In [3]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from surprise import Dataset
from surprise import NMF
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
DATA_PATH = "gs://zw2624-bucket/input/subsample_data_3.csv"
```

load data

In [38]:
```
ratings = pd.read_csv(DATA_PATH)
ratings.reset_index(drop=True, inplace=True)
```

In [6]:
```
print('minimum_rating',min(ratings['rating']))
print('maximum_rating',max(ratings['rating']))
```

```
minimum_rating 0.5
maximum_rating 5.0
```

Test-Train Split

In [7]:
```
train, test= train_test_split(ratings, test_size=0.2, random_state=42)
reader = Reader(rating_scale=(0.5, 5))
train_set = Dataset.load_from_df(train, reader)
test_set = Dataset.load_from_df(test, reader)
```

Hyper-parameter Tuning + Cross Validation

In [ ]:
```
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_jo
bs=-1)
gridCV2.fit(train_set)
```

best params

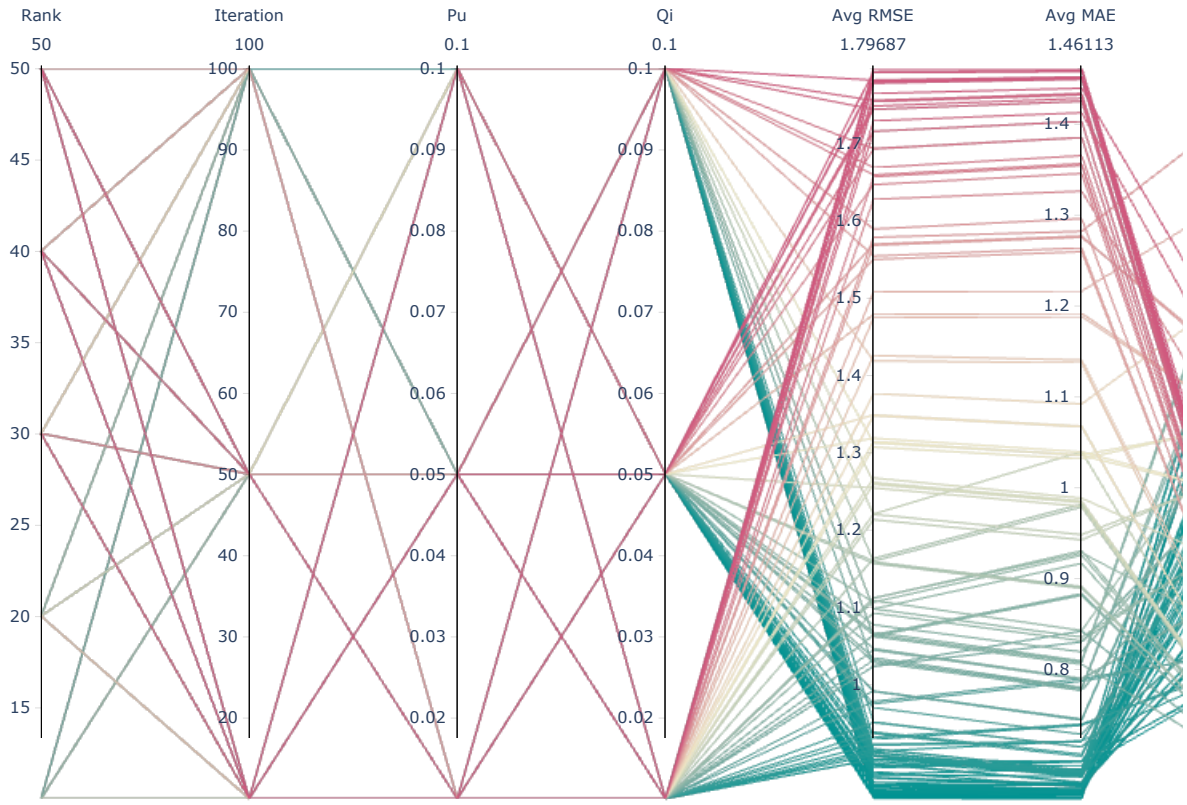
In [122]:
```
gridCV2.best_params['rmse']
```

Out[122]:
```
{'n_factors': 50, 'n_epochs': 100, 'reg_pu': 0.1, 'reg_qi': 0.1}
```

CV result visualization

In [89]:
```
results_df2 = pd.DataFrame.from_dict(gridCV2.cv_results)
```

In [120]:
```
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()
```



Test Result & Evaluation

In [101]:
```
pred = gridCV2.test(test_set.df.to_numpy())
```

In [121]:
```
# helper function for coverage
def coverage(threshold1, threshold2,prediction):
    start_time=time.time()
    predictions = pd.DataFrame(prediction)
    pred = predictions.groupby('uid')
    df1= pred.apply(lambda x: x.sort_values(by=["est"],ascending=False))
    df2=df1.reset_index(drop=True)
    df3 = df2.groupby('uid').head(10)

