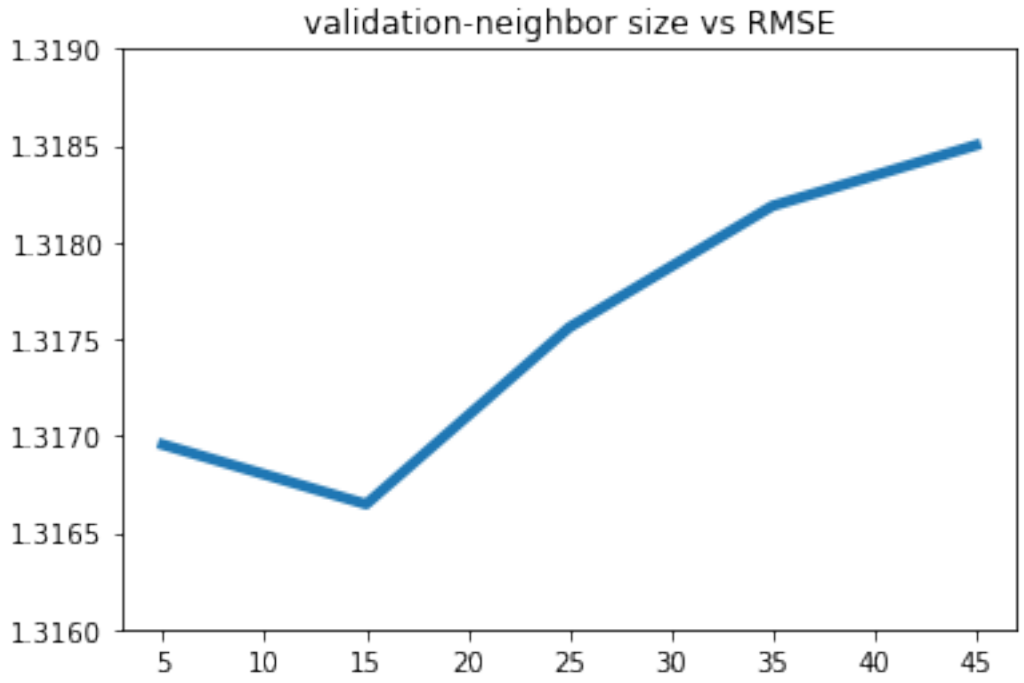
[34]:
```
val_rmse_list
```

[34]:
```
[1.3169542628110793,
 1.3166467226003291,
 1.3175601883527568,
 1.3181877412237253,
 1.3185010364667882]
```

[39]:
```
lines = plt.plot(neighbor_num,val_rmse_list)
plt.setp(lines[0], linewidth=4)

plt.ylim(1.316, 1.319)

plt.title('validation-neighbor size vs RMSE')
plt.savefig('validation-neighborsize_vs_RMSE.jpg')
plt.show()
```



0.0.4 NMF-Grid_seach:

[11]:
```
train_ratings = pd.read_csv("train_final.csv")
test_ratings = pd.read_csv("test_final.csv")
reader = Reader(rating_scale=(0.5, 5))
ratings = Dataset.load_from_df(train_ratings, reader)
raw_ratings = ratings.raw_ratings
random.seed(42)
shuffle(raw_ratings)
ratings.raw_ratings = raw_ratings

train_set = ratings.build_full_trainset()
reader = Reader(rating_scale=(0.5, 5))
test_rates = Dataset.load_from_df(test_ratings, reader)
test=test_rates.raw_ratings
```

```
test_set = ratings.construct_testset(test)
```

[ ]:
```
from surprise.model_selection.search import GridSearchCV


param_grid = {'k': [20, 25, 30, 35, 40],

              'sim_options': {'name': ['pearson_baseline','pearson', 'cosine'],
                              'user_based': [False]}
              }
gridCV3 = GridSearchCV(KNNWithMeans, param_grid, measures=['rmse'], refit=True,␣
 ↪cv=5, n_jobs=-1)
gridCV3.fit(ratings)
```

[116]:
```
from surprise.model_selection.search import GridSearchCV
params = {
     'n_factors':[10, 20, 30, 40, 50],
     'n_epochs':[10, 50, 100],
     'reg_pu':[0.01, 0.05, 0.1],
     'reg_qi':[0.01, 0.05, 0.1]
}
gridCV2 = GridSearchCV(NMF, param_grid=params, measures=['rmse', 'mae'],␣
 ↪refit=True, cv=5, n_jobs=-1)
gridCV2.fit(ratings)
```

[117]:
```
gridCV2.best_params['rmse']
```

[117]:
```
{'n_factors': 50, 'n_epochs': 50, 'reg_pu': 0.1, 'reg_qi': 0.05}
```

[38]:
```
import sys

import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objs as go
pyo.init_notebook_mode()
fig2 = px.parallel_coordinates(results_df2,
                               color="mean_test_rmse",
                               dimensions=['param_n_factors',"param_n_epochs",␣
 ↪'param_reg_pu', 'param_reg_qi',
                                           'mean_test_rmse', 'mean_test_mae',␣
 ↪'mean_fit_time'],
                              labels={"param_n_factors": "Rank",
                                      "param_n_epochs": "Iteration",
                                      "param_reg_pu": "Pu",
                                      "param_reg_qi": "Qi",
                                      "mean_test_rmse": "Avg RMSE",
                                      "mean_test_mae": "Avg MAE",
                                      'mean_fit_time': "Avg Fitting Time"},
```

```
                                color_continuous_scale=px.colors.diverging.
↪Tealrose)
fig2.show()
```

0.0.5 evaluate on test set with the parameters we searched for KNN

test set with three ratings for each user

[80]:
```
method = KNNWithMeans(min_k=5,sim_option={
         'name': 'person',
         'user_based': False
         })
method.fit(train_set)
test_result = method.test(test_set)
test_rmse = accuracy.rmse(test_result)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.4981
```

test set with only the last rating for each user

[83]:
```
test_ratings2 = pd.read_csv("test_final2.csv")
reader = Reader(rating_scale=(0.5, 5))
test_rates2 = Dataset.load_from_df(test_ratings2, reader)
test2=test_rates2.raw_ratings
test_set2 = ratings.construct_testset(test2)
method = KNNWithMeans(min_k=5,sim_option={
         'name': 'pearson',
         'user_based': False
         })
method.fit(train_set)
test_result2 = method.test(test_set2)
test_rmse2 = accuracy.rmse(test_result2)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 1.5203
```

User coverage for KNN

[21]:
```
correct = 0
for user in predictions.uid.unique():
    real = np.argsort(predictions.loc[predictions['uid'] == user]['r_ui'])
    pred = np.argsort(predictions.loc[predictions['uid'] == user]['est'])
    correct += (list(real) == list(pred))
print(correct / len(predictions.uid.unique()))
```

```
0.30594556237011505
```

0.0.6 evaluate on test set with the parameters we searched for NMF

test set with three ratings for each user

[107]:
```
method = NMF(n_factors=50,n_epochs=50,reg_pu=0.1,reg_qi=0.05)
method.fit(train_set)
test_result = method.test(test_set)
test_rmse = accuracy.rmse(test_result)
```

```
RMSE: 1.5031
```

test set with only the last rating for each user

[85]:
```
method = NMF(n_factors=50,n_epochs=50,reg_pu=0.1,reg_qi=0.05)
method.fit(train_set)
test_result2 = method.test(test_set2)
test_rmse2 = accuracy.rmse(test_result2)
```

```
RMSE: 1.5311
```

User coverage for NMF

[108]:
```
predictions = pd.DataFrame(test_result)
correct = 0
for user in predictions.uid.unique():
    real = np.argsort(predictions.loc[predictions['uid'] == user]['r_ui'])
    pred = np.argsort(predictions.loc[predictions['uid'] == user]['est'])
    correct += (list(real) == list(pred))
print(correct / len(predictions.uid.unique()))
```

```
0.23118252318921384
```

0.0.7 evaluate on test set for baseline bias model

test set with three ratings for each user

[105]:
```
from surprise.prediction_algorithms.baseline_only import BaselineOnly
train_ratings = pd.read_csv("train_final.csv")
test_ratings = pd.read_csv("test_final.csv")
reader = Reader(rating_scale=(0.5, 5))
ratings = Dataset.load_from_df(train_ratings, reader)
raw_ratings = ratings.raw_ratings
random.seed(42)
shuffle(raw_ratings)
ratings.raw_ratings = raw_ratings
copy_ratings=ratings
test_rating = Dataset.load_from_df(test_ratings, reader)
test_raw_rating=test_rating.raw_ratings

print('Using SGD')
```

```
bsl_options = {'method': 'sgd',
               'learning_rate': .00005,
               }
algo = BaselineOnly(bsl_options=bsl_options)

algo.fit(ratings.build_full_trainset())
test_set = ratings.construct_testset(test_raw_rating)
test_result4 = algo.test(test_set)
test_rmse = accuracy.rmse(test_result)
```

```
Using SGD
Estimating biases using sgd...
RMSE: 1.4782
```

test set with only the last rating for each user

[103]:
```
test_ratings2 = pd.read_csv("test_final2.csv")
test_rating2 = Dataset.load_from_df(test_ratings2, reader)
test_raw_rating2=test_rating2.raw_ratings
print('Using SGD')
bsl_options = {'method': 'sgd',
               'learning_rate': .00005,
               }
algo = BaselineOnly(bsl_options=bsl_options)

algo.fit(ratings.build_full_trainset())
test_set2 = ratings.construct_testset(test_raw_rating2)
test_result3 = algo.test(test_set2)
test_rmse2 = accuracy.rmse(test_result3)
```

```
Using SGD
Estimating biases using sgd...
RMSE: 1.5070
```

User coverage for baseline bias model

[106]:
```
predictions2 = pd.DataFrame(test_result4)
correct = 0
for user in predictions2.uid.unique():
    real = np.argsort(predictions2.loc[predictions2['uid'] == user]['r_ui'])
    pred = np.argsort(predictions2.loc[predictions2['uid'] == user]['est'])
    correct += (list(real) == list(pred))
print(correct / len(predictions2.uid.unique()))
```

```
0.20573774646459528
```