ECE 475 Project 7: Recommender Systems (& NMF)

Tiffany Yu, Jonathan Lam, Harris Paspuleti

> **To implement a basic recommendation system. Many of those datasets are already loaded into the Surpise package to make this easy. You should tune and cross validate your system to select the best values for the # of latent dimensions, the regularization parameter, and any other hyperparameters.**

In [ ]:

```
# get surprise package for NMF
!pip install scikit-surprise
```

```
Collecting scikit-surprise
  Downloading https://files.pythonhosted.org/packages/97/37/5d334adaf5ddd65da99fc65f6507e
0e4599d092ba048f4302fe8775619e8/scikit-surprise-1.1.1.tar.gz (11.8MB)
     |████████████████████████████████| 11.8MB 8.3MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (fr
om scikit-surprise) (0.17.0)
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (f
rom scikit-surprise) (1.18.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (fr
om scikit-surprise) (1.4.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (fro
m scikit-surprise) (1.15.0)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp36-cp36m-linux_x86_
64.whl size=1670943 sha256=804d6c6fafdc481a633606b7f462c7a284a9177d87f39128828be1f9e86e1a
11
  Stored in directory: /root/.cache/pip/wheels/78/9c/3d/41b419c9d2aff5b6e2b4c0fc8d25c5382
02834058f9ed110d0
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.1
```

In [ ]:

```
import csv
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt

from surprise import NMF
from surprise import Dataset
from surprise.model_selection import cross_validate
from surprise.model_selection.search import GridSearchCV
from surprise.model_selection.split import train_test_split
from surprise import accuracy
```

Dataset: MovieLens 100K

> **MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.**

This dataset includes a set of user details, movie details, and user ratings for movies. Each user rated at least 20 movies.

For our current analysis, we do not use the user nor the movie details (only use the ratings information).

For our current analysis, we do not use the user nor the movie details (only use the ratings information).

In [ ]:

```
# Load movielens-100k dataset (https://surprise.readthedocs.io/en/stable/dataset.html)
data = Dataset.load_builtin('ml-100k')

# Split into training and testing sets
train, test = train_test_split(data, test_size=0.2)

# show the first few ratings: tuples of (user, movie, rating)
some_ratings = []
for i, rating in enumerate(train.all_ratings()):
  if i > 20:
    break;
  some_ratings.append(rating)

print("Here is a sample of the ratings to show their format:")
some_ratings
```

```
Here is a sample of the ratings to show their format:
```

Out[ ]:

```
[(0, 0, 3.0),
 (0, 425, 5.0),
 (0, 421, 5.0),
 (0, 565, 3.0),
 (0, 396, 5.0),
 (0, 698, 4.0),
 (0, 867, 4.0),
 (0, 27, 2.0),
 (0, 888, 3.0),
 (0, 371, 2.0),
 (0, 253, 3.0),
 (0, 753, 5.0),
 (0, 904, 4.0),
 (0, 1009, 3.0),
 (0, 62, 2.0),
 (0, 1013, 2.0),
 (0, 49, 3.0),
 (0, 166, 4.0),
 (0, 820, 3.0),
 (0, 342, 3.0),
 (0, 707, 2.0)]
```

Hyperparameter tuning

We use the Surprise libary's builtin `GridSearchCV` for automatic hyperparameter tuning and cross-validation.

All models learn a bias for the user and movie matrices, and use the default number of epochs (50).

Explanation of hyperparameters:

- `n_factors` **: Dimensionality of the latent space**
- `reg_pu` **: Regularization coefficient for the p_u (user) matrix**
- `reg_qi` **: Regularization coefficient for the q_i (item/movie) matrix**
- `reg_bu` **: Regularization coefficient for the user matrix bias**
- `reg_bi` **: Regularization coefficient for the movie matrix bias**

For sake of time, we did not try very many hyperparameter combinations.

In [ ]:

```
# grid search
params = {
    'n_factors': [10, 20, 30, 40],
    'reg_pu': [0.02, 0.002],
    'reg_qi': [0.02, 0.002],
    'biased': [True],
    'reg_bu': [0.02, 0.002],
    'reg_bi': [0.02, 0.002]
}
grid_search = GridSearchCV(NMF, params, refit=True,
                           return_train_measures=True, n_jobs=-1)
grid_search.fit(data)

algo = grid_search.best_estimator['rmse']
```

Just to get an idea of what our model looks like (and make sure it is learning), print out some of the learned parameters and the cross validation results:

