**Module 19**

**Recommendation Systems**

Many large companies, such as Google, Instagram, Spotify, Amazon, Netflix, etc., use recommendation systems to increase user engagement. Spotify, for example, recommends songs similar to those you have liked or listened to so that you will continue listening to music on their platform. Likewise, Amazon recommends products to its users based on the data they have collected on them.

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3548093?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3548093/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3548094?wrap=1)
* [Download Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3548094/download?download_frd=1)

**Glossary**

**Collaborative Filtering**

A method of making automatic predictions about the interests of a user by collecting preferences or taste information from many users

**Content-Based Filtering**

A method of making recommendations based on user preferences for product features

**Funk SVD**

A stochastic gradient descent process developed by Simon Funk that minimizes the error on the known values

**Matrix Factorization**

A type of algorithm that works by decomposing the user-item interaction matrix into the product of two lower-dimensionality rectangular matrices

**SVD**

The abbreviation for singular value decomposition

**Install Surprise:**

Make sure “admin” on Mac!

pip install scikit-surprise

pip install -U --trusted-host pypi.org --trusted-host files.pythonhosted.org scikit-surprise

Manish:

**conda install -c conda-forge scikit-surprise**

**Matilde’s Session:**

**Savio’s Session:**

<https://github.com/kelvins/awesome-mlops>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.linear_kernel.html>

He works for Densu? Marketing purpose: cosine similarity they use

**Notes:**

**Recommendation Systems**

A recommendation engine is a machine learning algorithm that ranks or rates products based on certain criteria. As a general definition, a recommender system is a system that predicts the rating a user will give an item. The predictions are then ranked and returned to the user.

Recommender systems fall into three main categories:

* **Content-based filtering** uses similarities in the features of products, services, and content to make recommendations
* **Collaborative filtering**uses similar users' preferences to provide recommendations to a particular user
* **Hybrid recommender systems** combine multiple recommender strategies to make recommendations, using the advantages of each in different ways

**Content-Based Filtering**

Content-based filtering is a technique that takes advantage of similarities in features to make decisions. Typically, this technique is applied to recommender systems, which are algorithms used to suggest things to users based on their experience.

In this method, a user's interests are compared to the product's features. Those products with the most overlap between their features and user interests will be recommended, as shown in Figure 1 below.

Diagram, schematic

Description automatically generated

r̂ is equal to row (P) ⋅ col (Q). i,𝑗 ij

2 We can give the squared error of a prediction 𝑒𝑖,𝑗, as the difference between

2

(𝑟̂ − 𝑟 )

**Matrix Factorization**

The matrix factorization method generates latent features when multiplied by two entities of different types. Matrix factorization identifies the relationship between items' and users' entities in collaborative filtering. Using the input of users' ratings on the shop items, you hope to predict how the users will rate the items so the users can receive recommendations based on the prediction.

<https://www.youtube.com/watch?v=ZspR5PZemcs>

**Other Matrix Factorization Techniques**

In this module, you have learned some matrix factorization techniques for recommender systems. These techniques discover latent features in users and items. With this method, cold start problems and data sparsity can be reduced. This lesson will introduce you to some other techniques for matrix factorization methods, such as:

* Singular value decomposition (SVD)
* Probabilistic matrix factorization (PMF)
* Non-negative matrix factorization (NMF)
* Bayesian probabilistic matrix factorization (BPMF)

**Singular Value Decomposition (SVD)**

The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize, is a technique for reducing dimensions and creating superior quality recommendations for users. The SVD is commonly used to produce low-rank approximations before computing neighborhoods in collaborative filtering. SVD can also be used in collaborative filtering to discover latent associations between users and items to predict the probability of users selecting certain things.

