matrix_factorisation

January 28, 2022

1 Exercise sheet 10: exercise 3

Jaishree Janu

Matrix factorisation using an off-the-shelf library

[29]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from collections import defaultdict

# Surprise: https://surprise.readthedocs.io/en/
import surprise

from surprise.reader import Reader
from surprise import Dataset
from surprise.model_selection import GridSearchCV

# Cross Validation
from surprise.model_selection import cross_validate

# Matrix Factorization Algorithm
from surprise import NMF

np.random.seed(3116)  # replicating results
sns.set_theme(style="whitegrid", palette="pastel")
sns.set(rc={'figure.figsize':(17, 5)})

# Inspired from: https://www.jiristodulka.com/post/recsys_cf/
```

[2]:
```
ratings_df = pd.read_csv("remf/u.data", sep="\t", header=None)
```

[3]:
```
ratings_df.columns = ["userId", "movieId", "rating", "timestamp"]
```

1.1 Data preprocessing

[4]:
```
min_movie_ratings = 2 #a movie has was rated at least
min_user_ratings =  5 #a user rated movies at least


ratings_flrd_df = ratings_df.groupby("movieId").filter(lambda x: x['movieId'].
 ↪count() >= min_movie_ratings)
ratings_flrd_df = ratings_flrd_df.groupby("userId").filter(lambda x:␣
 ↪x['userId'].count() >= min_user_ratings)



"{0} movies deleted; all movies are now rated at least: {1} times. Old␣
 ↪dimensions: {2}; New dimensions: {3}"\
.format(len(ratings_df.movieId.value_counts()) - len(ratings_flrd_df.movieId.
 ↪value_counts())\
        ,min_movie_ratings,ratings_df.shape, ratings_flrd_df.shape )
```

[4]:
```
'141 movies deleted; all movies are now rated at least: 2 times. Old dimensions:
(100000, 4); New dimensions: (99859, 4)'
```

1.2 Data loading via `Surprise`

In this section, we use the Data Processor of Surprise to read our custom movie lens dataset. The library allows us to build train and validation splits similar to sklearn's as demonstrated below.

[5]:
```
reader = Reader(rating_scale=(0.5, 5)) #line_format by default order of the␣
 ↪fields
data = Dataset.
 ↪load_from_df(ratings_flrd_df[["userId",        "movieId",        "rating"]],␣
 ↪reader=reader)

trainset = data.build_full_trainset()

testset = trainset.build_anti_testset()
```

The following code enables us to run NMF on the movielens dataset with 3-fold grid serach CV.

We use the n_factors range: (start=10, stop=100, step=10). The function writes out the RMSE corresponding to n_factors

[26]:
```
def rmse_vs_factors(algorithm, data):
  """Returns: rmse_algorithm i.e. a list of mean RMSE of CV = 3 in␣
 ↪cross_validate() for each  factor k in range(1, 101, 1)
  100 values
  Arg:  i.) algorithm = Matrix factoization algorithm, e.g SVD/NMF/PMF, ii.) ␣
 ↪data = surprise.dataset.DatasetAutoFolds
```

```
    """

    rmse_algorithm = {}

    for k in range(10, 100, 10):
        algo = algorithm(n_factors = k)

        #["test_rmse"] is a numpy array with min accuracy value for each testset
        loss_fce = cross_validate(algo, data, measures=['RMSE'], cv=3,␣
↪verbose=True)["test_rmse"].mean()
        rmse_algorithm[k] = loss_fce