In [ ]:

```
print(grid_search.best_score, grid_search.best_params, grid_search.cv_results)
```

```
{'rmse': 0.953552256033319, 'mae': 0.7469554124946353} {'rmse': {'n_factors': 10, 'reg_pu
': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, 'mae': {'n_fa
ctors': 10, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.
002}} {'split0_test_rmse': array([1.12671965, 0.95368045, 0.95090741, 0.95168667, 0.96127
206,
       0.96771622, 0.96343015, 0.97002468, 1.10883185, 0.96771741,
       0.9631692 , 0.96989675, 0.97158942, 0.9766609 , 0.97787778,
       0.98198263, 1.31584452, 1.18207468, 1.2660726 , 1.23342571,
       1.27359044, 1.31343888, 1.45571804, 1.47059832, 1.47713136,
       1.90276703, 1.30630309, 1.25483712, 1.80809646, 1.76598118,
       1.52439289, 1.60655147, 1.38715191, 1.2228146 , 1.28089097,
       1.46255891, 1.93239573, 1.68333947, 1.44744428, 2.32940068,
       1.31393296, 1.39052051, 1.6002692 , 1.29269805, 1.61572357,
       1.36836678, 1.71614943, 1.7443026 , 1.51647067, 1.48985898,
       1.23764595, 1.25609113, 1.32320693, 1.8089286 , 1.51132864,
       1.35366777, 1.38264658, 1.93861343, 1.56012656, 1.33076641,
       1.66453789, 1.43874401, 1.42124056, 1.42699327]), 'split0_train_rmse': array([1.09
494471, 0.82894334, 0.81912312, 0.82866759, 0.81609649,
       0.81928227, 0.81197344, 0.8109066 , 1.05844246, 0.81570303,
       0.81565164, 0.81170432, 0.81342611, 0.81052242, 0.80694748,
       0.8082808 , 1.27987854, 1.17072659, 1.24279676, 1.20336299,
       1.26260439, 1.28340369, 1.43907055, 1.43765754, 1.44416105,
       1.87082839, 1.276055  , 1.23322462, 1.79514949, 1.74761332,
       1.49740724, 1.56995021, 1.36279707, 1.20396436, 1.25769037,
       1.42251341, 1.92393414, 1.67406066, 1.41587232, 2.33549028,
       1.28809376, 1.36891215, 1.56319027, 1.26819225, 1.59076439,
       1.34689121, 1.67426619, 1.72748165, 1.47324816, 1.47586992,
       1.19972935, 1.21738893, 1.2793612 , 1.77473322, 1.49019601,
       1.3236147 , 1.35176904, 1.93283963, 1.54228655, 1.29451702,
       1.64866795, 1.41419831, 1.39653201, 1.40498487]), 'split1_test_rmse': array([0.957
8362 , 0.95660042, 0.96030242, 0.95824099, 0.97006745,
       0.96924462, 0.97296251, 0.96784403, 0.967115  , 0.96812397,
       0.97132114, 0.96953766, 0.97416366, 0.97853791, 0.98172119,
       0.97390263, 1.24864739, 1.56488316, 1.31758603, 1.28044844,
       1.38535283, 2.10757309, 2.0311708 , 1.25891636, 1.44532336,
       1.23976034, 1.47038099, 1.5209942 , 1.55909743, 1.95103046,
       1.43080111, 1.45021189, 1.36112209, 1.28249282, 1.3153543 ,
       1.42651181, 1.50118939, 1.48415302, 1.43893092, 1.51442187,
       1.96476608, 1.47338056, 1.6501675 , 1.4957885 , 2.02389459,
       2.11349589, 1.94558135, 2.28154057, 1.47319227, 1.34365673,
       1.24881571, 1.49416376, 1.55417545, 1.77999499, 1.45323653,
       1.40291997, 1.48223372, 1.38457983, 2.25498252, 1.49256109,
       1.45813015, 1.60603217, 1.94118652, 1.70531816]), 'split1_train_rmse': array([0.82
723242, 0.82623144, 0.8207223 , 0.81705689, 0.81663112,
       0.81672125, 0.81449109, 0.81293855, 0.81328882, 0.81372156,
       0.80844257, 0.81020483, 0.80806669, 0.80749009, 0.80640939,
       0.80220142, 1.22846284, 1.54054907, 1.27499007, 1.25725866,
       1.36573022, 2.09444374, 2.01256114, 1.22961918, 1.43085508,
       1.19893464, 1.42678317, 1.49092354, 1.50725412, 1.94103673,
       1.40323207, 1.40886717, 1.32141833, 1.2471837 , 1.28090066,
       1.39600264, 1.46884122, 1.46657492, 1.4124337 , 1.48744224,
       1.95140595, 1.4318439 , 1.6247481 , 1.45564082, 1.99278958,
       2.1098266 , 1.92392821, 2.28467115, 1.44665497, 1.31277695,
       1.2235223 , 1.45679334, 1.53262283, 1.76703582, 1.42255336,
       1.38780033, 1.45209258, 1.34665376, 2.25890768, 1.46560841,
       1.41322179, 1.54947572, 1.91232746, 1.69501755]), 'split2_test_rmse': array([0.950
64383, 0.95096055, 0.94962223, 0.9500036 , 0.95833966,
       0.96932538, 0.96192836, 0.96546021, 0.96027629, 0.95987607,
       0.96944137, 0.96597663, 0.97640751, 0.96669372, 0.96985083,
       0.97452261, 1.45997998, 2.0345994 , 1.25876783, 1.18867224,
       1.52553929, 1.21856352, 1.47322512, 1.65151549, 2.16393136,
       1.30047121, 1.28784365, 1.92560739, 2.0533516 , 1.6107345 ,
       1.7291813 , 1.5310527 , 1.39841073, 1.52021681, 1.29448061,
       1.18712045, 1.26363778, 1.59315696, 2.07191768, 1.34840731,
       1.61098985, 1.60547988, 1.52687754, 2.23617758, 1.43618578,
       1.75841425, 1.90521781, 1.4045004 , 1.44667007, 1.51656437,
       1.44219474, 1.16801012, 1.45782801, 1.3845492 , 1.24080563,
       1.75477752, 1.34014404, 1.4182336 , 1.40958935, 1.53226556,
       1.50047696, 2.44259323, 2.07185971, 2.16331437]), 'split2_train_rmse': array([0.82
610007, 0.82479289, 0.8214557 , 0.82355385, 0.8193141 ,
       0.81688956, 0.81209574, 0.81141458, 0.81461618, 0.81413053,
       0.81414706, 0.81440253, 0.81102274, 0.80842588, 0.80715858,
       0.80747788, 1.4395832 , 2.02227308, 1.23986693, 1.16417239,
       1.50389908, 1.18743723, 1.46503316, 1.64904204, 2.16333697,
       1.28028017, 1.26903003, 1.90904926, 2.05212203, 1.58229121,
       1.7090181 , 1.51351441, 1.35980268, 1.4906797 , 1.27593564,
       1.15394371, 1.23443726, 1.57145944, 2.05750373, 1.31675708,
       1.58759014, 1.57568775, 1.50328568, 2.22321941, 1.40945237,
       1.74068722, 1.87951577, 1.38989903, 1.43465469, 1.51237645,
       1.41861206, 1.13912279, 1.42162109, 1.35252327, 1.22136241,
       1.7348762 , 1.30403402, 1.40232066, 1.38401217, 1.51616178,
       1.4827609 , 2.44734427, 2.07448418, 2.14988323]), 'split3_test_rmse': array([0.952
90732, 0.9539179 , 0.95346237, 0.95628911, 0.95829373,
       0.96316094, 0.95875407, 0.96616403, 0.96683996, 0.96734258,
       0.96869506, 0.96120746, 0.97199538, 0.97521209, 0.9753017 ,
       0.97449908, 1.74349587, 1.26266514, 1.17039699, 1.29248545,
       1.80056384, 1.31750135, 1.57674457, 1.67041406, 2.40256243,
       1.32455094, 2.00408043, 1.2548823 , 1.34944747, 1.91233224,
       1.62165537, 1.40502404, 1.3180064 , 1.26470931, 1.20349349,
       1.91575091, 1.34169695, 1.98289808, 1.90086892, 1.39723572,
       1.50589115, 1.40362242, 1.37605806, 1.47937256, 1.83075118,
       1.54064472, 1.64456455, 1.47976693, 1.2921631 , 1.65599155,
       1.84058164, 1.55775434, 1.54793306, 1.40477873, 1.36530886,
       1.83278872, 1.3705851 , 1.38927018, 1.42809924, 1.3762888 ,
       1.45880944, 1.53530377, 1.33686885, 1.41598549]), 'split3_train_rmse': array([0.82
561274, 0.82352473, 0.82321092, 0.82109328, 0.81971579,
       0.81456774, 0.81210992, 0.81485983, 0.81562557, 0.81827036,
       0.80983852, 0.80813111, 0.81137424, 0.81064366, 0.80640802,
       0.80783452, 1.70621493, 1.22329865, 1.13326096, 1.26243811,
       1.78228415, 1.28643869, 1.55492636, 1.63573052, 2.39810449,
       1.29249424, 1.96609066, 1.22040543, 1.31083064, 1.88741828,
       1.59679243, 1.37422276, 1.28184245, 1.24068041, 1.17927727,
       1.90566713, 1.31792605, 1.95629873, 1.89276892, 1.34840026,
       1.49439974, 1.37583372, 1.3501611 , 1.46539466, 1.80957817,
       1.52069854, 1.58580281, 1.44052631, 1.26827912, 1.62549052,
       1.83721497, 1.52765175, 1.5084837 , 1.34993367, 1.32716033,
       1.79857527, 1.3354473 , 1.37292723, 1.39724822, 1.3325284 ,
       1.42370782, 1.48866219, 1.30857875, 1.39271521]), 'split4_test_rmse': array([0.954
5566 , 0.95664939, 0.95736858, 0.95154091, 0.96634157,
       0.97287203, 0.96974386, 0.97143182, 0.97046323, 0.96626402,
       0.97379703, 0.96801789, 0.97231604, 0.97276194, 0.98264282,
       0.98185131, 1.44304839, 1.35040008, 1.23130474, 1.62117892,
       1.58190445, 1.20571812, 1.44651165, 1.62357041, 1.22304471,
       1.43134355, 1.2085166 , 1.54332326, 2.05982862, 1.77570712,
       1.88489322, 1.73307092, 1.46126418, 1.29597786, 1.39784167,
       2.23484977, 1.3741303 , 1.46073103, 1.40637114, 1.40011268,
       1.52735561, 1.99271722, 1.70643891, 1.61428064, 1.83208555,
       1.68000218, 1.52894139, 1.33302387, 1.30719576, 1.39288763,
       1.43192183, 1.32617638, 1.63152027, 1.34263301, 1.74770132,
       1.27102044, 1.49669123, 1.53980187, 1.23449761, 1.30051032,
       1.25080905, 1.28600028, 1.23798562, 1.43952827]), 'split4_train_rmse': array([0.82
884392, 0.82712005, 0.82263637, 0.82001217, 0.81812531,
       0.81609555, 0.81143645, 0.80907516, 0.81268806, 0.80932167,
       0.81184813, 0.80771344, 0.81200895, 0.80803485, 0.80926807,
       0.80248564, 1.4135166 , 1.31359614, 1.20375775, 1.57755923,
       1.55911889, 1.16944291, 1.4132578 , 1.60065981, 1.18953953,
       1.39580971, 1.17952447, 1.51796188, 2.07796179, 1.76244785,
       1.87539935, 1.70620512, 1.44520011, 1.25506248, 1.36690151,
       2.22649187, 1.3472133 , 1.44205405, 1.37417552, 1.38053615,
       1.49022944, 1.97301275, 1.69139741, 1.58066763, 1.81432778,
       1.6590389 , 1.49621728, 1.296639  , 1.2689719 , 1.37562751,
       1.40343832, 1.2987945 , 1.58847957, 1.31496403, 1.7190942 ,
       1.23763996, 1.48206292, 1.51584327, 1.18850302, 1.25631959,
       1.19873072, 1.24293872, 1.19881469, 1.41905309]), 'mean_test_rmse': array([0.98853
272, 0.95436174, 0.9543326 , 0.95355226, 0.96286289,
