# 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

1- SVD filtering With Surprise

# 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)
```

```
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 | Mean | Std |
|---|---|---|---|---|---|---|---|
| RMSE (testset) | 0.9304 | 0.9304 | 0.9417 | 0.9423 | 0.9343 | 0.9358 | 0.0052 |
| Fit time | 5.88 | 5.81 | 5.79 | 5.85 | 5.84 | 5.83 | 0.03 |
| Test time | 0.15 | 0.24 | 0.14 | 0.14 | 0.14 | 0.16 | 0.04 |

Slightly better performance compared with neighborhood filtering

[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 **u** in **f** characteristics in a row vector: $\mathbf{p_u}$
  - $Q_{(n,f)}$: 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:
  - $q_i^T . p_u$ 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:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \cdot p_u$$



[By Amina Delali]

# 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:

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda\left(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2\right)$$

Consider only available ratings

A regularization parameter= a constant value

- 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

2- SVD Filtering:
More details

- 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.
    - Then, the difference $e_{ui}$ is used to update the parameters values as this way:

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$$
$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$$
$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$$
$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)$$

The learning rate: another constant that defines the

[By Amina Delali]

## 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.
  - $\lambda$ **(lr_all)** is set by default to **0.02**, and $\gamma$ (**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**



[By Amina Delali]

2- SVD Filtering: More details

# Another example with GridSearchCV

```python
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)

# best RMSE, adn MAE scores
print("Best RMSE score: %1.2f" % myGrid.best_score['rmse'])
print("Best MAE score:  %1.2f" %  myGrid.best_score['mae'])

# The parameters that gave the best RMSE and MAE scores
print("Parameters for best RMSE score:", myGrid.best_params['rmse'])
print("Parameters for best MAE score:" , myGrid.best_params['mae'])
```

**R**oot **M**ean **S**quare **E**rror

**M**ean **A**bsolute **E**rror

```
Best RMSE score: 0.96
Best MAE score:  0.77
Parameters for best RMSE score: {'n_epochs': 20, 'lr_all': 0.005, 'reg_all': 0.4}
Parameters for best MAE score: {'n_epochs': 20, 'lr_all': 0.005, 'reg_all': 0.4}
```

[By Amina Delali]

3- Filtering with SVM Classification

## 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]

# 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
myData.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 |

```
import numpy as np
# all the ratings values
np.unique(myDF["rating"].values)
```

```
array([1, 2, 3, 4, 5])
```

[By Amina Delali]

3- Filtering with SVM Classification

# 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.

```
from pandas import DataFrame as DF
# the number of the items rated by the user "226"
# the corresponding inner id for ther user "226" is 218
# it can be found by trainset.to_inner_uid("226")
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)]
ratesofU = [trainset.ur[218][i][1] for i in range (NI)]
# the number of all users
NU = trainset.n_users ;
print ("the number of features = :",NU)
```

```
the number of features = : 943
```

```
The number of items rated by the user '226' is: 50
```

[By Amina Delali]

3- Filtering with SVM Classification

# The features and Labels (suite)

```python
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)
myDFL = 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

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 933 | 934 | 935 | 936 | 937 | 938 | 939 | 940 | 941 | 942 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 5 | ... | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 4 | 0 | 4 | 0 | 0 | 4 | 4 | 4 | 4 | ... | 4 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| **3** | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 4 | 4 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

```python
# We have to eliminate the column corresponding to the user 218
myDFX=myDFX.drop(axis=1,columns=218)
```

[By Amina Delali]

# 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

```
# construct the features array for the item "393"
# innder_id=528  trainset.to_inner_iid("393")

NIR = len(trainset.ir[528])

itemX =  np.zeros((1,NU),dtype = int)

for j in range(NIR):
    userNum = trainset.ir[528][j][0]
    itemX[0,userNum] = trainset.ir[528][j][1]
itemDF = DF(itemX)
itemDF= itemDF.drop(axis=1,columns=218)
itemDF
```

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”)**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 933 | 934 | 935 | 936 | 937 | 938 | 939 | 940 | 941 | 942 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 4 | 0 | 4 | 3 | 0 | 4 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

```
myModel.predict(itemDF.values)
```

```
array([4])
```

[By Amina Delali]

4- Some Tests

# 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)
```

- 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 available labels
print ("All the labels",np.unique(myY))
print("Training lables",np.unique(y_train))
print("Testing Labels",np.unique(y_test))
```

```
All the labels [1 2 3 4 5]
Training lables [1 2 3 4 5]
Testing Labels [2 3 4 5]
```

[By Amina Delali]

# The prediction with the test, train split

4- Some Tests

```
myModel.fit(x_train,y_train)
myPrediction = myModel.predict(x_test)
myModel.score(x_test,y_test)
```

```
0.23076923076923078
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test,myPrediction)
```

```
array([[0, 1, 0, 0],
       [1, 1, 1, 1],
       [0, 3, 1, 0],
       [0, 0, 3, 1]])
```

```
from sklearn.metrics import classification_report
myCR = classification_report(y_test, myPrediction)
print(myCR)
```

```
              precision    recall  f1-score   support

           2       0.00      0.00      0.00         1
           3       0.20      0.25      0.22         4
           4       0.20      0.25      0.22         4
           5       0.50      0.25      0.33         4

   micro avg       0.23      0.23      0.23        13
   macro avg       0.23      0.19      0.19        13
weighted avg       0.28      0.23      0.24        13
```

The missing label is not represented

[By Amina Delali]

4- Some Tests

# Prediction with cross-validation

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import SCORERS
# availabile scoring keys
SCORERS.keys()
```

To see the available measures (scoring)

```
dict_keys(['explained_variance', 'r2', 'neg_median_absolute_error', 'neg_mean_abso
```

```
from math import sqrt
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"])
print ("Test RMSE mean: %1.2f" % sqrt(np.abs(theScores["test_neg_mean_squared_error"]).mean()))
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 -0.93333333]
Test RMSE mean: 1.07
Test MAE mean: 0.75
```

Same results as with Knn collaborative filtering

[By Amina Delali]

**5-Predictions with Custom Data: Preparation**

# The data

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

```python
myFilePath = "AAA-Ped-Week7/A3P-w6-ratings.json"
myMovDF= pd.read_json(myFilePath)
myMovDF.index.name = "item_id"
print("The number of movies = ",myMovDF.shape[0])
print("The number of users = ",myMovDF.shape[1])
myMovDF
```

```
The number of movies =  6
The number of users =  8
```

| item_id | Adam Cohen | Bill Duffy | Brenda Peterson | Chris Duncan | Clarissa Jackson | Da Sm |
|---|---|---|---|---|---|---|
| Goodfellas | 4.5 | 4.5 | 2.0 | NaN | 2.5 | |
| Raging Bull | NaN | NaN | 1.0 | 4.5 | 4.0 | |
| Roman Holiday | 3.0 | NaN | 4.5 | NaN | 1.5 | N |

A user's name: later it will be the **user's raw_id**

Movies names

No rating available for the movie "Ranging Bull" by "Bill Duffy"

How the data is organized
Is not convenient for Surprise.
So we will have to rearrange the data

[By Amina Delali]

## Prepare the data

5-Predictions with Custom Data: Preparation

- 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.

```
myMovDFind= myMovDF.reset_index()
```

Now, the movies names are in a column

| | item_id | Adam Cohen | Bill Duffy | Brenda Peterson | Chris Duncan |
|---|---|---|---|---|---|
| **0** | Goodfellas | 4.5 | 4.5 | 2.0 | NaN |
| **1** | Raging Bull | NaN | NaN | 1.0 | 4.5 |

```
myMovDFmelt = myMovDFind.melt(id_vars="item_id",var_name="user_id",value_name="ratings")
```

| | item_id | user_id | ratings |
|---|---|---|---|
| **0** | Goodfellas | Adam Cohen | 4.5 |
| **1** | Raging Bull | Adam Cohen | NaN |
| **2** | Roman Holiday | Adam Cohen | 3.0 |

All the users and the corresponding ratings are in 2 columns (wide to long conversion)

[By Amina Delali]

# Prepare the data (suite)

5-Predictions with Custom Data: Preparation

```
myMovDFFin= myMovDFmelt[["user_id","item_id","ratings"]]
```

| | user_id | item_id | ratings |
|---|---|---|---|
| 0 | Adam Cohen | Goodfellas | 4.5 |

Reorder the columns

```
myMovDFFin.dropna(inplace=True)
```

| | user_id | item_id | ratings |
|---|---|---|---|
| 0 | Adam Cohen | Goodfellas | 4.5 |
| 2 | Adam Cohen | Roman Holiday | 3.0 |
| 3 | Adam Cohen | Scarface | 3.0 |

Drop the rows corresponding to the missing user-item ratings

```
# The unique values available:
#useful to identify the rating scale
np.unique(myMovDFFin.ratings.values)
```

```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

The rating scale will be from **1** to **5**

[By Amina Delali]

# 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**

```
from surprise import Reader
myReader = Reader(rating_scale =(1,5))
myNewData = Dataset.load_from_df(myMovDFFin, reader=myReader)
newTrainSet =myNewData.build_full_trainset()
```

Load the data from the dataframe we already prepared.

```
mySVD2 = SVD()
mySVD2.fit(newTrainSet)
```

```
# predict rating for "Ranging Bull" movie by
# the user Adam Cohen
mySVD2.predict("Adam Cohen","Raging Bull")
```

```
Prediction(uid='Adam Cohen', iid='Raging Bull', r_ui=None, est=3.2041813814410713
```

- 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]

# Make a list of recommendation

6-Predictions with Custom Data: Prediction

```
# 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

```
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
```

```
Movies recommended to Chris:
1 - Vertigo  ( 3.49 )
2 - Goodfellas  ( 3.34 )
3 - Scarface  ( 3.33 )
4 - Roman Holiday  ( 3.21 )
```

[By Amina Delali]

# 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