    s1=df3[df3['r_ui'] > threshold1].groupby('uid')['r_ui'].count().reset_index()
    s2 = df3.pivot_table(index=['uid'],aggfunc='size').reset_index()
    s2.columns = ['uid','counts']

    df=pd.merge(s1, s2, on='uid')

    # #number of high true rating(larger than 4) devided by top N predictions
    df['rate']=df['r_ui']/df['counts']

    user_coverage=float(sum(df['rate']> threshold2))/df3['uid'].nunique()

    item=df3.groupby('iid').apply(lambda x: x.sort_values(by=["est"],ascending=False)).reset_in
dex(drop=True)

    s=item[item['r_ui'] > threshold1].groupby('iid')['r_ui'].count().reset_index()
    ss = item.pivot_table(index=['iid'],aggfunc='size').reset_index()
    ss.columns = ['iid','counts']
    dff=pd.merge(s, ss, on='iid')
    dff['rate']=dff['r_ui']/dff['counts']
    item_coverage=float(sum(dff['rate']> threshold2))/df3['iid'].nunique()

    catalog_coverage = float(df3['iid'].nunique())/predictions['iid'].nunique()
    end_time=time.time()
    duration=end_time-start_time
    return user_coverage, item_coverage, catalog_coverage,duration

coverage(4, 0.5, predictions)
```

Out[121]:
```
(0.16992864424057086,
 0.09956538917423943,
 0.8473384666889856,
 7.253134489059448)
```

In [104]:
```
test_rmse = accuracy.rmse(pred)
test_mae = accuracy.mae(pred)
```

```
RMSE: 0.8447
MAE:  0.6571
```

Future Exploration on Sample Size

In [112]:

```
import time
def default_nmf(train_set,test_set):
    method = NMF()
    method.fit(train_set.build_full_trainset())
    train_result = method.test(train_set.df.to_numpy())
    train_rmse = accuracy.rmse(train_result)
    train_mae = accuracy.mae(train_result)
    r_user_coverage,r_item_coverage,r_catelog_coverage,_=coverage(4.0, 0.5,train_result)
    test_result = method.test(test_set.df.to_numpy())
    test_rmse = accuracy.rmse(test_result)
    test_mae = accuracy.mae(test_result)
    e_user_coverage,e_item_coverage,e_catelog_coverage,_=coverage(4.0, 0.5,test_result)
    return train_rmse,test_rmse,train_mae,test_mae,\
r_user_coverage,r_item_coverage,r_catelog_coverage,\
e_user_coverage,e_item_coverage,e_catelog_coverage

def change_size(size, ratings):
    using, discard = train_test_split(ratings, test_size=size, random_state=50)
    train, test = train_test_split(using, test_size=0.2, random_state=50)
    train_set = Dataset.load_from_df(train, reader)
    test_set = Dataset.load_from_df(test, reader)
    start_time = time.time()
    print('use'+" "+str(round((1-size)*100,2))+"% of data")
    train_rmse,test_rmse,train_mae,test_mae,\
    r_user_coverage,r_item_coverage,r_catelog_coverage,\
    e_user_coverage,e_item_coverage,e_catelog_coverage =default_nmf(train_set,test_set)

    end_time = time.time()

    duration=end_time-start_time
    return train_rmse,test_rmse,train_mae,test_mae,duration,\
r_user_coverage,r_item_coverage,r_catelog_coverage,\
e_user_coverage,e_item_coverage,e_catelog_coverage
```

In [ ]:

```
ratings = pd.read_csv(DATA_PATH)
ratings.reset_index(drop=True, inplace=True)
reader = Reader(rating_scale=(0.5, 5))


size_train_rmse_list=[]
size_train_mae_list=[]
size_test_rmse_list=[]
size_test_mae_list=[]
size_e_u_cov=[]
size_e_i_cov=[]
size_e_c_cov=[]
size_r_u_cov=[]
size_r_i_cov=[]
size_r_c_cov=[]
size_list = np.arange(0.1, 1.0, 0.1)
duration_list=[]
for size in size_list:
    train_rmse,train_mae,test_rmse,test_mae,duration,\
    r_user_coverage,r_item_coverage,r_catelog_coverage,\
    e_user_coverage,e_item_coverage,e_catelog_coverage=change_size(size,ratings)
    size_train_rmse_list.append(train_rmse)
    size_train_mae_list.append(train_mae)

    size_test_rmse_list.append(test_rmse)
    size_test_mae_list.append(test_mae)

    size_e_u_cov.append(e_user_coverage)
    size_e_i_cov.append(e_item_coverage)
    size_e_c_cov.append(e_catelog_coverage)

    size_r_u_cov.append(r_user_coverage)
    size_r_i_cov.append(r_item_coverage)
    size_r_c_cov.append(r_catelog_coverage)

    duration_list.append(duration)
    print('duration is',duration)
```

```
use 90.0% of data
RMSE: 0.7635
MAE:  0.5855
RMSE: 0.8846
MAE:  0.6784
duration is 61.11758899688721
use 80.0% of data
RMSE: 0.7501
MAE:  0.5747
RMSE: 0.8864
MAE:  0.6801
duration is 55.64995455741882
use 70.0% of data
RMSE: 0.7379
MAE:  0.5647
RMSE: 0.8905
MAE:  0.6831
duration is 50.34626770019531
use 60.0% of data
RMSE: 0.7194
MAE:  0.5483
RMSE: 0.8977
MAE:  0.6909
duration is 45.02714824676514
use 50.0% of data
RMSE: 0.6926
MAE:  0.5262
RMSE: 0.9063
MAE:  0.6978
duration is 39.74428987503052
use 40.0% of data
RMSE: 0.6618
MAE:  0.4981
RMSE: 0.9185
MAE:  0.7068
duration is 34.80535173416138
use 30.0% of data
RMSE: 0.6095
MAE:  0.4521
RMSE: 0.9428
MAE:  0.7256
duration is 29.039472818374634
use 20.0% of data
RMSE: 0.5317
MAE:  0.3834
RMSE: 0.9791
MAE:  0.7537
duration is 23.681453227996826
use 10.0% of data
RMSE: 0.3745
MAE:  0.2540
RMSE: 1.0529
MAE:  0.8173
duration is 16.54901695251465
```

In [114]:
```
sample_size_list=[1-i for i in size_list]
model_size = pd.DataFrame()
model_size['Sample_size']=sample_size_list
model_size['Running_time']=duration_list
model_size['RMSE_test']=size_test_rmse_list
model_size['MAE_test']=size_test_mae_list
model_size['Catalog_Coverage_test']=size_e_c_cov
model_size['User_Coverage_test']=size_e_u_cov
model_size['Item_Coverage_test']=size_e_i_cov
```

In [115]:
```
model_size
```

Out[115]:

	Sample_size	Running_time	RMSE_test	MAE_test	Catalog_Coverage_test	User_Coverage_test	Item_Coverage_test
0	0.9	61.117589	0.585483	0.678427	0.863773	0.167171	0.093545
1	0.8	55.649955	0.574737	0.680055	0.863073	0.167596	0.093871
2	0.7	50.346268	0.564697	0.683089	0.866868	0.165064	0.082785
3	0.6	45.027148	0.548330	0.690876	0.873235	0.163561	0.087794
4	0.5	39.744290	0.526208	0.697799	0.874029	0.165837	0.084621
5	0.4	34.805352	0.498069	0.706798	0.888697	0.164306	0.080902
6	0.3	29.039473	0.452110	0.725607	0.884669	0.170477	0.079491
7	0.2	23.681453	0.383433	0.753691	0.911111	0.177903	0.085573
8	0.1	16.549017	0.253970	0.817301	0.956561	0.206375	0.101592

In [119]:
```
list(model_size['Running_time'])
```

Out[119]:
```
[61.11758899688721,
 55.64995455741882,
 50.34626770019531,
 45.02714824676514,
 39.74428987503052,
 34.80535173416138,
 29.039472818374634,
 23.681453227996826,
 16.54901695251465]
```

In [ ]: