

Recommender Systems: Model-based collaborative filtering

AAA-Python Edition



Plan

- 1- SVD filtering: With Surprise
- 2- SVD Filtering: More details
- 3- Filtering with SVM Classification
- 4- Some Tests
- 5- Predictions with Custom Data: Preparation
- 6- Predictions with Custom Data: Prediction



Prediction

```
from surprise import SVD
from surprise import Dataset

# Load the movielens-100k dataset
myData = Dataset.load_builtin('ml-100k')
trainset = myData.build_full_trainset()
# SVD algorithm.
Recommender = SVD()
Recommender.fit(trainset)
```

Slightly better performance compared with neighborhood filtering

```
print(Recommender.predict("226","527"))
 user: 226
                  item: 527
                                    r ui = None
                                                   est = 4.16
                                                                {'was impossible': False}
            The estimation of the review is
            equal to 4.16
   from surprise.model selection import cross validate
   cross validate(Recommender,myData,cv=5,measures=['RMSE'],verbose =True)
                  Evaluating RMSE of algorithm SVD on 5 split(s).
                                    Fold 1
                                            Fold 2
                                                     Fold 3 Fold 4
                                                                     Fold 5
                                                                                      Std
                                                                              Mean
                                                             0.9423
                  RMSE (testset)
                                    0.9304 0.9304
                                                     0.9417
                                                                      0.9343
                                                                             0.9358
                                                                                      0.0052
                 Fit time
                                    5.88
                                             5.81
                                                     5.79
                                                             5.85
                                                                      5.84
                                                                              5.83
                                                                                      0.03
                                    0.15
                                             0.24
                                                     0.14
                                                             0.14
                                                                      0.14
                                                                              0.16
                                                                                      0.04
                  Test time
[By Amina Delali]
```





Concept

- Make the assumption that there are factors (characteristics)
 related to each item. Each item can be described by the degree of
 the presence of each characteristic in that item. At the same
 time, each user can have different degrees of interest on each of
 those characteristics.
- These two relationships can be modeled by two matrices:
 - $P_{(m,f)}$: models the interests of each user \mathbf{u} in \mathbf{f} characteristics in a row vector: $\mathbf{p}_{\mathbf{u}}$
 - $P = Q_{(n,f)}$: models the extent of presence of each characteristic in an Item i in a row vector \mathbf{q}_i
- The interaction between each user and item is computed by:
 - \rightarrow q_i^T . p_i which could estimate the rating of the user u for the item i
 - The estimation is enhanced by other parameters to explain the bias in ratings: $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \cdot p_u$

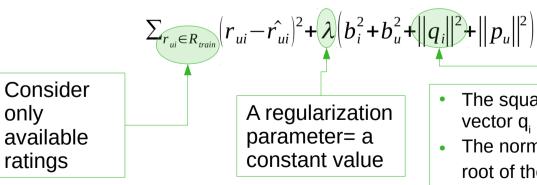
4





Computation

- Singular Value decomposition (SVD) could be used to extract the matrices **P** and **Q**. The values of the ratings could also estimate the bias values with the mean of all the ratings, the mean of the ratings of each user and the mean of the ratings of each item.
- The problem is the fact that not all the ratings of all the users for all the items are available. This is why, we have to find another way to estimate these values.
- The values estimated should minimize the following equation:



- The square of the norm of the vector q_i
- The norm of q_i is the square root of the sum of the squares of q_i values.

[By Amina Delali]





Stochastic Gradient Descent

- The gradient descent is an iterative algorithm that tries to find the (a local) minimum of function. In machine learning, the gradient descent variations algorithms are used to estimate a model's parameters by minimizing a cost function by recursively updating these parameters.
- The SGD (stochastic gradient descent) is a variation in which, in one iteration (epoch), the parameters are updated for each sample (in our case for each rating). So in one epoch the parameters could be updated several times:
 - > The 4 parameters are initialized.
 - For each rating r_{ui} a prediction $\hat{r_{ui}}$ is made and the difference: $e_{ui} = r_{ui} \hat{r_{ui}}$ is computed.
 - \rightarrow Then, the difference e_{ui} is used to update the parameters values as this way:

$$b_{u} \leftarrow b_{u} + \gamma (e_{ui} - \lambda b_{u})$$

$$b_{i} \leftarrow b_{i} + \gamma (e_{ui} - \lambda b_{i})$$

$$p_{u} \leftarrow p_{u} + \gamma (e_{ui} \cdot q_{i} - \lambda p_{u})$$

$$q_{i} \leftarrow q_{i} + \gamma (e_{ui} \cdot p_{u} - \lambda q_{i})$$

rate: another constant that defines the

The learning



2- SVD Filtering: More details

Stochastic Gradient Descent (suite)

- > The process is repeated for a certain number of iterations in order to find a local minimum for the previous equation.
- In Surprise library, the parameters are as follow:
 - The parameters: b_u and b_i (also called **baselines**) are initialized to
 0
 - User and Item factors: p_i and q_i are randomly initialized according to a normal distribution defined by the mean **init_mean** and the standard deviation **init_std_dev** parameters.
 - $\rightarrow \lambda$ (Ir_all) is set by default to 0.02, and γ (reg_all) to 0.005
 - By default the number of factors is 100
 - The number of iterations is by default set to 20 (n_epoch)
 - To use the biases (baselines) parameters, the **biased** parameter is set by default to **True**





etails

Another example with GridSearchCV

Root Mean Square Error

```
from surprise.model selection import GridSearchCV
   param grid = {'n epochs': [5, 10, 20], 'lr all': [0.002, 0.005],
                 'reg all': [0.4, 0.6]}
  myGrid = GridSearchCV(SVD, param grid, measures=['rmse', 'mae'], cv=3)
  myGrid.fit(myData)
                                                                          Mean Absolute Error
 9 # best RMSE, adn MAE scores
10 print("Best RMSE score: %1.2f" % myGrid.best score['rmse'])
   print("Best MAE score: %1.2f" % myGrid.best score(\( \)mae \( \)
13 # The parameters that gave the best RMSE and MAE scores
14 print("Parameters for best RMSE score:", myGrid.best params['rmse'])
15 print("Parameters for best MAE score:" , myGrid.best params['mae'])
16
```





Concept

- The other way to perform a model-based collaborative filtering, is to train a model on user's reviews, and then to use that model to predict new ones for new items.
- In this lesson we will present an implementation using an SVM (Support Vector Machine). Precisely we will use a Linear SVM classifier to predict the new reviews.
- As described in [Xia et al., 2006], there are two ways to consider the problem:
 - Each item represents a class, and training set is the users ratings for each item other than that item.
 - Each user represents a class, and training set is the item's rating according to each user other than that user.
- But, the problem here is that the matrices representing the rating will not be complete. So, we will use default values for missing ratings.

[By Amina Delali]



min

The original data

 We will use the data we already downloaded using **Dataset** module from **Surprise**. But, first, we will access **directly** to the downloaded dataset file, to see its content

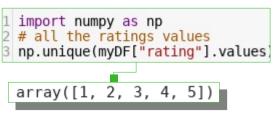
```
# it prints the location of the ratings file
mvData.ratings file

'/root/.surprise_data/ml-100k/ml-100k/u.data'

import pandas as pd

# we will use the location of the ratings file
# to load the data in a DataFrame
theRatingsFile =myData.ratings_file

# the file is organized in 4 columns
myDF = pd.read_csv(theRatingsFile,sep="\t",names =["user_id" ,"item_id" ,"rating" ,"timestamp"])
myDF.head(5)
```



	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116

10

[By Amina Delali]





The features and Labels

- We will apply an SVC classifier for one user, and the classes will be the different ratings.
- We have to construct the **features matrix** corresponding to each item ratings done by the user "226". And construct the the corresponding label vector using the ratings of that user.
- It is more convenient to use the data built by Surprise library, than the original file. the number of features = : 943

```
1 from pandas import DataFrame as DF
2 # the number of the items rated by the user "226"
3 # the corresponding inner id for ther user "226" is 218
4 # it can be found by trainset.to inner uid("226")
5 NI = len(trainset.ur[218])
 print("The number of items rated by the user '226' is:",NI)
  ratedbyU = [trainset.ur[218][i][0] for i in range (NI)]
8 ratesofU = [trainset.ur[218][i][1] for i in range (NI)]
9 # the number of all users
0 NU = trainset.n users ;
  print ("the number of features = :",NU)
  The number of items rated by the user '226' is: 50
                                                                                                11
```



3- Filtering with SVM Classification

The features and Labels (suite)

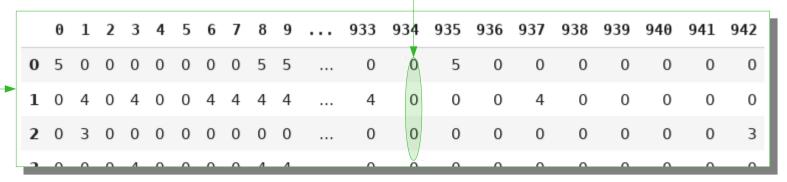
```
myX = np.zeros((NI,NU),dtype = int)
myY = np.array(ratesofU, dtype=int)

# we will fill the myX features matrix
# with the corresponding ratings for each
# user creating new indices for the items
# and keeping the uers inner ids

for (item,newInd) in zip(ratedbyU,range(NI)):
    for j in range(len(trainset.ir[item])):
        userNum = trainset.ir[item][j][0]
        myX[newInd,userNum] = ratesofU[newInd]

myDFX = DF(myX)
myDFX = DF(myY)
#we clearly see how is sparse is the resulting matrix
myDFX.head(5)
```

All these values are unavailable ratings: which mean that the corresponding users didn't rate the corresponding items





3- Filtering with SVM Classification

Prediction for one item

```
# LinearSVC like and SVM classifier (SVC) with # a linear kernel from sklearn.svm import LinearSVC 
myModel = LinearSVC()
myModel.fit(myDFX.values,myY)
```

A linear SVM classifier

All the model we used to predict the ratings for the user of that item, all predicted values either approaching 4 or slightly bigger than 4

After dropping the column corresponding to the user 218 ("226")



myModel.predict(itemDF.values)

array([4])

13

[By Amina Delali]



Splitting the data

- We will just split the data that we have already created using 2 methods:
 - split into test and training sets
 - split into folds (cross-validation)

```
from sklearn.model selection import train test split
   x train, x test,y train, y test = train test split(myDFX.values,myY,test size=0.25)
   print(x test.shape)
                  (13, 942)
# The available labels
print ("All the labels", np.unique(myY))
print("Training lables", np.unique(y train))
print("Testing Labels", np.unique(v test))
      All the labels [1 2 3 4 5]
      Training lables [1 2 3 4 5]
      Testing Labels [2 3 4 5]
```

- We will not run our tests on all the data as in the previous examples.
- We will use only the **50** items related to to the (active) user **"226"**



The prediction with the test, train split





Prediction with cross-validation

```
from sklearn.model selection import cross validate
                                                                           To see the available
from sklearn.metrics import SCORERS
# availabile scoring keys
                                                                           measures (scoring)
SCORERS.keys()
  dict keys(['explained variance', 'r2', 'neg median absolute error', 'neg mean abso
 1 from math import sqrt
 2 theScores =cross validate(myModel,myDFX.values,myY,cv=3,
 scoring = ["neg_mean_squared_error", "neg_mean_absolute_error"])

print( "TEST Negative MSE: ", theScores["test_neg_mean_squared_error"])

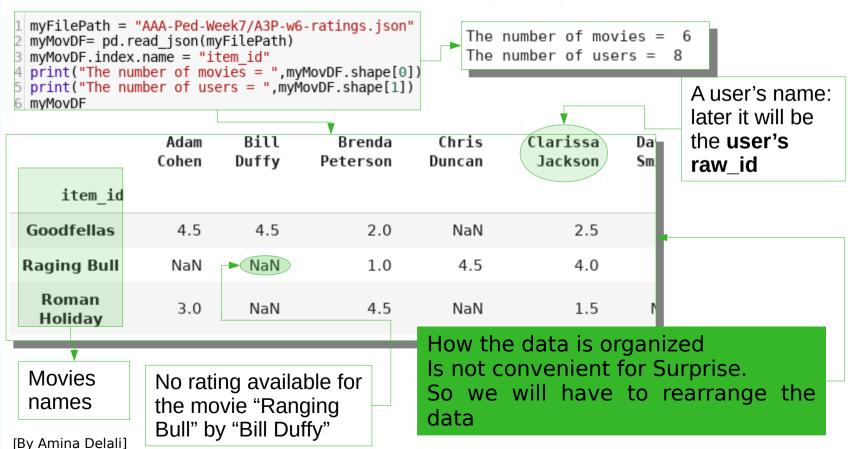
print( "TEST Negative MAE: ", theScores["test_neg_mean_absolute_error"])
 6 print ("Test RMSE mean: %1.2f" % sqrt(np.abs(theScores["test neg mean squared error"]).mean()))
 7 print ("Test MAE mean: %1.2f" % np.abs(theScores["test neg mean absolute error"]).mean())
  TEST Negative MSE: [-0.61111111 -0.94117647 -1.86666667]
  TEST Negative MAE: [-0.61111111 -0.70588235 -0.933333331
  Test RMSF mean: 1.07
  Test MAE mean: 0.75
                            Same results as with Knn collaborative
                            filtering
                                                                                                         16
[By Amina Delali]
```