       0.96846384, 0.96536379, 0.96818496, 0.99470527, 0.96586481,
       0.96928476, 0.96692728, 0.9732944 , 0.97397331, 0.97747886,
       0.97735165, 1.44220323, 1.47892449, 1.24882564, 1.32324215,
       1.51339017, 1.43255899, 1.59667403, 1.53500293, 1.74239864,
       1.43977862, 1.45542495, 1.49992885, 1.76596431, 1.8031571 ,
       1.63818478, 1.5451822 , 1.38519106, 1.31724228, 1.29841221,
       1.64535837, 1.48261003, 1.64085571, 1.65310659, 1.59791565,
       1.58458713, 1.57314412, 1.57196224, 1.62366346, 1.74772813,
       1.69218476, 1.74809091, 1.64862687, 1.40713837, 1.47979185,
       1.44023198, 1.36043915, 1.50293274, 1.54417691, 1.4636762 ,
       1.52303488, 1.41446013, 1.53409978, 1.57745906, 1.40647843,
       1.4665527 , 1.66173469, 1.60182825, 1.63022791]), 'std_test_rmse': array([0.069133
23, 0.00212011, 0.0039883 , 0.0031494 , 0.00464488,
       0.00314527, 0.00521833, 0.00226059, 0.05715874, 0.00305758,
       0.0035293 , 0.00317621, 0.00178878, 0.00410008, 0.0046422 ,
       0.00373441, 0.17001433, 0.30580057, 0.04812968, 0.15344031,
       0.17955946, 0.34068247, 0.22218983, 0.15500148, 0.45647673,
       0.23963335, 0.28723895, 0.24643328, 0.27820963, 0.12078721,
       0.15835102, 0.11656043, 0.04702939, 0.10443626, 0.06245841,
       0.37733372, 0.2375818 , 0.18871866, 0.27778967, 0.36978221,
       0.21352064, 0.22321691, 0.11437867, 0.32310108, 0.20236638,
       0.24894188, 0.15713422, 0.34567333, 0.09065016, 0.1082451 ,
       0.21818985, 0.14553827, 0.10537685, 0.20554145, 0.16875867,
       0.22639619, 0.06295271, 0.20996219, 0.35424557, 0.09067038,
       0.13192817, 0.40490216, 0.33802085, 0.28753789]), 'mean_train_rmse': array([0.8805
4677, 0.82612249, 0.82142968, 0.82207676, 0.81797656,
       0.81671128, 0.81242133, 0.81183895, 0.86293222, 0.81422943,
       0.81198558, 0.81043125, 0.81117975, 0.80902338, 0.80723831,
       0.80565605, 1.41353122, 1.45408871, 1.2189345 , 1.29295828,
       1.49472735, 1.40423325, 1.5769698 , 1.51054182, 1.72519942,
       1.40766943, 1.42349666, 1.47431295, 1.74866361, 1.78416148,
       1.61636984, 1.51455194, 1.35421213, 1.28751413, 1.27214109,
       1.62092375, 1.4584704 , 1.62208956, 1.63055084, 1.5737252 ,
       1.56234381, 1.54505806, 1.54655651, 1.59862296, 1.72338246,
       1.67542849, 1.71194605, 1.62784343, 1.37836177, 1.46042827,
       1.4165034 , 1.32795026, 1.46611368, 1.511838  , 1.43607326,
       1.49650129, 1.38508117, 1.51411691, 1.55419153, 1.37302704,
       1.43341784, 1.62852384, 1.57814742, 1.61233079]), 'std_train_rmse': array([0.10720
473, 0.00186978, 0.0014457 , 0.00389999, 0.00142695,
       0.00152419, 0.00106372, 0.00195053, 0.09776048, 0.00292761,
       0.00265683, 0.00245439, 0.00175996, 0.00130825, 0.00105711,
       0.00271809, 0.16640661, 0.31100006, 0.04841506, 0.14684466,
       0.17744894, 0.34842298, 0.22296787, 0.15950261, 0.46846951,
       0.23988349, 0.28268019, 0.25050725, 0.30088586, 0.12476431,
       0.16470766, 0.11888022, 0.05421545, 0.10308236, 0.05983841,
       0.38879765, 0.24457545, 0.18631037, 0.28650911, 0.38519002,
       0.21773226, 0.22652377, 0.11645859, 0.32796989, 0.20219822,
       0.254993  , 0.16546108, 0.35853927, 0.09046614, 0.1087885 ,
       0.22865085, 0.14505923, 0.10774802, 0.21194027, 0.16803314,
       0.22662134, 0.06933664, 0.21716746, 0.36988833, 0.10048229,
       0.14446284, 0.42211091, 0.34858894, 0.29133463]), 'rank_test_rmse': array([15,  3,
2,  1,  4,  9,  5,  8, 16,  6, 10,  7, 11, 12, 14, 13, 29,
       33, 17, 20, 38, 26, 48, 41, 60, 27, 30, 36, 63, 64, 53, 43, 22, 19,
       18, 55, 35, 54, 57, 49, 47, 45, 44, 51, 61, 59, 62, 56, 24, 34, 28,
       21, 37, 42, 31, 39, 25, 40, 46, 23, 32, 58, 50, 52]), 'split0_test_mae': array([0.
85340428, 0.74764644, 0.74512498, 0.74475262, 0.75142782,
       0.75750802, 0.75333852, 0.7575778 , 0.85958439, 0.75544959,
       0.75394885, 0.7564572 , 0.75917736, 0.76289087, 0.76148228,
       0.76506923, 1.00083899, 0.89767931, 0.95503269, 0.94437036,
       0.95439935, 1.00608649, 1.11658584, 1.13982912, 1.13534963,
       1.54748801, 0.988979  , 0.94802946, 1.44428642, 1.39484323,
       1.17323967, 1.24693971, 1.08231383, 0.92751846, 0.97322231,
       1.13266444, 1.56138649, 1.30478834, 1.08959251, 1.97479931,
       0.99978137, 1.06015886, 1.25437042, 0.98586245, 1.2639133 ,
       1.05948447, 1.360073  , 1.36984881, 1.17227034, 1.14870524,
       0.93822918, 0.9590122 , 0.99932788, 1.46166697, 1.16444477,
       1.02667634, 1.0555321 , 1.56241185, 1.19652328, 1.01420747,
       1.34519066, 1.11354883, 1.07578111, 1.10100424]), 'split0_train_mae': array([0.828
90759, 0.65041538, 0.64233104, 0.64868593, 0.63827099,
       0.64247297, 0.63567967, 0.6351583 , 0.81863561, 0.63839991,
       0.63791967, 0.63477498, 0.6362122 , 0.6339952 , 0.62875512,
       0.63122395, 0.96967062, 0.88355219, 0.93517084, 0.91753747,
       0.9397114 , 0.97895076, 1.09771219, 1.1131212 , 1.10962286,
       1.51330514, 0.96470012, 0.92809919, 1.43106428, 1.37528469,
       1.15008686, 1.21130201, 1.05582411, 0.90818614, 0.95168104,
       1.09782126, 1.54871295, 1.29155306, 1.05803984, 1.97956884,
       0.971164  , 1.0393474 , 1.21887503, 0.96039146, 1.24047674,
       1.03849396, 1.31709732, 1.34973511, 1.13241042, 1.13498575,
       0.90775298, 0.92729628, 0.96162597, 1.42370778, 1.14196786,
       1.00363193, 1.02917875, 1.55494121, 1.17958016, 0.98317246,
       1.32841904, 1.08775514, 1.04954214, 1.07673668]), 'split1_test_mae': array([0.7491
6607, 0.74874853, 0.75126797, 0.74916753, 0.75567557,
       0.75586036, 0.75962952, 0.75578638, 0.75212779, 0.75352385,
       0.75514196, 0.75604545, 0.75904808, 0.7610005 , 0.76313096,
       0.7586177 , 0.95214262, 1.21355348, 1.00375792, 0.97103956,
       1.05303688, 1.73351357, 1.65314416, 0.96529177, 1.122417  ,
       0.93622833, 1.15381359, 1.17333288, 1.2017342 , 1.57007267,
       1.08889174, 1.10541948, 1.04512319, 0.97313002, 1.00130008,
       1.100923  , 1.14536317, 1.14309902, 1.09294164, 1.1468375 ,
       1.5390504 , 1.14470085, 1.28885877, 1.16240198, 1.67436066,
       1.73480152, 1.57192593, 1.92709296, 1.14582779, 1.02851579,
       0.95471929, 1.14410375, 1.20920948, 1.40947893, 1.09714065,
       1.05621847, 1.15307043, 1.05683378, 1.89076355, 1.15708667,
       1.11801339, 1.26024047, 1.57915634, 1.35051446]), 'split1_train_mae': array([0.649
67565, 0.64953351, 0.64429604, 0.64177335, 0.63929033,
       0.64046955, 0.6372398 , 0.63635262, 0.6372387 , 0.63742852,
       0.63285634, 0.63387812, 0.63167247, 0.63227363, 0.62989856,
       0.62709495, 0.92870597, 1.18995495, 0.96971564, 0.95207731,
       1.03140204, 1.7208433 , 1.63394175, 0.93815955, 1.10314461,
       0.90699021, 1.11571133, 1.14991279, 1.15676986, 1.56350452,
       1.06591326, 1.0713976 , 1.01050357, 0.9393959 , 0.9720729 ,
       1.07145534, 1.11672543, 1.12736464, 1.07054593, 1.12442545,
       1.52597613, 1.11023008, 1.26576297, 1.1232566 , 1.64019267,
       1.72976527, 1.5521985 , 1.9310018 , 1.12023157, 1.00222834,
       0.93476597, 1.11284312, 1.19380748, 1.39735225, 1.06817351,
       1.04020355, 1.12874076, 1.02192181, 1.89563071, 1.13339007,
       1.07693181, 1.20985362, 1.54667171, 1.34123927]), 'split2_test_mae': array([0.7473
9421, 0.74784975, 0.74542023, 0.74588931, 0.75087521,
       0.75978645, 0.75338316, 0.75555328, 0.75107524, 0.75195175,
       0.75996583, 0.75463858, 0.76202813, 0.75649869, 0.75861401,
       0.76304636, 1.1227001 , 1.65806025, 0.95876043, 0.91113034,
       1.1915175 , 0.92380773, 1.12025252, 1.28606564, 1.78443254,
       0.98232284, 0.97590514, 1.57197835, 1.66994225, 1.24464968,
       1.36455234, 1.16210802, 1.06624604, 1.18549072, 0.98908815,
       0.90521737, 0.95956649, 1.20536042, 1.70762061, 1.02907832,
       1.2494465 , 1.24836905, 1.18176198, 1.88244991, 1.0989785 ,
       1.38732907, 1.53921007, 1.07068354, 1.12924429, 1.20019893,
       1.11055322, 0.88899929, 1.11997792, 1.03949982, 0.94963339,
       1.38836246, 1.02851806, 1.10032943, 1.0697745 , 1.18592792,
       1.15340111, 2.10453203, 1.69386965, 1.79388521]), 'split2_train_mae': array([0.648
23572, 0.64682518, 0.64380138, 0.6453061 , 0.64151861,
       0.63983354, 0.63455134, 0.63441994, 0.63655227, 0.6366446 ,
       0.63551094, 0.63654562, 0.63362252, 0.63111007, 0.62968951,
       0.63056381, 1.10481705, 1.64682919, 0.94026819, 0.88715002,
       1.17050114, 0.89247149, 1.10838849, 1.27877027, 1.78407061,
       0.96352686, 0.95499137, 1.55111431, 1.6681957 , 1.21553067,
       1.34502602, 1.14512431, 1.03365396, 1.15797972, 0.97216023,
       0.87805667, 0.93137391, 1.18316778, 1.6929619 , 0.99797369,
       1.22743756, 1.22482557, 1.15824059, 1.8669874 , 1.07242881,
       1.37252412, 1.51409554, 1.05731487, 1.11591404, 1.19157528,
       1.08619997, 0.86227315, 1.08423221, 1.0097523 , 0.93207226,
       1.36708001, 0.99861368, 1.08661129, 1.04684715, 1.1701736 ,
       1.12959176, 2.11052821, 1.69789271, 1.77896176]), 'split3_test_mae': array([0.7474
9069, 0.74772897, 0.74842601, 0.74973379, 0.74992447,
       0.75500426, 0.75039508, 0.75669345, 0.75390989, 0.75756581,
       0.75594678, 0.75127645, 0.75973009, 0.76176562, 0.76221314,
       0.76037692, 1.36482907, 0.96299614, 0.89085098, 0.97350275,
       1.4020625 , 0.9998794 , 1.21796715, 1.30430496, 2.05183291,
       1.01446486, 1.6315985 , 0.95759352, 1.02444812, 1.54519864,
       1.25344414, 1.0881809 , 0.99581831, 0.9575839 , 0.92219967,
       1.52623805, 1.01085005, 1.62395343, 1.51963456, 1.05968371,
       1.16498008, 1.06443271, 1.03811574, 1.13679343, 1.4573391 ,
       1.17972873, 1.28603272, 1.13389341, 0.99547246, 1.28743871,
       1.44341054, 1.20924187, 1.19243611, 1.07157798, 1.03206357,
       1.46621396, 1.04370349, 1.05677748, 1.09484245, 1.04942898,
       1.09373936, 1.17579316, 1.01561364, 1.08680988]), 'split3_train_mae': array([0.647
36594, 0.64669332, 0.6460459 , 0.64422488, 0.64137359,
       0.63830323, 0.63607431, 0.63785185, 0.63900733, 0.64078712,
       0.63298913, 0.63185018, 0.63388374, 0.63358647, 0.62923382,
       0.63057699, 1.33081825, 0.93315456, 0.86089207, 0.95218984,
       1.38554513, 0.97422358, 1.20333879, 1.27570633, 2.0531165 ,
       0.98206473, 1.59192798, 0.92684039, 0.99289931, 1.52253944,
       1.22944818, 1.06010314, 0.96460702, 0.93404872, 0.90481256,
       1.51841853, 0.99583819, 1.60371934, 1.5123406 , 1.0199443 ,
       1.15457047, 1.0413878 , 1.01594088, 1.12478922, 1.44085435,
       1.15979338, 1.24082174, 1.100003  , 0.97691289, 1.25948865,
       1.44369927, 1.18053393, 1.15937413, 1.02753335, 1.00230245,
       1.43514757, 1.01406161, 1.04105655, 1.07000269, 1.01597601,
       1.06270524, 1.13725152, 0.98795827, 1.07071716]), 'split4_test_mae': array([0.7484
2693, 0.74886187, 0.75075723, 0.74523381, 0.75596175,
       0.76133544, 0.75620984, 0.75872802, 0.75892279, 0.75561609,
       0.7603749 , 0.75611142, 0.75847172, 0.76031588, 0.76556849,
       0.7655397 , 1.09821746, 1.03602763, 0.93556613, 1.25712806,
       1.22013492, 0.92445191, 1.09276157, 1.26335729, 0.92867365,
       1.088312  , 0.92132554, 1.17847636, 1.68435981, 1.41519813,
       1.49612374, 1.36957798, 1.11661073, 0.99158688, 1.08333837,
       1.86499256, 1.05704086, 1.12583068, 1.08397898, 1.06601687,
       1.16760473, 1.61741264, 1.35397446, 1.25925362, 1.49141511,
       1.32886695, 1.18807266, 1.01120228, 0.99586101, 1.06969341,
       1.10297583, 1.01081157, 1.26573819, 1.02763901, 1.40114016,
       0.97673214, 1.16881805, 1.16904615, 0.94251873, 0.98985755,
       0.94935797, 0.97234509, 0.94652241, 1.10732041]), 'split4_train_mae': array([0.650
78512, 0.64968451, 0.64514702, 0.64286915, 0.64026699,
       0.63908145, 0.63407197, 0.63288722, 0.63544128, 0.632937  ,
       0.63445122, 0.631715  , 0.63387055, 0.63185681, 0.63087556,
       0.62696634, 1.0715412 , 1.00155851, 0.91138682, 1.21508089,
       1.19639503, 0.89344901, 1.05794394, 1.23498077, 0.89706969,
       1.05683302, 0.89316113, 1.15178559, 1.69952709, 1.39815444,
       1.48551117, 1.33911606, 1.10050454, 0.95812669, 1.05313117,
       1.85164146, 1.03136095, 1.10081416, 1.04929204, 1.04435219,
       1.13085953, 1.59353441, 1.33742017, 1.22796944, 1.47436128,
       1.30714958, 1.15322502, 0.97868606, 0.96166201, 1.05161797,
       1.08033286, 0.98576318, 1.22573045, 1.00027346, 1.37213222,
       0.94417633, 1.15407207, 1.14700509, 0.90442049, 0.95263269,
       0.90544037, 0.93377559, 0.91011386, 1.08354964]), 'mean_test_mae': array([0.769176
43, 0.74816711, 0.74819929, 0.74695541, 0.75277296,
       0.7578989 , 0.75459122, 0.75686779, 0.77512402, 0.75482142,
       0.75707566, 0.75490582, 0.75969108, 0.76049431, 0.76220178,
       0.76252998, 1.10774565, 1.15366336, 0.94879363, 1.01143422,
       1.16423023, 1.11754782, 1.24014225, 1.19176976, 1.40454115,
       1.11376321, 1.13432435, 1.16588211, 1.40495416, 1.43399247,
       1.27525032, 1.19444522, 1.06122242, 1.007062  , 0.99382972,
       1.30600708, 1.14684141, 1.28060638, 1.29875366, 1.25528314,
       1.22417262, 1.22701482, 1.22341627, 1.28535228, 1.39720134,
       1.33804215, 1.38906287, 1.3025442 , 1.08773518, 1.14691042,
       1.10997761, 1.04243374, 1.15733792, 1.20197254, 1.12888451,
       1.18284067, 1.08992843, 1.18907974, 1.2388845 , 1.07930172,
       1.1319405 , 1.32529192, 1.26218863, 1.28790684]), 'std_test_mae': array([0.0421189
3, 0.00052622, 0.00257627, 0.00207681, 0.00253449,
       0.00236897, 0.00311899, 0.00117408, 0.04231606, 0.00192231,
       0.00260878, 0.00191848, 0.00123511, 0.00217314, 0.00226221,
       0.00267262, 0.14286192, 0.2733946 , 0.0365832 , 0.12489549,
       0.1528575 , 0.31000146, 0.21091378, 0.12702029, 0.43400739,
       0.22245015, 0.26048102, 0.22626188, 0.25919739, 0.11715052,
       0.14308817, 0.10362484, 0.04018665, 0.0916547 , 0.05224101,
       0.34458845, 0.21609234, 0.18275541, 0.26389228, 0.36186004,
       0.17711121, 0.20684095, 0.10803489, 0.31115507, 0.19799628,
       0.22916758, 0.1468576 , 0.33516373, 0.07641626, 0.09227225,
       0.18156516, 0.11803642, 0.0916909 , 0.19198563, 0.1535582 ,
       0.20270216, 0.05882443, 0.19112703, 0.33583322, 0.07816591,
       0.12722096, 0.40080003, 0.31048645, 0.27126292]), 'mean_train_mae': array([0.68499
4  , 0.64863038, 0.64432428, 0.64457188, 0.6401441 ,
       0.64003215, 0.63552342, 0.63533399, 0.67337504, 0.63723943,
       0.63474546, 0.63375278, 0.6338523 , 0.63256444, 0.62969051,
       0.62928521, 1.08111062, 1.13100988, 0.92348671, 0.9848071 ,
       1.14471095, 1.09198763, 1.22026503, 1.16814762, 1.38940485,
       1.08454399, 1.10409839, 1.14155045, 1.38969125, 1.41500275,
       1.2551971 , 1.16540862, 1.03301864, 0.97954743, 0.97077158,
       1.28347865, 1.12480229, 1.26132379, 1.27663606, 1.23325289,
       1.20200154, 1.20186505, 1.19924793, 1.26067883, 1.37366277,
       1.32154526, 1.35548762, 1.28334817, 1.06142619, 1.1279792 ,
       1.09055021, 1.01374193, 1.12495405, 1.17172383, 1.10332966,
       1.15804788, 1.06493338, 1.17030719, 1.21929624, 1.05106897,
       1.10061764, 1.29583282, 1.23843574, 1.2702409 ]), 'std_train_mae': array([0.071966
38, 0.00155718, 0.00125603, 0.00237975, 0.00123721,
       0.00141996, 0.00112477, 0.00168702, 0.07263951, 0.00256255,
       0.00186556, 0.0018239 , 0.00144048, 0.0010763 , 0.0007115 ,
       0.0018567 , 0.14046603, 0.27809487, 0.03638612, 0.11766329,
       0.1524475 , 0.31664789, 0.21227276, 0.12976543, 0.44696555,
       0.21967089, 0.25466522, 0.22785043, 0.2781981 , 0.12269264,
       0.14741326, 0.10258581, 0.04530324, 0.09063293, 0.047962  ,
       0.35264579, 0.2202467 , 0.18328409, 0.2723331 , 0.37559852,
       0.18232026, 0.20712024, 0.10879717, 0.31502487, 0.19706522,
       0.23483244, 0.15452112, 0.3468106 , 0.07557929, 0.09277133,
       0.19106079, 0.11733633, 0.09425046, 0.19535864, 0.15134019,
       0.20196598, 0.06368943, 0.19708641, 0.34933642, 0.08543281,
       0.13625382, 0.41728906, 0.32010159, 0.27420609]), 'rank_test_mae': array([15,  2,
3,  1,  4, 10,  5,  8, 16,  6,  9,  7, 11, 12, 13, 14, 26,
       35, 17, 20, 37, 29, 48, 41, 62, 28, 32, 38, 63, 64, 51, 42, 22, 19,
       18, 57, 33, 52, 55, 49, 45, 46, 44, 53, 61, 59, 60, 56, 24, 34, 27,
       21, 36, 43, 30, 39, 25, 40, 47, 23, 31, 58, 50, 54]), 'mean_fit_time': array([ 8.7
0077324,  9.18824935,  9.2647296 ,  9.27673426,  9.30260401,
        9.31332755,  9.28104296,  9.35330048,  9.13306317,  9.16470718,
        8.83388491,  9.00846195,  8.79199295,  8.92526679,  8.67318897,
        9.12619419, 11.79453897, 11.813445  , 11.80383377, 11.73262701,
       13.3070085 , 11.77521586, 11.52053161, 11.56952367, 11.17670321,
       11.54245973, 11.27851124, 11.58191743, 11.39072914, 11.40067682,
       11.12915335, 11.29315486, 14.03336987, 14.04079928, 13.91670623,
       13.86813145, 13.3442184 , 15.78374019, 13.61588078, 14.00177021,
       13.44561887, 13.94466748, 13.537925  , 13.76969047, 13.88315678,
       14.00760875, 13.87433329, 13.93899808, 16.38297491, 16.38706818,
       16.26456356, 16.20416617, 15.73288245, 15.88243442, 15.67933855,
       15.99228129, 15.7930501 , 15.86172714, 15.80224471, 15.89892192,
       15.67439847, 16.02205677, 15.83664865, 14.4030746 ]), 'std_fit_time': array([0.461
78218, 0.17651622, 0.0881564 , 0.20088733, 0.19487085,
       0.11412923, 0.09452429, 0.21995843, 0.32264241, 0.27410367,
       0.44974326, 0.45292282, 0.48771432, 0.5613795 , 0.55415364,
       0.47016903, 0.03663351, 0.10916021, 0.1424056 , 0.11406221,
       2.01106688, 0.06933074, 0.26586771, 0.61785455, 0.43533319,
       0.45689052, 0.40496399, 0.43774283, 0.51564935, 0.51069213,
       0.50163369, 0.7405142 , 0.05950839, 0.20678139, 0.29246022,
       0.50034924, 0.571175  , 2.27587699, 0.3716294 , 0.44402895,
       0.64588315, 0.31571237, 0.50555493, 0.40439654, 0.24508718,
       0.28166377, 0.2696396 , 0.35476953, 0.15721747, 0.16021795,
       0.15038527, 0.52488413, 0.54788713, 0.56125457, 0.49306142,
       0.40815473, 0.59979609, 0.38808099, 0.45052894, 0.57914859,
       0.43591216, 0.45492958, 0.71956365, 2.83924265]), 'mean_test_time': array([0.32855
392, 0.33987112, 0.28374987, 0.29850364, 0.29417343,
       0.2571403 , 0.25794449, 0.30069022, 0.33102102, 0.34844813,
       0.33161035, 0.33510728, 0.36483727, 0.29677863, 0.34120293,
       0.33787866, 0.26053619, 0.25765357, 0.25623589, 0.25146823,
       0.33518562, 0.25575504, 0.32877011, 0.29723454, 0.32811751,
       0.3177094 , 0.38025622, 0.29158783, 0.37456713, 0.3353961 ,
       0.32473726, 0.2703567 , 0.25639935, 0.27944288, 0.34474425,
       0.27782001, 0.36074743, 0.38743358, 0.32333484, 0.31697865,
       0.35323849, 0.31659946, 0.35647373, 0.31300406, 0.33982897,
       0.31289678, 0.35006819, 0.33592286, 0.26198096, 0.26265121,
       0.31552949, 0.34579372, 0.37260404, 0.32660551, 0.36445231,
       0.32939882, 0.36773071, 0.34792595, 0.38429432, 0.32091365,
       0.32713633, 0.30926332, 0.4018796 , 0.25495715]), 'std_test_time': array([0.099588
61, 0.0603519 , 0.05243584, 0.07245491, 0.05112234,
       0.00634851, 0.00567331, 0.060772  , 0.06619902, 0.08980315,
       0.06495614, 0.07519341, 0.09610699, 0.04547867, 0.07496804,
       0.07924298, 0.00414254, 0.00313497, 0.00551314, 0.00205953,
       0.14085961, 0.00629812, 0.07461223, 0.03094416, 0.06285521,
       0.06959133, 0.11672899, 0.0453184 , 0.08273593, 0.08901193,
       0.07645081, 0.02119846, 0.00610686, 0.02668782, 0.07849718,
       0.03905549, 0.09677198, 0.12684426, 0.06638764, 0.08273705,
       0.10037155, 0.07731802, 0.09962349, 0.07400749, 0.09592071,
       0.08466747, 0.07204308, 0.10099934, 0.00544467, 0.00507193,
       0.06652689, 0.09516547, 0.13018486, 0.05262307, 0.08439227,
       0.08693082, 0.09670481, 0.11409435, 0.10301092, 0.08114936,
       0.06714662, 0.06911306, 0.11542881, 0.0523585 ]), 'params': [{'n_factors': 10, 're
g_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors
': 10, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002},
{'n_factors': 10, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.002, 'reg_b
i': 0.02}, {'n_factors': 10, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.
002, 'reg_bi': 0.002}, {'n_factors': 10, 'reg_pu': 0.02, 'reg_qi': 0.002, 'biased': True,
'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 10, 'reg_pu': 0.02, 'reg_qi': 0.002, 'bias
ed': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 10, 'reg_pu': 0.02, 'reg_qi':
0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 10, 'reg_pu': 0.02
, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 10, '
reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_fact
ors': 10, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.00
2}, {'n_factors': 10, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.002, '
reg_bi': 0.02}, {'n_factors': 10, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_b
u': 0.002, 'reg_bi': 0.002}, {'n_factors': 10, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased'
: True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 10, 'reg_pu': 0.002, 'reg_qi': 0.0
02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 10, 'reg_pu': 0.002,
'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 10, 'reg
_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_fact
ors': 20, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}
, {'n_factors': 20, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_
bi': 0.002}, {'n_factors': 20, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu':
0.002, 'reg_bi': 0.02}, {'n_factors': 20, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True,
'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 20, 'reg_pu': 0.02, 'reg_qi': 0.002, 'bi
ased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 20, 'reg_pu': 0.02, 'reg_qi':
0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 20, 'reg_pu': 0.02
, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 20, 'r
eg_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_fac
tors': 20, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.0
2}, {'n_factors': 20, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 'r
eg_bi': 0.002}, {'n_factors': 20, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_b
u': 0.002, 'reg_bi': 0.02}, {'n_factors': 20, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased':
True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 20, 'reg_pu': 0.002, 'reg_qi': 0.0
02, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 20, 'reg_pu': 0.002, '
reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 20, 'reg_
pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factor
s': 20, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.00
2}, {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 're
g_bi': 0.02}, {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu':
0.02, 'reg_bi': 0.002}, {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True,
'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi': 0.02, 'bias
ed': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi':
0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 30, 'reg_pu': 0.02,
'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 30, 'reg
_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factor
s': 30, 'reg_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002
}, {'n_factors': 30, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu': 0.02, 're
g_bi': 0.02}, {'n_factors': 30, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_bu'
: 0.02, 'reg_bi': 0.002}, {'n_factors': 30, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': Tr
ue, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 30, 'reg_pu': 0.002, 'reg_qi': 0.02,
'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 30, 'reg_pu': 0.002, 're
g_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 30, 'reg_pu'
: 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors':
30, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {
'n_factors': 30, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_
bi': 0.002}, {'n_factors': 40, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True, 'reg_bu':
0.02, 'reg_bi': 0.02}, {'n_factors': 40, 'reg_pu': 0.02, 'reg_qi': 0.02, 'biased': True,
'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 40, 'reg_pu': 0.02, 'reg_qi': 0.02, 'bias
ed': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 40, 'reg_pu': 0.02, 'reg_qi':
```

```
0.02, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 40, 'reg_pu': 0.02
, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors': 40, 're
g_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_facto
rs': 40, 'reg_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02
}, {'n_factors': 40, 'reg_pu': 0.02, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'r
eg_bi': 0.002}, {'n_factors': 40, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': True, 'reg_b
u': 0.02, 'reg_bi': 0.02}, {'n_factors': 40, 'reg_pu': 0.002, 'reg_qi': 0.02, 'biased': T
rue, 'reg_bu': 0.02, 'reg_bi': 0.002}, {'n_factors': 40, 'reg_pu': 0.002, 'reg_qi': 0.02,
 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.02}, {'n_factors': 40, 'reg_pu': 0.002, 'reg
_qi': 0.02, 'biased': True, 'reg_bu': 0.002, 'reg_bi': 0.002}, {'n_factors': 40, 'reg_pu'
: 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.02}, {'n_factors':
40, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.02, 'reg_bi': 0.002}, {
'n_factors': 40, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu': 0.002, 'reg_
bi': 0.02}, {'n_factors': 40, 'reg_pu': 0.002, 'reg_qi': 0.002, 'biased': True, 'reg_bu':
0.002, 'reg_bi': 0.002}], 'param_n_factors': [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 30, 30
, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40], 'param_reg_pu': [0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.0
2, 0.02, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.02, 0.02, 0.02, 0.02,
0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.02, 0.0
2, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002, 0.002,
0.002, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.002,
0.002, 0.002, 0.002], 'param_reg_qi': [0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002
, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.02, 0.02, 0.02, 0.02, 0.002, 0.00
2, 0.002, 0.002, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.02, 0.02, 0.02, 0.
02, 0.002, 0.002, 0.002, 0.002, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.02,
0.02, 0.02, 0.02, 0.002, 0.002, 0.002, 0.002, 0.02, 0.02, 0.02, 0.02, 0.002, 0.002, 0.002
, 0.002], 'param_biased': [True, True, True, True, True, True, True, True, True, True, Tr
ue, True, True, True, True, True, True, True, True, True, True, True, True, True, True, T
rue, True, True, True, True, True, True, True, True, True, True, True, True, True, True,
True, True, True, True, True, True, True, True, True, True, True, True, True, True, True,
True, True, True, True, True, True, True, True, True], 'param_reg_bu': [0.02, 0.02, 0.002
, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.
02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0
.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002
, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002, 0.02, 0.0
2, 0.002, 0.002, 0.02, 0.02, 0.002, 0.002], 'param_reg_bi': [0.02, 0.002, 0.02, 0.002, 0.
02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002,
0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002
, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.0
02, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0.002, 0.02, 0
.002, 0.02, 0.002, 0.02, 0.002]}
```

Evaluating the model

In this section, we evaluate the model on all of the test dataset ratings, and manually calculate the RMSE and accuracy.

In [ ]:

```
# Fit and test the model
predictions = algo.test(test)

prediction_array = []
for prediction in predictions:
  uid = int(prediction.uid)
  iid = int(prediction.iid)
  r_ui = int(prediction.r_ui)
  est = float(prediction.est)
  prediction_array.append([uid, iid, r_ui, est])

pred = np.array(prediction_array)

# calculating RMSE
rmse = np.sqrt(np.mean((pred[:,2] - pred[:,3]) ** 2))

# calculating accuracy
est_rounded = pred[:,3].astype(int)
accuracy = np.sum(pred[:,2] != est_rounded) / pred.shape[0]
```

```
# print out test metrics
print(f'RMSE: {rmse}; Accuracy: {accuracy}')

# print out some predictions
print('Test dataset ratings vs. model predictions:')
print(np.vstack((pred[:,2], pred[:,3])).T)
```

```
RMSE: 0.8450194768270142; Accuracy: 0.6539
Test dataset ratings vs. model predictions:
[[5.         3.79930586]
 [5.         3.58003842]
 [3.         3.28499177]
 ...
 [3.         3.28001867]
 [4.         3.87734224]
 [4.         2.30047792]]
```