    return rmse_algorithm
```

[20]:
```
rmse_nmf = rmse_vs_factors(NMF, data)
```

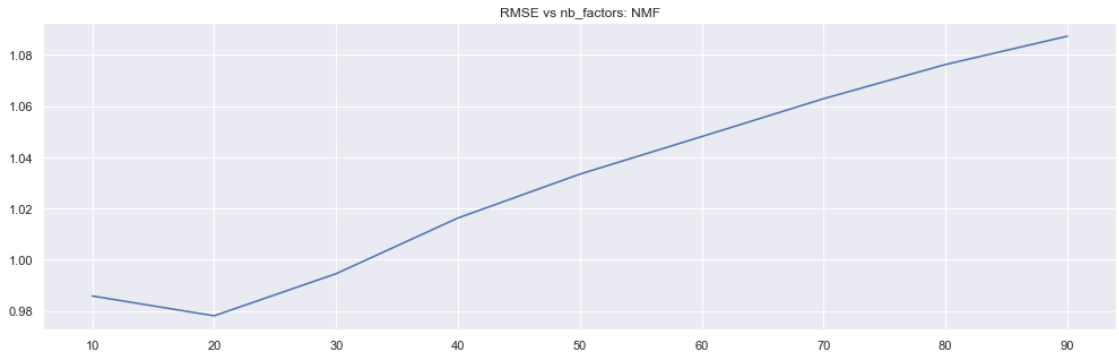
```
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9917  0.9837  0.9823  0.9859  0.0042
Fit time          2.75    2.97    2.67    2.80    0.13
Test time         0.11    0.25    0.11    0.15    0.07
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9770  0.9718  0.9857  0.9782  0.0057
Fit time          3.74    3.57    3.21    3.51    0.22
Test time         0.27    0.11    0.11    0.16    0.08
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0004  0.9925  0.9908  0.9946  0.0042
Fit time          4.17    4.32    4.45    4.31    0.11
Test time         0.29    0.11    0.25    0.22    0.08
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0281  1.0062  1.0146  1.0163  0.0090
Fit time          5.76    5.27    5.68    5.57    0.22
Test time         0.13    0.26    0.13    0.17    0.06
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0394  1.0302  1.0306  1.0334  0.0043
Fit time          6.58    6.18    6.17    6.31    0.19
Test time         0.13    0.26    0.11    0.17    0.07
Evaluating RMSE of algorithm NMF on 3 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0458  1.0511  1.0473  1.0481  0.0023
Fit time          7.73    6.87    6.80    7.13    0.42
Test time         0.25    0.13    0.26    0.21    0.06
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0637  1.0604  1.0644  1.0628  0.0017
Fit time          7.61    7.96    7.84    7.80    0.14
Test time         0.12    0.12    0.25    0.16    0.06
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0735  1.0770  1.0780  1.0762  0.0019
Fit time          7.64    7.99    7.83    7.82    0.14
Test time         0.11    0.26    0.11    0.16    0.07
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0903  1.0961  1.0752  1.0872  0.0088
Fit time          8.31    8.71    8.89    8.64    0.25
Test time         0.28    0.12    0.26    0.22    0.07
```

[30]:
```
plt.plot(rmse_nmf.keys(), rmse_nmf.values())
plt.title("RMSE vs nb_factors: NMF")
```

[30]: `[<matplotlib.lines.Line2D at 0x7ff48245e100>]`

[30]: `Text(0.5, 1.0, 'RMSE vs nb_factors: NMF')`



1.3 NMF using surprise's own movielens interface

- This section is to explain the unusual choice of this *off_the_shelf* library. The library positions itself as Python scikit for recommender systems and hence provides really good tools

and documentation of the learning techniques. As we have observed in the exercise so far, the API is simple to navigate and we can easily do cross validation, as well as expose RMSE results.

- The following code shows an even more concise way of using the inbuilt movielens 1M (one million) dataset of Surprise. This procedure gets us to a similar RMSE level as the previous section with some improvements.

[25]:
```
from surprise import NMF
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-1m')

# Use the famous SVD algorithm.
algo = NMF()

# Run 5-fold cross-validation and print results.
cross_validate(algo, data, measures=['RMSE'], cv=3, verbose=True)
```

```
Dataset ml-1m could not be found. Do you want to download it? [Y/n] y
Trying to download dataset from
http://files.grouplens.org/datasets/movielens/ml-1m.zip…
Done! Dataset ml-1m has been saved to /Users/mean-machine/.surprise_data/ml-1m
Evaluating RMSE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9203  0.9211  0.9200  0.9205  0.0005
Fit time          33.27   34.26   34.18   33.90   0.45
Test time         2.14    2.29    1.85    2.09    0.18
```

[25]:
```
{'test_rmse': array([0.92025963, 0.9211457 , 0.92000087]),
 'fit_time': (33.26960277557373, 34.26421904563904, 34.17604088783264),
 'test_time': (2.1389052867889404, 2.2887229919433594, 1.8460800647735596)}
```