Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

Collaborative Filtering

Prince Grover Follow
Dec 28, 2017 · 10 min read

Comparison of different methods to build recommendation system using collaborative filtering

We see the use of recommendation systems all around us. These systems are personalizing our web experience, telling us what to buy **(Amazon)**, which movies to watch **(Netflix)**, whom to be friends with **(Facebook)**, which songs to listen **(Spotify)** etc. These recommendation systems leverage our shopping/ watching/ listening patterns and predict what we could like in future based on our behavior patterns. The most basic models for recommendations systems are **collaborative filtering models** which are based on assumption that people like things similar to other things they like, and things that are liked by other people with similar taste.

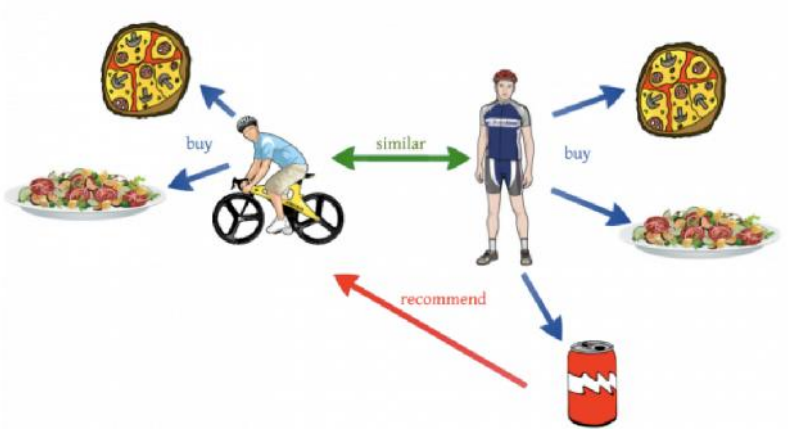

Figure 1: Example of collaborative filtering. Reference: her

It is easy to build targeted recommendation models, then why not just build one for your own customers. I have written this post to make it even more easier for you. Most of the content of this post is inspired from **fast.ai Deep Learning part 1 v2 course.**

Introduction

In this post, I have discussed and compared different collaborative filtering algorithms to predict user rating for a movie. For comparison, I have used MovieLens data which has 100,004 ratings from 671 unique users on 9066 unique movies. **The readers can treat this post as 1-stop source to know how to do collaborative filtering on python and test different techniques on their own dataset.** *(I have also provided my own recommendation about which technique to use based on my analysis).*

Before reading further, I hope that the you have basic understanding of collaborative filtering and its application in recommender systems. If not, I strongly recommend you to go through below blogposts which are written by a fellow student at USF: Shikhar Gupta

Blogs: Collaborative filtering and embeddings—Part 1 and Part 2

Layout of post

- **Types of collaborative filtering techniques**
  - **Memory based**
  - **Model based**
  * Matrix Factorization
  * Clustering
  * Deep Learning
- **Python Implementations**
  - Surprise package
  - fast.ai library
- **Comparison and Conclusions**

Types of collaborative filtering techniques

A lot of research has been done on collaborative filtering (CF), and most popular approaches are based on **low-dimensional factor models** (model based matrix factorization. I will discuss these in detail). The CF techniques are broadly divided into 2-types:

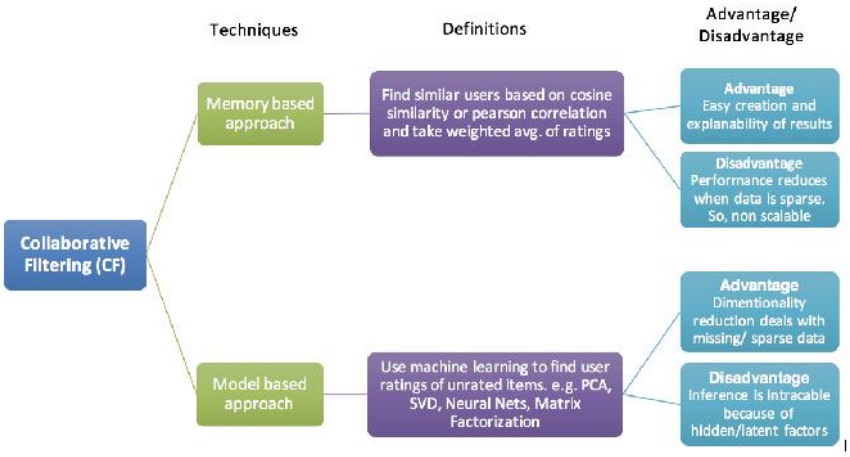

Figure 2: Types of collaborative filtering approaches. Reference: Wikipedia

Say we want to recommend a new item to our product users (e.g. movie to subscribers of Netflix or clothing to online buyer at Amazon). For that we can use one of many techniques as discussed below.

1. Memory based approach:

Quoted by Agnes Johannsdottir in her blogpost

> **Memory-Based Collaborative Filtering** approaches can be divided into two main sections: user-item filtering and item-item filtering. A **user-item filtering** takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. In contrast, **item-item filtering** will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

> Item-Item Collaborative Filtering: “Users who liked this item also liked …”
> User-Item Collaborative Filtering: “Users who are similar to you also liked …”

The key difference of memory-based approach from the model-based techniques *(hang on, will be discussed in next paragraph)* is that we are not learning any parameter using gradient descent (or any other optimization algorithm). The closest user or items are calculated only by using **Cosine similarity or Pearson correlation coefficients**, which are only based on arithmetic operations.
Edit: As stated in above paragraph, the techniques where we don’t use parametric machine learning approach are classified as Memory based

techniques. Therefore, **non parametric ML approaches like KNN (clustering)** should also come under Memory based approach. When I had originally written this blog, I kept it under Model based approach, which is not right.

Quoted in this blog

> A common distance metric is **cosine similarity**. The metric can be thought of geometrically if one treats a given user's (item's) row (column) of the ratings matrix as a vector. For user-based collaborative filtering, two users' similarity is measured as the **cosine of the angle between the two users' vectors**. For users u and u′, the cosine similarity is:

$$sim(u,u') = cos(\theta) = \frac{\mathbf{r}_u \cdot \mathbf{r}_{u'}}{\|\mathbf{r}_u\| \|\mathbf{r}_{u'}\|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{u'i}^2}}$$

We can predict user-u's rating for movie-i by taking weighted sum of movie-i ratings from all other users (u′s) where weighting is similarity number between each user and user-u.

$$\hat{r}_{ui} = \sum_{u'} sim(u,u') r_{u'i}$$

We should also normalize the ratings by total number of u′ (other user's) ratings.

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u,u') r_{u'i}}{\sum_{u'} |sim(u,u')|}$$

Final words on Memory-based approach: As no training or optimization is involved, it is an easy to use approach. But its performance decreases when we have sparse data which hinders scalability of this approach for most of the real-world problems.

If you are interested to try this approach, below are the links of great posts showing step by step python implementation of the same.
(I have not discussed the implementations here as I would personally use scalable model-based approaches)

> **Link 1:** Implementing your own recommender systems in Python
> **Link 2:** Intro to Recommender Systems: Collaborative Filtering

2. Model based approach

In this approach, CF models are developed using machine learning algorithms to predict user's rating of unrated items. As per my understanding, the algorithms in this approach can further be broken down into 3 sub-types.

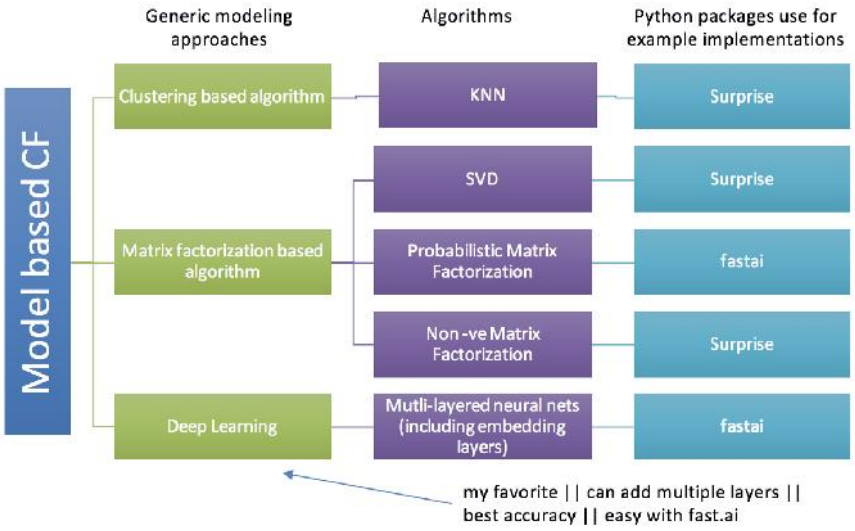

Figure 3. Types of model based collaborative fi tering approaches

Brief explanation of above mentioned algorithms:

- **Matrix Factorization (MF):** The idea behind such models is that attitudes or preferences of a user can be determined by a small number of hidden factors. We can call these factors as **Embeddings.**

> **Matrix decomposition can be reformulated as an optimization problem with loss functions and constraints.** Now the constraints

> are chosen based on property of our model. For e.g. for Non negative matrix decomposition, we want non negative elements in resultant matrices.

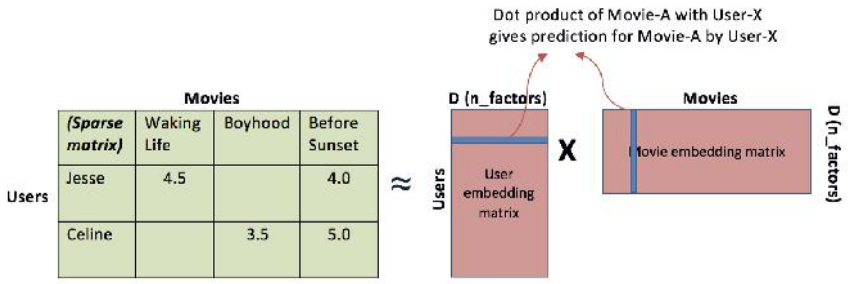

Figure 4. Visualization of matrix factorization

Embeddings:
Intuitively, we can understand embeddings as low dimensional hidden factors for items and users. For e.g. say we have 5 dimensional **(i.e. D or n_factors = 5** in above figure**)** embeddings for both items and users (# 5 chosen randomly). Then for user-X & movie-A, we can say the those 5 numbers **might** represent 5 different characteristics about the movie, like ***(i)*** *how much movie-A is sci-fi intense* ***(ii)*** *how recent is the movie* ***(iii)*** *how much special effects are in movie A* ***(iv)*** *how dialogue driven is the movie* ***(v)*** *how CGI driven is the movie. Likewise, 5 numbers in user embedding matrix might represent,* ***(i)*** *how much does user-X like sci-fi movie* ***(ii)*** *how much does user-X like recent movies …and so on.* In above figure, a higher number from dot product of user-X and movie-A matrix means that movie-A is a good recommendation for user-X.

(I am not saying these numbers actually represent such information. We don't actually know what these factors mean. This is just to build an intuition)

Learn more about embeddings in this great blogpost by another fellow USF student: Kerem Turgutlu.
Link: Structured Deep Learning

Matrix factorization can be done by various methods and there are several research papers out there. In next section, there is python implementation for orthogonal factorization (SVD) or probabilistic factorization (PMF) or Non-negative factorization (NMF).

- **Clustering based algorithm (KNN):** The idea of clustering is same as that of memory-based recommendation systems. In memory-based algorithms, we use the similarities between users

and/or items and use them as **weights** to predict a rating for a user and an item. The difference is that the similarities in this approach are calculated based on an unsupervised learning model, rather than Pearson correlation or cosine similarity. In this approach, we also limit the number of similar users as **k**, which makes system more scalable.

- **Neural Nets/ Deep Learning:** There is a ton of research material on collaborative filtering using matrix factorization or similarity matrix. But there is lack on online material to learn how to use deep learning models for collaborative filtering. **This is something that I learnt in fast.ai deep learning part 1 v2.**

Below is the visualization to explain what is happening when we are using neural nets for this problem.

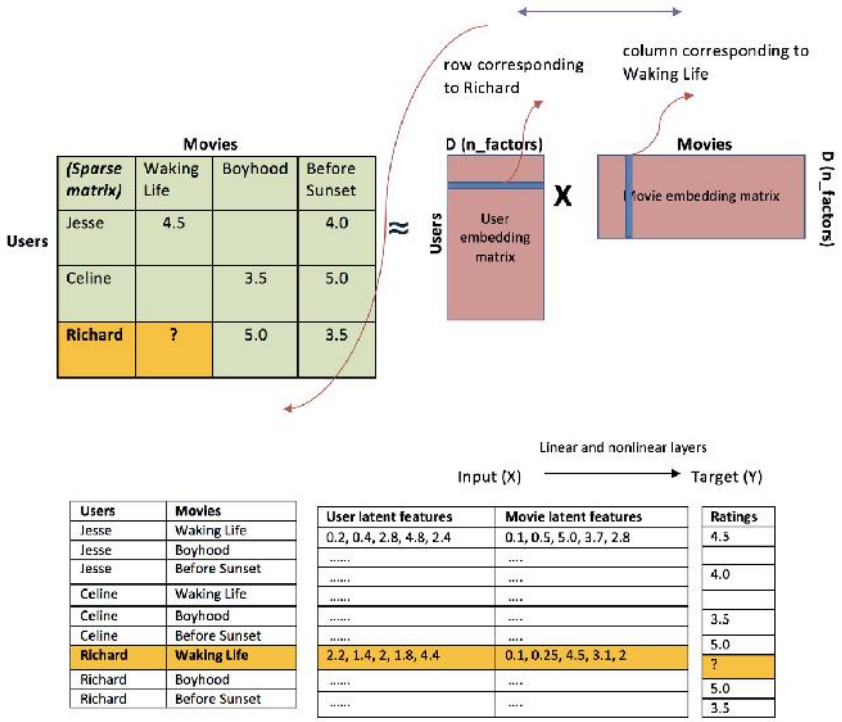

Figure 5. Matrix factorization and embeddings for neural net

We can think of this as an extension to matrix factorization method. For SVD or PCA, we decompose our original sparse matrix into product of 2 low rank orthogonal matrices. For neural net implementation, we don't need them to be orthogonal, we want our model to learn the values of embedding matrix itself. The **user latent features** and **movie latent features** are looked up from the embedding matrices for specific movie-user combination. These are the input values for further linear and non-linear layers. We can pass this input to multiple **relu, linear or**

sigmoid layers and learn the corresponding weights by any optimization algorithm (Adam, SGD, etc.).

Python Implementations

Github repo link: here

Let's look at the python implementation of above discussed **algorithms.** I have explored 2 different python packages which give options of various algorithms to chose from.

(a) Surprise package:

This package has been specially developed to make recommendation based on collaborative filtering easy. It has default implementation for a variety of CF algorithms.

> **Step 1:** Download MovieLens data and read it in pandas df
> http://files.grouplens.org/datasets/movielens/ml-latest-small.zip

> **Step 2:** Install Surprise package by `pip install scikit-surprise.`
> Load the data into `Dataset class`

```
# http://surprise.readthedocs.io/en/stable/getting_started.
# I believe in loading all the datasets from pandas df
# you can also load dataset from csv and whatever suits

ratings = pd.read_csv('ratings_small.csv') # reading data i

from surprise import Reader, Dataset

# to load dataset from pandas df, we need `load_fromm_df` m

ratings_dict = {'itemID': list(ratings.movieId),
                'userID': list(ratings.userId),
                'rating': list(ratings.rating)}
df = pd.DataFrame(ratings_dict)
```

Code chunk 1. Surprise data oader

> **Step 3:** Now implementing any MF algorithm after data preparation is as simple as running 1 line of code. Below are the codes and outputs of Singular Value Decomposition (SVD) and Non negative Matrix Factorization (NMF). The code can also be used for KNN by just changing `algo = KNNBasic()` in below code. (Please check wikipedia links for SVD and NMF)

```
# Split data into 5 folds

data.split(n_folds=5)

from surprise import SVD, evaluate
from surprise import NMF

# svd
algo = SVD()
evaluate(algo, data, measures=['RMSE'])
```

Code chunk 2. SVD and NMF using Surprise

Validation RMSE scores from SVD and NMF

```
Evaluating RMSE of algorithm SVD.          Evaluating RMSE of algorithm NMF.

------------                               ------------
Fold 1                                     Fold 1
RMSE: 0.8990                               RMSE: 0.9476
------------                               ------------
Fold 2                                     Fold 2
RMSE: 0.8983                               RMSE: 0.9449
------------                               ------------
Fold 3                                     Fold 3
RMSE: 0.8941                               RMSE: 0.9479
------------                               ------------
Fold 4                                     Fold 4
RMSE: 0.8962                               RMSE: 0.9494
------------                               ------------
Fold 5                                     Fold 5
RMSE: 0.8962                               RMSE: 0.9450
------------                               ------------
------------                               ------------
Mean RMSE: 0.8967                          Mean RMSE: 0.9469
------------                               ------------
------------                               ------------
```

Figure 6. RMSE scores of SVD and NMF using Surprise

Best RMSE = 0.8967 (SVD) and corresponding MSE is 0.804

(b) fast.ai library:

fast.ai is massive python library that is making machine learning and deep learning easy for people with basic coding skills.

Shallow Learning

Below are 5 lines of codes which implement **probabilistic matrix factorization** for CF. The implementation leverages the fact that the 2 factorized matrices are just embedding matrices which can be modeled by adding embedding layer in neural net (We can call it **Shallow Learning**).

```
ratings = pd.read_csv('ratings_small.csv') # loading data f
"""
ratings_small.csv has 4 columns - userId, movieId, ratings,
it is most generic data format for CF related data
"""

val_indx = get_cv_idxs(len(ratings))  # index for validatio
wd = 2e-4 # weight decay
n_factors = 50 # n_factors - dimension of embedding matrix

# data loader
cf = CollabFilterDataset.from_csv(path, 'ratings_small.csv'
```

Code chunk 3. Collaborative filtering using fast.ai (based on concept of PMF)

Training results:

```
A Jupyter Widget
[ 0.       0.72795  0.80337]
[ 1.       0.75064  0.80203]
[ 2.       0.75122  0.80124]
```

Figure 7. Left: Training MSE, Right: Validation MSE. Rows: Epochs

Best validation MSE obtained from fast.ai implementation of PMF is 0.801 which is close to what we got from SVD.

Deep Learning

We can add more linear and non linear layers to our neural net to make it deep neural net model.

Steps to make a deep neural net for collaborative filtering using fast.ai

> **Step 1:** Load data into PyTorch data-loader. The fast.ai library is built on top of PyTorch. If you want to customize dataset class for specific

format of data, learn it here.

I used fast.ai's `ColumnarModelData.from_data_frame` function to load the dataset.You can also define your own data-loader function. **X** has data about **userId, movieId and timestamp** and **Y** has data about only **ratings (target variable).**

```
x = ratings.drop(['rating'],axis=1)
y = ratings['rating'].astype(np.float32)
data = ColumnarModelData.from_data_frame(path, val_indx, x,
```

Code chunk 4. Data-loader (fast.ai function)

Step 2: Defining custom neural net class (The syntax is specific to PyTorch, but same logic can be employed for Keras also).

We have to create 2 functions. **__init__ (),** the constructor for the class and **forward(),** the forward pass function.

```
# nh = dimension of hidden linear layer
# p1 = dropout1
# p2 = dropout2

class EmbeddingNet(nn.Module):
    def __init__(self, n_users, _n_movies, nh = 10, p1 = 0.
        super().__init__()
        (self.u, self.m, self.ub, self.mb) = [get_emb(*o) f
            (n_users, n_factors), (n_movies, n_factors),
            (n_users,1), (n_movies,1)
        ]]

        self.lin1 = nn.Linear(n_factors*2, nh)  # bias is T
        self.lin2 = nn.Linear(nh, 1)
        self.drop1 = nn.Dropout(p = p1)
        self.drop2 = nn.Dropout(p = p2)

    def forward(self, cats, conts): # forward pass i.e.  do
                                    # and vector from user

```

Code chunk 5. Neural net on PyTorch

A little more about **layers** that have been used in forward pass:

- `dropout` This layer will drop the activations with given probability parameter. The activations have p1, p2 probabilities to be turned to 0. This is done to reduce overfitting.
- `embedding` This layer creates a lookup table for embeddings corresponding to unique users and unique movies. The values in this layer are updated by back-propagation.
- `linear` linear matrix multiplication with added bias.
- `relu` non-linear layers used.
- `sigmoid` used to restrict predicted ratings b/w minimum and maximum rating from training data.

> **Step 3:** Model fitting and prediction.

```
# n_users: count unique users (671), n_movies: count unique
model = EmbeddingNet(n_users, n_movies)

# model.parameters() for back-propagation of weights
# lr = 1e-3, weight decay = 1e-5 and using adam optimizer
opt = optim.Adam(model.parameters(), 1e-3, weight_decay=1e-

# fitting model,
fit(model, data, 3, opt, F.mse_loss)
```

Code chunk 6. Training deep neural net (fast.ai function)

Results:

```
[ 0.       0.79631  0.78994]
[ 1.       0.78677  0.79127]
[ 2.      0.7614   0.7906]
```

Figure 8. Left: training MSE, Right: validation MSE. Rows: Epochs

Best validation MSE = 0.7906. This is best among all the models discussed above.

Comparisons and Conclusions

Below is the plot of MSE obtained from different approaches on MovieLens 100k data.

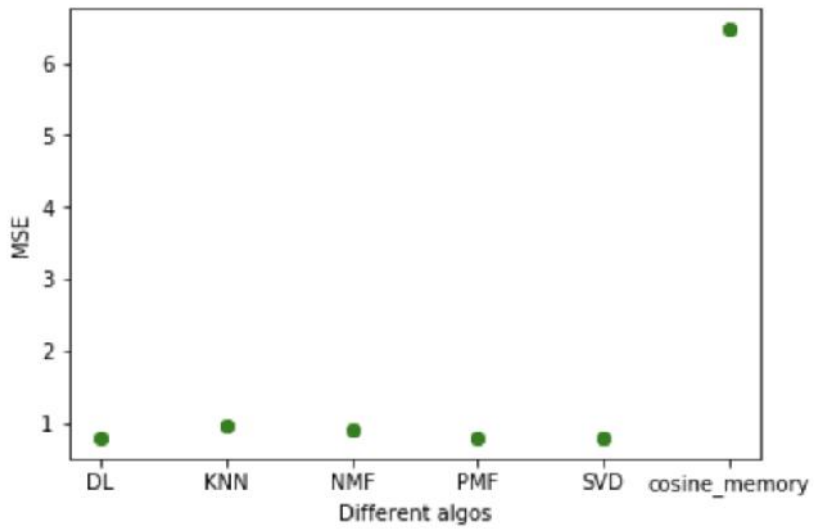

Figure 9. Comparison of MSE scores using different CF methods

Neural net (DL) and SVD give the best results. Neural net implementation will also perform well on imbalanced data, with infrequent users unlike other MF algorithms.

It can be useful to built customer targeted recommendation system for your products/ services. Most easiest and well-researched method out there is collaborative filtering. I have written this post with aim that rather than going through technical research papers and spending hours to learn about collaborative filtering, readers can find all the useful materials, together with implementation at just one place.

References and other useful resources:

1. **My GitHub repo link with python implementations**
2. **Recommender systems**
3. **Collaborative filtering using Fast.ai**
4. **Surprise : A Python scikit for recommender systems**
5. **Collaborative filtering and embeddings—Part 1 and Part 2**
   https://medium.com/@shik1470/63b00b9739ce
   https://medium.com/@shik1470/919da17ecefb
6. **Tutorial on Collaborative Filtering and Matrix Factorization in Python**

7. **Research papers**
http://www.sciencedirect.com/science/article/pii/S1877050915007462
https://www.cs.toronto.edu/~amnih/papers/pmf.pdf