Making recommendations for a user

In [ ]:

```
# do predictions!
num_users = len(train.all_users())
num_items = len(train.all_items())

# read in user and movie data
# (this was not used when making the recommender systems, but it's here so we can get som
e context)
user_data = pd.read_csv('https://raw.githubusercontent.com/tiyu0203/fml/master/u.user', '
|', header=None)
item_data = pd.read_csv('https://raw.githubusercontent.com/tiyu0203/fml/master/u.item', '
|', header=None)

# choose some arbitrary user
uid = 152

# get all of the first user's rated movies
all_ratings = []
for rating in train.all_ratings():
  all_ratings.append([*rating])
all_ratings = np.array(all_ratings)

# get all ids of movies that the user rates
user_rated_movies = all_ratings[all_ratings[:,0] == uid,:]
user_rated_movie_titles = item_data.loc[user_rated_movies[:,1], 1]
user_rated_movie_ratings = user_rated_movies[:,2]
print('Rated movies: ', np.vstack((user_rated_movie_titles, user_rated_movie_ratings)).T
)

# predict highest recommendations (estimated >4.9 for this particular user)
recommendations = []
for i in range(num_items):
  if algo.predict(str(uid), str(i)).est > 4.9:
    recommendations.append(i)
recommendations = np.array(recommendations)
print('Recommended movies: \n', item_data.loc[recommendations, 1])

# show user information as well
print('User: ', user_data.loc[uid,:])
```

```
Rated movies:  [['Ace Ventura: Pet Detective (1994)' 3.0]
 ['Tales From the Crypt Presents: Demon Knight (1995)' 5.0]
 ['Patton (1970)' 3.0]
 ['Jurassic Park (1993)' 3.0]
 ['Juror, The (1996)' 4.0]
 ['Dragonheart (1996)' 3.0]
 ["Preacher's Wife, The (1996)" 3.0]
 ['Madness of King George, The (1994)' 4.0]
 ['Nadja (1994)' 5.0]
 ['Little Women (1994)' 5.0]
 ['To Wong Foo, Thanks for Everything! Julie Newmar (1995)' 4.0]
```

```
['To Wong Foo, Thanks for Everything! Julie Newmar (1995)' 1.0]
['Trainspotting (1996)' 5.0]
['Pulp Fiction (1994)' 4.0]
['Tales from the Hood (1995)' 4.0]
['Four Days in September (1997)' 3.0]
['Thin Man, The (1934)' 3.0]
['Jungle2Jungle (1997)' 2.0]
['Independence Day (ID4) (1996)' 1.0]
['Crow, The (1994)' 4.0]
['From Dusk Till Dawn (1996)' 4.0]
["Jackie Chan's First Strike (1996)" 5.0]
['Sting, The (1973)' 3.0]
['Sleepless in Seattle (1993)' 2.0]
['Last Dance (1996)' 2.0]
['Conan the Barbarian (1981)' 3.0]
['My Life as a Dog (Mitt liv som hund) (1985)' 3.0]
['Henry V (1989)' 3.0]
['Sudden Death (1995)' 4.0]
['Basic Instinct (1992)' 5.0]
['So I Married an Axe Murderer (1993)' 4.0]
['Contempt (M\\E9pris, Le) (1963)' 1.0]
['Wedding Singer, The (1998)' 2.0]
['Private Benjamin (1980)' 3.0]
['Kansas City (1996)' 1.0]
['Quick and the Dead, The (1995)' 3.0]
['Graduate, The (1967)' 3.0]
['Body Snatcher, The (1945)' 3.0]
['Meet Me in St. Louis (1944)' 1.0]
['To Kill a Mockingbird (1962)' 1.0]
['Bob Roberts (1992)' 3.0]
['Baby-Sitters Club, The (1995)' 3.0]
['Basquiat (1996)' 3.0]
['As Good As It Gets (1997)' 3.0]
['Naked Gun 33 1/3: The Final Insult (1994)' 3.0]
['Swingers (1996)' 5.0]
['Sense and Sensibility (1995)' 3.0]
['Mouse Hunt (1997)' 4.0]
['Copycat (1995)' 4.0]
['Legends of the Fall (1994)' 4.0]
['Hoodlum (1997)' 3.0]
['Commandments (1997)' 1.0]
['U Turn (1997)' 3.0]
["Schindler's List (1993)" 4.0]
['GoodFellas (1990)' 4.0]
['Leaving Las Vegas (1995)' 3.0]
['Hugo Pool (1997)' 1.0]
['Jude (1996)' 4.0]
['Shall We Dance? (1996)' 2.0]
['Wishmaster (1997)' 4.0]
['Alien: Resurrection (1997)' 5.0]
['Great Escape, The (1963)' 4.0]
['Wonderland (1997)' 4.0]
['Local Hero (1983)' 3.0]
['Haunted World of Edward D. Wood Jr., The (1995)' 2.0]
['Castle Freak (1995)' 3.0]
['Bride of Frankenstein (1935)' 1.0]
['Desperado (1995)' 4.0]
['Blues Brothers 2000 (1998)' 4.0]
['To Catch a Thief (1955)' 4.0]
['Ice Storm, The (1997)' 2.0]
['Alien (1979)' 2.0]
['Kiss the Girls (1997)' 1.0]
['Promesse, La (1996)' 3.0]
['Delicatessen (1991)' 4.0]
['Johnny Mnemonic (1995)' 3.0]
['Birdcage, The (1996)' 2.0]
['Clerks (1994)' 4.0]
['Threesome (1994)' 1.0]
['Star Wars (1977)' 3.0]
['Frighteners, The (1996)' 3.0]
["Mr. Holland's Opus (1995)" 4.0]
['In the Line of Duty 2 (1987)' 2.0]
['Fly Away Home (1996)' 3.0]
```

```
['Fly Away Home (1996)' 3.0]
["Antonia's Line (1995)" 5.0]
['Blood & Wine (1997)' 2.0]
['Cinema Paradiso (1988)' 3.0]
['Desperate Measures (1998)' 2.0]
["April Fool's Day (1986)" 4.0]
['Young Frankenstein (1974)' 4.0]
['Twilight (1998)' 4.0]
['Cool Runnings (1993)' 2.0]
['Citizen Kane (1941)' 3.0]
['Maya Lin: A Strong Clear Vision (1994)' 2.0]
['Four Weddings and a Funeral (1994)' 5.0]
["Ulee's Gold (1997)" 4.0]
['Batman Returns (1992)' 3.0]
['Frisk (1995)' 3.0]
['Unforgiven (1992)' 4.0]
['Manon of the Spring (Manon des sources) (1986)' 2.0]
['Tom & Viv (1994)' 1.0]
['Ridicule (1996)' 3.0]
['Remains of the Day, The (1993)' 3.0]
['Lawnmower Man, The (1992)' 3.0]
['Dial M for Murder (1954)' 5.0]
['Seventh Seal, The (Sjunde inseglet, Det) (1957)' 4.0]
['Right Stuff, The (1983)' 5.0]
['Net, The (1995)' 3.0]
['Blade Runner (1982)' 2.0]
['Angels and Insects (1995)' 4.0]
['Cape Fear (1991)' 4.0]
['Liar Liar (1997)' 4.0]
['Bananas (1971)' 5.0]
['Bean (1997)' 4.0]
['Shadowlands (1993)' 4.0]
['Showgirls (1995)' 3.0]
['Simple Wish, A (1997)' 2.0]
["Someone Else's America (1995)" 4.0]
['Ninotchka (1939)' 3.0]
['Steel (1997)' 3.0]
['Crossing Guard, The (1995)' 4.0]
['Blue Chips (1994)' 2.0]
['Hudsucker Proxy, The (1994)' 2.0]
['Courage Under Fire (1996)' 4.0]
['Three Wishes (1995)' 2.0]
['When Harry Met Sally... (1989)' 2.0]
['Phenomenon (1996)' 5.0]
['Murder at 1600 (1997)' 3.0]
['Big Night (1996)' 3.0]
['Ghost and the Darkness, The (1996)' 2.0]
['Flintstones, The (1994)' 4.0]
['Homeward Bound: The Incredible Journey (1993)' 5.0]
["Romy and Michele's High School Reunion (1997)" 2.0]
['Supercop (1992)' 2.0]
['Jaws 3-D (1983)' 3.0]
['Mystery Science Theater 3000: The Movie (1996)' 4.0]
['Bogus (1996)' 4.0]
['Welcome to the Dollhouse (1995)' 4.0]
['Bad Boys (1995)' 3.0]
['Day the Earth Stood Still, The (1951)' 3.0]
['Like Water For Chocolate (Como agua para chocolate) (1992)' 4.0]
['Jaws (1975)' 4.0]
['Treasure of the Sierra Madre, The (1948)' 3.0]
['Davy Crockett, King of the Wild Frontier (1955)' 4.0]
['Shaggy Dog, The (1959)' 3.0]
['Bad Moon (1996)' 4.0]
['Flipper (1996)' 4.0]
['Star Trek III: The Search for Spock (1984)' 5.0]
['Aliens (1986)' 2.0]
['187 (1997)' 4.0]
['Theodore Rex (1995)' 5.0]
['In the Mouth of Madness (1995)' 3.0]
['Candidate, The (1972)' 4.0]
['Big Blue, The (Grand bleu, Le) (1988)' 2.0]
['Wolf (1994)' 4.0]
['Mad Love (1995)' 3.0]
```

```
 ['Mad Love (1995)' 3.0]
 ['Apple Dumpling Gang, The (1975)' 4.0]
 ['Bonnie and Clyde (1967)' 2.0]
 ['Evil Dead II (1987)' 4.0]
 ['Air Bud (1997)' 2.0]
 ['Braveheart (1995)' 3.0]
 ['Pink Floyd - The Wall (1982)' 4.0]
 ["Carlito's Way (1993)" 2.0]
 ['Willy Wonka and the Chocolate Factory (1971)' 5.0]
 ['Wizard of Oz, The (1939)' 5.0]
 ['Sabrina (1995)' 3.0]
 ['Kiss Me, Guido (1997)' 3.0]
 ['3 Ninjas: High Noon At Mega Mountain (1998)' 4.0]
 ['Apocalypse Now (1979)' 3.0]
 ['Jumanji (1995)' 4.0]
 ['Hunt for Red October, The (1990)' 4.0]
 ['Brazil (1985)' 4.0]
 ['Sword in the Stone, The (1963)' 5.0]
 ['First Kid (1996)' 4.0]
 ['House of Yes, The (1997)' 4.0]
 ['Game, The (1997)' 1.0]]
Recommended movies:
 12                Mighty Aphrodite (1995)
19             Angels and Insects (1995)
22                    Taxi Driver (1976)
50             Legends of the Fall (1994)
59             Three Colors: Blue (1993)
                           ...
1639               Eighth Day, The (1996)
1642                    Angel Baby (1995)
1645                Men With Guns (1997)
1650          Spanish Prisoner, The (1997)
1651     Temptress Moon (Feng Yue) (1996)
Name: 1, Length: 145, dtype: object
User:  0        153
1         25
2          M
3    student
4      60641
Name: 152, dtype: object
```

Stretch goal #1

> **Implement non-negative matrix factorization (NMF) using alternating least squares (ALS)**

I.e., solve:

$$\min_{q,p} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2$$
$$+ \lambda(||q_i||^2 + ||p_u||^2)$$

(assuming q_i and p_u are column vectors)

Do this by minimizing q_i^T while keeping p_u fixed and vice versa and repeating until convergence.

See: https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea

As a matrix problem (ridge regression):

$$\min_{P_u,Q_i} (R_{ui} - P_u Q_i^T)^T (R_{ui} - P_u Q_i^T)$$
$$+ \lambda_u ||P_u||^2 + \lambda_i ||Q_i||^2$$

Each row of Q_i and P_u is a item/user. (Let f denote latent space dimension, i denote the number of items, and u denote the number of users.)

$$P_u \in M_{u\times f}$$
$$Q_i \in M_{i\times f}$$

$$R_{ui} \in M_{u\times i}$$

Update rules:

$$p_u^T \leftarrow r_u^T Q_i (Q_i^T Q_i + \lambda_u I_f)^{-1}$$
$$q_i^T \leftarrow r_i^T P_u (P_u^T P_u + \lambda_u I_f)^{-1}$$

(Here r_u and r_i are also column vectors, i.e., r_u is the transpose of the uth row of R_{ui}, and r_i is the ith row of R_{ui}.)

Model

This performs ALS as an algorithm for NMF. We did not implement a learned bias (this is similar to using `biased=False` in the Surprise library).

`__init__`

Initializes the model. Generates the (sparse) R_{ui} matrix, and initializes P_u and Q_i using random normal distributions.

Parameters:

- `dataset`**: training dataset; tuples of `(user, item, rating)` in the format of the Surprise training dataset**
- `n_factors`**: latent dimension (default 15)**
- `lambda_u`**: L2 regularization coefficient for `P_u` (default 0.02)**
- `lambda_i`**: L2 regularization coefficient for `Q_i` (default 0.02)**

`loss`

Calculates the loss on the training dataset. Used during training.

`update`

Performs the update rule on P_u (once for every user) and Q_i (once for every item). (This is considered one epoch).

By fixing Q_i, we can solve optimally for P_u (since this is a (quadratic) ridge regression problem), and vice versa. However, R_{ui} is sparse and we only want to train on the rated items (otherwise we would be training users to give most movies a zero rating). To accomplish this, we can only train on a single user at a time, only using the vector of movies that the user rates and "filtering" the items matrix Q_i to those same items. The same applies when solving for Q_i.

`train`

Performs the update rule `epoch` times, and reports the training loss.