SVD has also been successful with the following variants:

* FunkSVD
  1. Simon Funk's original algorithm factored the user-item rating matrix as the product of two lower dimensional matrices
* SVD++
  1. The SVD++ algorithm is an optimized SVD algorithm designed to provide implicit feedback to improve prediction accuracy
* Regularized SVD
  1. RSVD is a fast probability-based algorithm that can compute the near-optimal low-rank singular value decomposition of vast amounts of data with high accuracy. The idea of RSVD is to represent the data as compressed to capture the essential information. From this compressed representation, a low-rank singular value decomposition can be derived.
* Iterative SVD
  1. ISVD is used to estimate the singular value decomposition of an incomplete given matrix. ISVD uses first-order optimization over orthogonal manifolds and automatically estimates the rank of SVD. The purpose here is to estimate the singular vectors by optimizing the suitable space, which is the space of the orthogonal matrix manifolds.

**Probabilistic Matrix Factorization (PMF)**

The PMF approach assumes that a small number of unobserved factors determine a user's attitudes and preferences. User preferences are modeled by linearly combining item factor vectors with user-specific coefficients in a linear factor model. This technique has proven successful on huge, sparse, and imbalanced datasets.

**Non-negative Matrix Factorization (NMF)**

In NMF, the principal components of a set of non-negative data vectors are automatically extracted. The principal components can be used to extract significant and sparse features from those vectors.

NMF can reduce prediction errors compared to other techniques, such as SVD. Furthermore, when used in collaborative filtering, the NMF technique always leads to interpretable and sparse decompositions of non-negative matrices.

**Bayesian Probabilistic Matrix Factorization (BPMF)**

BPMF is a model in which capacity is automatically controlled by integrating all model parameters and hyperparameters, thereby allowing it to avoid parameter tuning and providing predictive distribution. Thus, the concept of BPMF is extended to recommendations where top N queries are recommended to users. This allows for more efficient and accurate predictions.

**The SURPRISE Library**

Python's SURPRISE module allows the creation and testing of rate prediction algorithms. Users familiar with the Python machine learning ecosystem should feel at home using this library since it closely resembles the scikit-learn API. SURPRISE provides a set of estimators (or prediction algorithms) for evaluating predictions. In addition, there are implementations of classical algorithms like SVD and NMF and similarity-based algorithms.

Furthermore, the SURPRISE library includes tools for model evaluation, including cross-validation iterators and learned metrics built into scikit-learn, as well as automatic hyperparameter search and grid search for model selection. Finally, a light API allows users to develop their recommendation technique with fewer lines of code.

**Hybrid Recommender Systems**

As you have learned, recommender systems are software tools that generate and present the user with suggested items and other entities based on various strategies. A hybrid recommender system combines multiple recommendation strategies to take advantage of their complementary attributes.

As an example, by combining collaborative and content-based filtering, you may overcome some of the shortcomings faced when each method is used separately. You can implement hybrid recommender system approaches by using content-based and collaborative methods to generate predictions individually, then combining them, or you can simply add the capabilities of collaborative methods to a content-based approach, as shown in Figure 3 below.

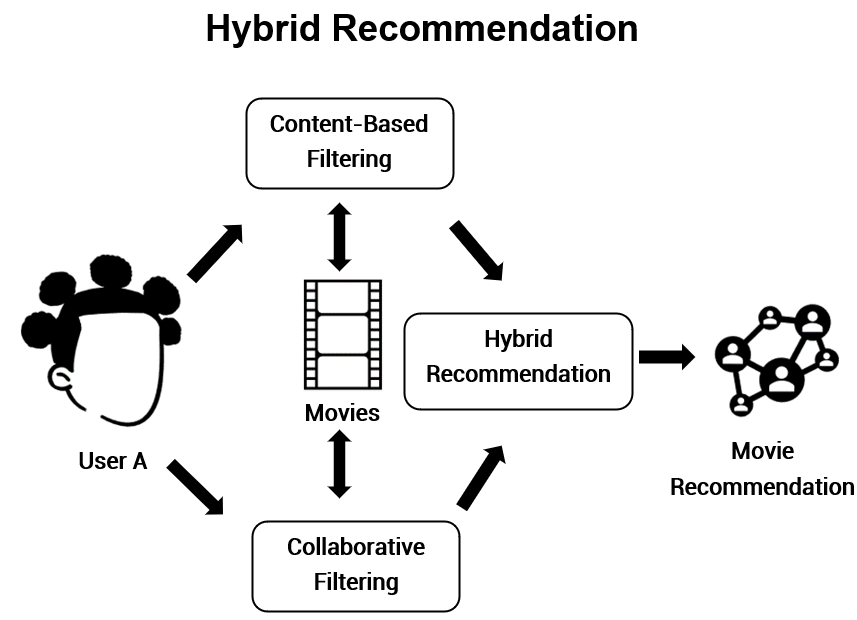


Figure 3

**Module Issues:**

**Codio 19.1 Problem 1**: The description is not clear that prediction must be done for **Dropdead** missing value.

**Codio 19.1 Problem 5**:Truncate output to 2 decimal places for just this entry to pass the hidden test: reviews\_df\_full.loc[['Dropdead'], 'Mandy'] = np.floor(reviews\_df\_full.loc[['Dropdead'], 'Mandy'][0]\*100)/100.0

**Codio 19.2**: It did not grade first so I had to upload the grading script!

**Codio 19.4**: It did not grade first so I had to upload the grading script!

**Codio 19.6 Problem 1**: It does not specify what columns are needed as ’userId', 'title', 'rating’.

**Codio 19.7 Problem 7**: The hidden test generates different result although everything is the same, to by-pass assign these variables:

slope\_one\_preds\_ = slope\_one\_preds

svd\_preds\_ = svd\_preds

**Quizes:**

A recommender system seeks to predict the "rating" or "preference" a user would give an item. : True

*You are correct! The answer “*True*” is correct because recommender systems predict the "rating" or "preference" a user would give an item.*

When recommending new items that users have never engaged with, the platform does not know the ratings in advance. In this instance, what information will the platform use to make recommendations for this item? : Everything it knows about users

*You are correct! The answer “*Everything it knows about users*” is correct because when the platform does not know the ratings in advance, it will attempt to recommend the item based on everything it knows about its users.*

In a recommender system, the item factors are factors that describe the first item. : False

*You are correct! The answer “*False*” is correct because, in a recommender system, the item factors are factors that describe each item.*

Imagine that you are using the item factors “lo-fi\_indie” and “slick\_pop” and the user rating to build a linear regression model. The parameters that you get from linear regression are θ₁ and θ₂. In the context of the recommender system, what are these parameters known as? : User factors

*You are correct! The answer “*User factors*” is correct because the parameters θ₁ and θ₂ measured from linear regression are called user factors.*

In recommender systems terminology, the predictions made by applying linear regression to the item factors and ratings data is given as the dot product of user factors with the item’s factors. : False

*You are correct! The answer “*False*” is correct because the prediction is the dot product of user factors with the item’s factors plus the bias term.*

One big problem in content-based filtering is that in real-world scenarios, the items are not typically categorized into content categories. : True

*You are correct! The answer “*True*” is correct because the problem in content-based filtering is that in the real world, not all items are categorized into content categories.*

In terms of recommender systems, how can the user factors be calculated? : Parameters of linear regression

*You are correct! The answer “*Parameters of linear regression*” is correct because the parameters of the linear regression model that were built using item factors are the user factors.*

To guess the unknown rating from users using collaborative filtering, the algorithm starts with declaring random user factors. : False

*You are correct! The answer “*False*” is correct because the collaborative filtering algorithm starts with declaring random item factors.*

To calculate the user factors for all users from randomly-defined item factors, which model is used? : Linear regression

*You are correct! The answer “*Linear regression*” is correct because the linear regression parameters are the user factors for the collaborative filtering approach.*

To recompute the item factors from the inferred user factors in the collaborative filtering approach, you build a linear regression model using the inferred user factors for each item to get the new inferred item factors. : True

*You are correct! The answer “*True*” is correct because you would build a linear regression model using the inferred user factors for each item to get the new inferred item factors.*

A drawback to the collaborative filtering approach is that it must run through all the item and user factors. : True

*You are correct! The answer “*True*” is correct because the collaborative filtering approach must run through all the item and user factors.*

What is the formula for the predicted rating r^i,j using the item factor matrix and the user factors matrix? : r^i,j= row*i* (P)⋅col*j*(Q)

*You are correct! The answer “*r^i,j= row*i* (P)⋅col*j*(Q)*” is correct because this is the correct formula for predicted ratings.*

The mean squared error formula for collaborative filtering approach is:

MSE=∑i=1M∑jϵRiN(rowi(P)⋅colj(Q)−ri,j)2

The symbol “*Ri*” represents all the items which user *i* has rated. : True

*You are correct! The answer “*True*” is correct because the symbol “Ri” represents all the items which user i has rated.*

Consider the user factors and item factors provided in the tables below.

*Q: Z × N*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Ommazh** | **Melt-Banana** | **BTS** | **Zhou Shen** | **Sanam** |
| F1 | **−**1.53 | **−**1.77 | 1.16 | 1.09 | 0.73 |
| F2 | 1.77 | 1.88 | 1.45 | 1.74 | 1.69 |

*P: M × Z*

|  |  |  |
| --- | --- | --- |
|  | **F1** | **F2** |
| An | **−**1.52 | 1.35 |
| Bhavana | **−**1.51 | 1.11 |
| Cordelia | 1.15 | 2.16 |
| Diego | 1.19 | 2.15 |

Given the user factors and item factors, what would be user “An’s” rating for item “Ommazh”? : −1.53 × −1.52 + 1.77 × 1.35

*You are correct! The answer “*−1.53 × −1.52 + 1.77 × 1.35*” is correct because the user “An’s” rating for item “Ommazh” would be the dot product of Ommazh column and An’s row.*

If the number of users in the data are four and the items in the data are five, considering 50 factors to be formed, what would be the dimensions of user factors matrix *P* and item factors matrix *Q*? : P: 4 × 50

Q: 50 × 5

*You are correct! The answer “*P: 4 × 50

Q: 50 × 5*” is correct because the dimensions for the user factor matrix are “P: M × Z” and the dimensions for the item factors matrix are “Q: Z × N”.*

The algorithm Funk SVD decomposes an M × N matrix into three matrices of sizes M × M, M × N, and N × N. : False

*You are correct! The answer “*False*” is correct because the algorithm Funk SVD decomposes into two matrices of size M × Z and Z × N, where Z is any size of users' choosing.*

Real SVD cannot be used for matrix decomposition because SVD does not have any way to deal with missing entries. : True

*You are correct! The answer “*True*” is correct because the SVD doesn't work with missing values.*

Scikit-learn supports the Funk SVD matrix decomposition algorithm. : False

*You are correct! The answer “*False*” is correct because scikit-learn does not support the Funk SVD matrix decomposition algorithm.*

The constructor used in the surprise.SVD() function to declare the number of factors is “n\_epochs”. : False

*You are correct! The answer “*False*” is correct because the constructor used in the*surprise.SVD()*function to declare the number of factors is “n\_factors”.*

Considering the SVD algorithm in the Python library, is the given statement correct? : Incorrect

*You are correct! The answer “*Incorrect*” is correct because the function*model.test()*gives results only for test sets. It does not work on objects of type training set.*

In an SVD model built using the SURPRISE library, the statement model.pu is used to get the user factors and the statement model.qi is used to get the item factors. : True

*You are correct! The answer “*True*” is correct because the statement*model.pu*is used to get the user factors and the statement*model.qi*is used to get the item factors.*

What does the given Python statement provide?

model.pu @ model.qi.T : Predictions

*You are correct! The answer “*Predictions*” is correct because the given statement provides the predictions of the model by dot product of user factors with the transpose of the item factors.*

Which of the following can be considered metrics for recommender systems? (*Check all that apply.) :* Promoted by advertisers, Items that are popular on the service, Predicted to have high ratings

*You are correct! The answers “*Predicted to have high ratings,*” “*Promoted by advertisers,*” and “*Items that are popular on the service*” are correct because these are considered the metrics for recommender systems.*

What are the systems that use a combination of metrics for recommendation called? : Hybrid recommender systems

*You are correct! The answer “*Hybrid recommender systems*” is correct because the systems that use a combination of metrics for recommendation are called hybrid recommender systems.*

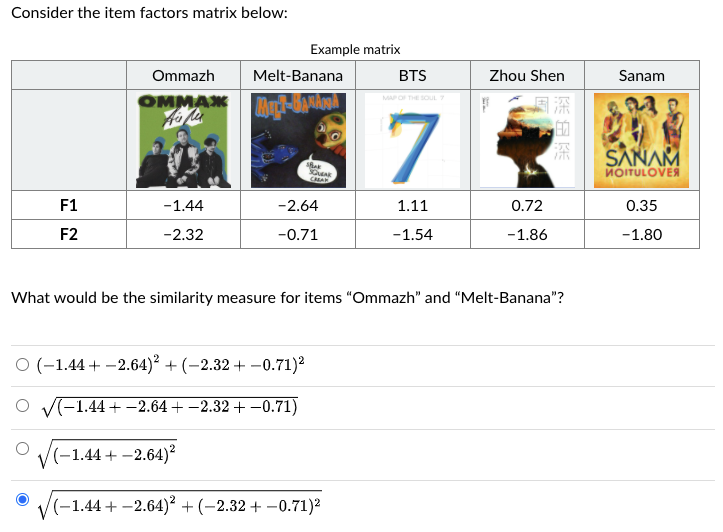
Consider an item factors matrix that tells you there are two factor values for five different items. What does Z equal in this instance? : 2

*You are correct! The answer “*2*” is correct because the value Z that represents the dimensions is equal to the number of factors.*

Consider the item factors matrix below:

Example matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Ommazh | Melt-Banana | BTS | Zhou Shen | Sanam |
|  |  |  |  |  |  |
| **F1** | −1.44 | −2.64 | 1.11 | 0.72 | 0.35 |
| **F2** | −2.32 | −0.71 | −1.54 | −1.86 | −1.80 |



What would be the similarity measure for items “Ommazh” and “Melt-Banana”?

*You are correct! The answer “sqrt(*(−1.44+−2.64)^2+(−2.32+−0.71)^2)*” is correct because the similarity is simply the distance between the two points in Z dimensional space where Z is equal to the number of factors.*

**Try-It Activity 19.1: Building a Recommender System with SURPRISE - Section B**

**Introduction to Dataset**

**ratings.csv** in ml-latest-small file set taken from [grouplens](https://grouplens.org/datasets/movielens/) describes 5-star rating and free-text tagging activity from MovieLens. It contains **100836** ratings across 9742 movies were created by 610 users between March 29, 1996 and September 24, 2018 at random for inclusion who had rated at least 20 movies.

**Ratings Data File Structure (ratings.csv)**

Each line of this file represents one rating of one movie by one user, and has the following format:

userId,movieId,rating,timestamp

The lines within this file are ordered first by userId, then, within user, by movieId.

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

User Ids have been anonymized but they are consistent across files.

Movie Ids are consistent across files and with those used on the MovieLens web site (e.g., id 1 corresponds to the URL <https://movielens.org/movies/1>). Only movies with at least one rating or tag are included in the dataset.

**Exploratory Data Analysis**

No preprocessing is needed, *rating* in the range of 0 and 5, the histogram as follows:

Chart, histogram

Description automatically generated

Dataset were prepared for model executions:

# set rating scale to 0 - 5

reader = Reader(rating\_scale=(0, 5))

# reformat data

data = Dataset.load\_from\_df(df[['movieId', 'userId', 'rating']], reader)

# build train and test sets

train = data.build\_full\_trainset()

test = train.build\_testset()

KNNBasic, SVD, NMF, SlopeOne, and CoClustering were built, fit, predicted and cross-validated:

%%time

knnb = KNNBasic(random\_state = 93)

knnb.fit(train)

# predictions

knnb\_preds = knnb.test(test)

# cross validations

knnb\_cross = cross\_validate(knnb, data, measures=['RMSE'], verbose=True)

%%time

fsvd = SVD(random\_state = 93)

fsvd.fit(train)

# predictions

fsvd\_preds = fsvd.test(test)

# cross validations

fsvd\_cross = cross\_validate(fsvd, data, measures=['RMSE'], verbose=True)

%%time

nmf = NMF(random\_state = 93)

nmf.fit(train)

# predictions

nmf\_preds = nmf.test(test)

# cross validations

nmf\_cross = cross\_validate(nmf, data, measures=['RMSE'], verbose=True)

%%time

sone = SlopeOne()

sone.fit(train)

# predictions

sone\_preds = sone.test(test)

# cross validations

sone\_cross = cross\_validate(sone, data, measures=['RMSE'], verbose=True)

%%time

cocl = CoClustering(random\_state = 93)

cocl.fit(train)

# predictions

cocl\_preds = cocl.test(test)

# cross validations

cocl\_cross = cross\_validate(cocl, data, measures=['RMSE'], verbose=True)

# set metrics to display!

grid\_options=['KNNBasic', 'Funk SVD', 'NMF', 'SlopeOne', 'CoClustering']

test\_accs = [knnb\_cross['test\_rmse'].mean(), fsvd\_cross['test\_rmse'].mean(), nmf\_cross['test\_rmse'].mean(),

sone\_cross['test\_rmse'].mean(), cocl\_cross['test\_rmse'].mean()]

elapsed\_times = [sum(knnb\_cross['fit\_time']) + sum(knnb\_cross['test\_time']),

sum(fsvd\_cross['fit\_time']) + sum(fsvd\_cross['test\_time']),

sum(nmf\_cross['fit\_time']) + sum(nmf\_cross['test\_time']),

sum(sone\_cross['fit\_time']) + sum(sone\_cross['test\_time']),

sum(cocl\_cross['fit\_time']) + sum(cocl\_cross['test\_time'])]

# plot accuracy and time elapsed

fig, ax = plt.subplots(1, 2, figsize = (15, 8))

ax[0].plot(grid\_options, test\_accs, '-o', label = 'Testing Accuracy')

ax[0].plot(np.argmax(test\_accs), max(test\_accs), 'ro', markersize = 12, alpha = 0.4, label = 'Best Score')

ax[0].tick\_params(axis='x', rotation=90)

ax[0].set\_xlabel('Models')

ax[0].set\_ylabel('RMSE Accuracy Score')

ax[0].set\_title(f'Models versus RMSE Accuracy Score best @{grid\_options[np.argmax(test\_accs)]}')

ax[0].legend()

# time plot

ax[1].plot(grid\_options, elapsed\_times, '-o', label = 'Elapsed Time')

ax[1].tick\_params(axis='x', rotation=90)

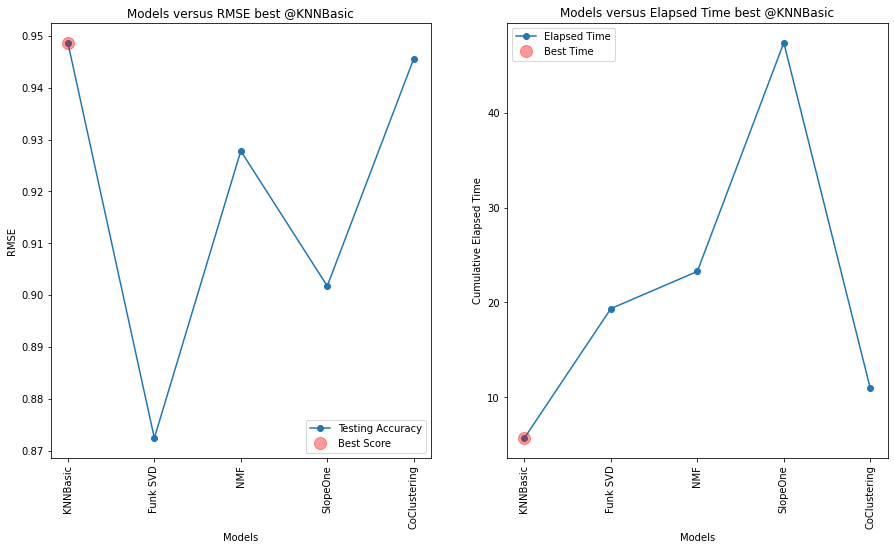
ax[1].set\_xlabel('Models')

ax[1].set\_ylabel('Cumulative Elapsed Time')

ax[1].set\_title(f'Models versus Elapsed Time best @{grid\_options[np.argmin(elapsed\_times)]}')

ax[1].plot(np.argmin(elapsed\_times), min(elapsed\_times), 'ro', markersize = 12, alpha = 0.4, label = 'Best Time')

ax[1].legend()



KNNBasic came with an impressive results, also it is the fastest.

|  |  |  |
| --- | --- | --- |
| **model** | **RMSE** | **elapsed\_time** |
| **KNNBasic** | 0.948712 | 5.704260 |
| **Funk SVD** | 0.872407 | 19.344301 |
| **NMF** | 0.927824 | 23.263689 |
| **SlopeOne** | 0.901739 | 47.370863 |
| **CoClustering** | 0.945613 | 10.935586 |

Table

Description automatically generated

**Fine Tuning Models**

I fine tuned the model to improve above metrics. SlopeOne does not take any parameters, others take number of factors, epochs as well as regulation and biased, number of neighbors and clusters as hyperparameters.

sim\_params = {'name': 'cosine',

'user\_based': False # compute similarities between items

}

knnb2 = KNNBasic(random\_state = 93, k = 50, min\_k = 3, sim\_options = sim\_params)

fsvd2 = SVD(random\_state = 93, n\_factors = 10, n\_epochs = 100, biased = False, lr\_all = 0.015)

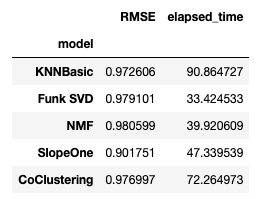
nmf2 = NMF(random\_state = 93, n\_factors = 10, n\_epochs = 100, biased = False, reg\_pu = 0.022, reg\_qi = 0.022)

sone2 = SlopeOne()

cocl2 = CoClustering(random\_state = 93, n\_cltr\_u = 12, n\_cltr\_i = 12, n\_epochs = 100)

Chart, line chart

Description automatically generated



**Conclusion**

SlopeOne cannot be tuned, and it is the slowest algorithm out of the box, hence very easy to use. Remaining all 4 models can be tuned KNNBasic, Funk SVD, NMF, and CoClustering but they become slower with more iterations by n\_epoch. When similarity parameter switched to ‘cosine’ from ‘MSD’ on *KNNBasic* slowed down drastically like 16x, lost its best performing crown. Also, worth to note Funk SVD and NMF prone to overfit when results in *RMSE (testset)* seen greater than one, Funk SVD is the fastest in the fine tuning round.

I think KNNBasic and CoClustering are slightly easier to fine tune versus Funk SVD and NMF, however, *KNNBasic* is the slowest of all while fine tuning but performed the best out of the box both in RMSE and elapsed time metrics. Another observation, *KNNBasic* slows down with increase in number of distinct users. So, SlopeOne is easy to use followed by KNNBasic and CoClustering. Funk SVD and NMF are harder to tune.

RMSE can be higher than 1 that means score difference is more than 1, skews the results!

**Discussion 19.1: A 'Surprising' Recommendation - Section B**

I found a surprising recommendation in Amazon application. This *fuel treatment* item popped up out of blue in “You might also like” section. I have never purchased nor searched for it but I purchased some *air filters* for my cars and my cars are registered in the *garage*. However, the last car-related item purchase was like a year ago.



My guess is I looked up some car wax products on Amazon 6 months ago but did not purchase them. This might have triggered some models in *similar items* category based on what others people’s behavior by either purchasing or searching for. I typically purchase such flammable or liquid items at a local store, again it is not my interest to purchase *fuel treatment* online. I may fit to some customer profile who typically purchase them, this may be one culprit. Secondly, my customer profile information might have sold to Amazon as an outside factor via third-party data providers like The **Dun & Bradstreet** as every person has *dnb* id nowadays. I think third-party data services is the biggest influencer on online business hence in those recommendation systems besides in-house data.

 ————— o —————

**Module 20**

**Notes:**

**Module Issues:**

**Quizes:**

 ————— o —————

**Module 21**

**Notes:**

**Module Issues:**

**Quizes:**