5-Predictions with Custom Data: Preparation

The data

We will use the data available at :
 Artificial Intelligence with Python GitHub Repository





5-Predictions with Custom Data: Preparation

[By Amina Delali]

Prepare the data

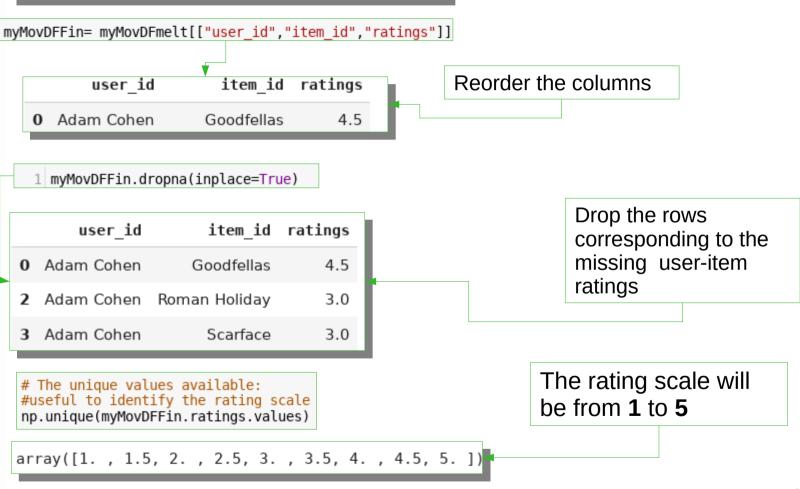
To use with **Surprise**, the dataframe must have the columns organized this way: **user_id**, **item_is** and **ratings**. Which is not the case in our DataFrame.





with -Predictions pa (1)

Prepare the data (suite)





6-Predictions with Custom Data: Prediction

Predict a review for One item

We will use SVD technique to predict the review of the user Adam
 Cohen for the movie Ranging Bull

- If we wanted to use an SVM classifier, we would:
 - Use the original dataframe, and select only the rows corresponding to the movies rated by "Adam"
 - Use the Ranging Bull raw values for prediction
- > The NaN values must be replaced by a default value [By Amina Delali]



6-Predictions with Custom Data: Prediction

Make a list of recommendation

```
# List the movies to recommend to Chris Duncan
# ordred by prediction score
uinId = newTrainSet.to_inner_uid("Chris Duncan")
# number of items rated by "Chris Duncan"
NI = len(newTrainSet.ur[uinId])
print("Number of movies already rated by 'Chris Duncan'=", NI)
nAllItems = newTrainSet.n_items
# items rated by Chris
ChrisItems = [newTrainSet.ur[uinId][i][0] for i in range(NI)]
# remaining Items
toPredItems = [i for i in newTrainSet.all_items() if i not in ChrisItems]
# compute the prediction of unrated items
predictions = np.zeros(len(toPredItems))
```

- The user Chris
 Duncan rated only
 2 movies. We will
 make a list of
 recommendations
 of movies he didn't
 rate by:
 - predicting its reviews on these movies
 - ordering the predicted reviews

21

```
for (item,newInd) in zip(toPredItems, range(len(toPredItems))):
    predictions[newInd]=mySVD2.predict("Chris Duncan",newTrainSet.to_raw_iid(item)).est

indSor=np.argsort(predictions)[::-1]
toPredItems = np.array(toPredItems)
itemsSor = toPredItems[indSor]
predSor = predictions[indSor]

print("\nMovies recommended to Chris: ")
for i in range(len(indSor)):
    print(i+1,"-", newTrainSet.to_raw_iid(itemsSor[i]), " (",np.round(predSor[i],2),")" )
```

Number of movies already rated by 'Chris Duncan'= 2

2 - Goodfellas (3.34) 3 - Scarface (3.33)

1 - Vertigo (3.49)

4 - Roman Holiday (3.21)

Movies recommended to Chris:



References

- [Buitinck et al., 2013] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013).
 - API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pages 108–122.
- [Francesco et al., 2011] Francesco, R., Lior, R., Bracha, S., and Paul B., K., editors (2011). Recommender Systems Handbook.
 Springer Science+Business Media.
- [Hug, 2017] Hug, N. (2017). Surprise, a Python library for recommender systems. http://surpriselib.com.
- [Xia et al., 2006] Xia, Z., Dong, Y., and Xing, G. (2006). Support vector machines for collaborative filtering. In Proceedings of the 44th annual Southeast regional conference, pages 169–174.
 ACM.



Thank you!

FOR ALL YOUR TIME