Parameters:

- `epochs`**: number of epochs to train**

`predict`

Predict the rating for a given user and item.

Parameters:

- `user`: (integral) id of user to predict
- `item`: (integral) id of item to predict

`predict_all`

Predict the ratings for every user for every item (i.e., estimate the full, dense R_{ui}).

`test`

Predict the ratings given user, item pairs.

Parameters:

- `dataset`: **Test dataset, tuples of** `(user, item, rating)`

In [ ]:

```
class ALS_NMF():

    # assumes dataset in the same format as the one given by the surprise library
    def __init__(self, dataset, n_factors=15, lambda_u=0.02, lambda_i=0.02):
        self.user_count, self.item_count = 0, 0
        for _ in dataset.all_users(): self.user_count += 1
        for _ in dataset.all_items(): self.item_count += 1

        self.R_ui = np.zeros((self.user_count, self.item_count))

        self.ratings = []
        for rating in dataset.all_ratings():
            self.ratings.append(rating)
            self.R_ui[rating[0], rating[1]] = rating[2]

        self.ratings = np.array(self.ratings, dtype=np.int)
        self.N = self.ratings.shape[0]

        # initializes P_u and Q_i
        self.P_u = np.random.normal(size=(self.user_count, n_factors))
        self.Q_i = np.random.normal(size=(self.item_count, n_factors))

        self.n_factors = n_factors

        self.lambda_u = lambda_u
        self.lambda_i = lambda_i

    # calculate loss (to check if update rule works)
    def loss(self):
        losses = np.zeros(self.N)
        for j, rating in enumerate(self.ratings):
            u, i, r = rating
            u, i = int(u), int(i)
            losses[j] = (r - self.P_u[u, :] @ self.Q_i[i, :].T) ** 2

        return np.mean(losses) + self.lambda_u*np.linalg.norm(self.P_u) + self.lambda_i*
np.linalg.norm(self.Q_i)

    # update rule
    def update(self):
        # assume Q_i is fixed, update P_u
        # loop through each user
        for u in range(self.user_count):
            # find the items that this user has rated
            unfiltered_user_ratings = self.R_ui[u, :]
            user_rated_items = unfiltered_user_ratings > 0

            # "filtered" things
            r_u = unfiltered_user_ratings[user_rated_items]
            Q_i = self.Q_i[user_rated_items, :]

            # convex quadratic optimal soln
            # p_u^T = r_u @ Q_i @ (Q_i^T @ Q_i + lambda_u * I_f)^-1
            self.P_u[u, :] = r_u @ Q_i @ np.linalg.pinv(Q_i.T @ Q_i + self.lambda_u * np
.eye(self.n_factors))

        # assume P_u is fixed, update Q_i
        # loop through each item
        for i in range(self.item_count):
            # find the users that rated this item
            unfiltered_item_ratings = self.R_ui[:, i].T
            users_rating_this_item = unfiltered_item_ratings > 0

            # "filtered" things
            r_i = unfiltered_item_ratings[users_rating_this_item]
            P_u = self.P_u[users_rating_this_item, :]

            # convex quadratic optimal soln
            self.Q_i[i, :] = r_i @ P_u @ np.linalg.inv(P_u.T @ P_u + self.lambda_i * np.
eye(self.n_factors))

    # train method
    def train(self, epochs):
        print(f'Initial loss: {self.loss()}')
        for i in range(epochs):
            self.update()
            # for j, rating in enumerate(self.ratings):
            #     self.update(rating)
            print(f'Epoch: {i}; Loss: {self.loss()}')

    # evaluate
    # r_ui = p_u^T @ q_i
    def predict(self, user, item):
        if user.item >= self.user_count or i >= self.item_count:
            est = np.random.random() * 5
        else:
            est = self.P_u[user, :] @ self.Q_i[item, :].T
        return est

    # predict cross product of all ratings for training set
    def predict_all(self):
        return self.P_u @ self.Q_i.T

    # this assumes that the test dataset is a list of tuples (unlike the train dataset)
    def test(self, dataset):
        ratings = []
        for u, i, r in (dataset if isinstance(dataset, list) else dataset.all_ratings())
:
            ratings.append([self.predict(int(u), int(i)), r])
        return np.array(ratings)
```

Choose the parameters and train the model

When choosing the lambdas, if we chose a higher number the loss would converge to a much larger number; if we chose a small number, the loss would converge to a really small number as well as decrease in a steeper exponential decay. We left `n_factors` the same as the default from the NMF function in the surprise library. After testing a variety of numbers for epochs, we realized that the loss converged relatively quickly so we wouldn't need many epochs.

In [ ]:

```
# choose regularization coefficient
lambda_u = 0.02
lambda_i = 0.02

# choose f
n_factors = 15
```

```
# choose epochs
epochs = 50

# run
model = ALS_NMF(train, n_factors, lambda_u, lambda_i)
model.train(epochs)

#Regularization coefficient for the p_u (user) matrix
P_u = model.P_u()
#Regularization coefficient for the q_i (item/movie) matrix
Q_i = model.Q_i()

print(f'P_u: {P_u}, Q_i: {Q_i}')
```

```
Initial loss: 34.80673227663466
Epoch: 0; Loss: 11.626744968095991
Epoch: 1; Loss: 7.1488177789313525
Epoch: 2; Loss: 6.855198026518326
Epoch: 3; Loss: 6.710935612941744
Epoch: 4; Loss: 6.60877224014469
Epoch: 5; Loss: 6.537936217060015
Epoch: 6; Loss: 6.484704692789716
Epoch: 7; Loss: 6.441820974992813
Epoch: 8; Loss: 6.40476382115052
Epoch: 9; Loss: 6.371439849812138
Epoch: 10; Loss: 6.343120364076325
Epoch: 11; Loss: 6.317881711400218
Epoch: 12; Loss: 6.293993108763518
Epoch: 13; Loss: 6.270658686126371
Epoch: 14; Loss: 6.249897929032098
Epoch: 15; Loss: 6.230931581722405
Epoch: 16; Loss: 6.215299061388394
Epoch: 17; Loss: 6.20149670735049
Epoch: 18; Loss: 6.189626131529177
Epoch: 19; Loss: 6.1799425362648055
Epoch: 20; Loss: 6.172100212112769
Epoch: 21; Loss: 6.165451455079193
Epoch: 22; Loss: 6.159029255519029
Epoch: 23; Loss: 6.153080408069977
Epoch: 24; Loss: 6.147599787424625
Epoch: 25; Loss: 6.142599610421463
Epoch: 26; Loss: 6.137694837852222
Epoch: 27; Loss: 6.132855586045467
Epoch: 28; Loss: 6.127942720259837
Epoch: 29; Loss: 6.123030971053174
Epoch: 30; Loss: 6.118364178585848
Epoch: 31; Loss: 6.114153134429683
Epoch: 32; Loss: 6.110514686016048
Epoch: 33; Loss: 6.107411928208701
Epoch: 34; Loss: 6.104623013399651
Epoch: 35; Loss: 6.101905470349132
Epoch: 36; Loss: 6.0991549062158725
Epoch: 37; Loss: 6.096329892911716
Epoch: 38; Loss: 6.093217255559823
Epoch: 39; Loss: 6.089669596566827
Epoch: 40; Loss: 6.08577225956736
Epoch: 41; Loss: 6.081678662186059
Epoch: 42; Loss: 6.077578280345212
Epoch: 43; Loss: 6.073496232967704
Epoch: 44; Loss: 6.069378567838467
Epoch: 45; Loss: 6.065248497242837
Epoch: 46; Loss: 6.0610428387872854
Epoch: 47; Loss: 6.057205415951149
Epoch: 48; Loss: 6.053581193456878
Epoch: 49; Loss: 6.05020695722102
```

Calculating the RMSE and Accuracy for Test dataset

Our RMSE on the test dataset is low, but the accuracy is no better than random guessing. We were not able to diagnose why this is the case. Increasing the regularization parameters to try to prevent overfitting did not help

diagnose why this is the case. Increasing the regularization parameters to try to prevent overfitting did not help. In the next section we also test on the training dataset and show that does train well, so we are not sure why it does not generalize past the training dataset.

In [2]:

```
test_res = model.test(test)
rmse = np.sqrt(np.mean((test_res[:, 0] - test_res[:, 1]) ** 2))
accuracy = np.mean(np.round(test_res[:, 0]) == np.round(test_res[:, 1]))
print(f'RMSE: {rmse}, Accuracy: {accuracy}')
```

```
RMSE: 1.9356349226785399, Accuracy: 0.21745
```

Calculating the RMSE and Accuracy for the Train dataset

As a sanity check, we ran the train dataset in our model to see if there was an issue on how we implemented the algorithm. This indicates that our model does indeed train correctly, so we are not sure why it doesn't generalize.

In [ ]:

```
test_res = model.test(train)
rmse = np.sqrt(np.mean((test_res[:, 0] - test_res[:, 1]) ** 2))
accuracy = np.mean(np.round(test_res[:, 0]) == np.round(test_res[:, 1]))
print(f'RMSE: {rmse}, Accuracy: {accuracy}')
```

```
RMSE: 0.5474335410733868, Accuracy: 0.6905625
```