Hands-on Tutorial : Recommendation Algorithms and Diversity

idir.benouaret@epita.fr

1 Introduction

Recommender systems are among the most popular applications of big data analytics. The main goal is to predict whether a user would like a certain item : a product, a movie, a song, etc. In this tutorial, we'll learn how to implement various recommendation algorithms using an open-source recommendation library : Cornac. Then, we will apply diversification algorithms to enhance the diversity of recommendations. We will be using the MovieLens dataset to build movie recommendations.

2 Dataset

The dataset we will use for this tutorial comes from the MovieLens website. It is very often used by researchers and industrials to experiment with a variety of recommender systems and compare their performance. We will use a subset of the dataset : MovieLens 1M Dataset. This dataset contains 1M user ratings for over 4000 movies (ratings.csv). In this file, users and movies are identified with integers, and ratings are integer values from 1 to 5. Furthermore, meta-data about movies is available. For this exercise, we will use the 18 movie genres used to annotate movies. This information is available in movies.csv along with the title of each movie.

Let's have a quick look at what this data looks like. Here's some python script to do that :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


ratings = pd.read_csv('/path/ratings.dat', sep='::', header=None, names=['UserID', '
    MovieID', 'Rating', 'Timestamp'], engine='python')
movies = pd.read_csv('/path/movies.dat', sep='::', header=None, names=['MovieID', '
    Title', 'Genres'], engine='python', encoding='latin-1')


print(ratings.head())
print(movies.head())


plt.figure(figsize=(10, 6))
sns.countplot(x='Rating', data=ratings)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```

3 Getting started with recommendations

Make sure to install the recommandation library. Please follow the installation here

3.1 Load and split the dataset

The first step before starting implementing and testing about recommendation algorithms is to load the dataset and split it into a train set and a test set :

The cornac library provides some utilities to do that very easily. Have a look at the following :

- Reader
- Dataset
- StratifiedSplit

```
from cornac.data import Reader, Dataset
from cornac.eval_methods import StratifiedSplit

data = Reader().read("path/ratings.dat",
sep="::",skip_lines=0,fmt="UIRT")

dataset= Dataset.from_uir(data)
split= StratifiedSplit(data,test_size=0.2, seed=42, ,rating_threshold=4)
train_data=split.train_set
test_set=split.test_set
```

We are now ready to implement some basic recommendation algorithms

3.2 Item-based Collaborative Filtering

Item-based k-Nearest Neighbors (ItemKNN) is a collaborative filtering algorithm used for building recommendation systems. ItemKNN focuses on the similarities between items. The main idea is to recommend items that are similar to those the user has already liked or interacted with. Here's a step-by-step breakdown of how ItemKNN works :

1. **Item Similarity Computation** : Compute the similarity between each pair of items. The most used similarity measure is the cosine similarity.
2. **Neighborhood Formation** : For each item, identify the k most similar items, forming its neighborhood. The value of k is data-dependent
3. **Recommendation Generation** : To make a recommendation for a user, look at the items the user has already rated. For each of these items, identify its k-nearest neighbors and aggregate their ratings to generate a list of recommended items. Items with the highest aggregated scores are recommended to the user.

Let's see how this is done in `cornac`

- ItemKNN

```
from cornac.models import ItemKNN
itemcf=ItemKNN(k=10)
itemcf.fit(train_data)
```

3.3 User-based Collaborative Filtering

User-based k-Nearest Neighbors (UserKNN) is another collaborative filtering algorithm used in recommendation systems. The core idea behind UserKNN is to make recommendations based on the

preferences of similar users. UserKNN is intuitive and leverages the idea that users who have rated items similarly in the past will continue to have similar preferences. It's especially effective when users have a rich history of interactions. Here's how it works :

1. **User Similarity Computation** : Calculate the similarity between users based on their ratings of items. Common similarity measures include cosine similarity, Pearson correlation, and Jaccard index.
2. **Neighborhood Formation** : For each user, identify the k most similar users (neighbors). The value of k can be tuned for better performance and depends on the dataset.
3. Recommendation Generation : To recommend items to a user, aggregate the ratings from their neighbors. Items that the neighbors liked or rated highly are recommended to the user.

Similarly, this is done easily using `cornac` :

- UserKNN

```
from cornac.models import UserKNN
usercf=ItemKNN(k=10)
usercf.fit(train_data)
```

3.4 Matrix-Factorization using SVD

Singular Value Decomposition (SVD) is a powerful matrix factorization technique used in recommendation systems, particularly for collaborative filtering. The core idea is to decompose the user-item interaction matrix into latent factors representing users and items, which can then be used to predict missing entries (e.g., ratings). SVD is popular due to its effectiveness in capturing the underlying structure in the data, reducing dimensionality, and handling sparse matrices typical of user-item interactions. Here's a step-by-step breakdown of how SVD works in recommendation systems :

1. **User-Item Matrix** : Start with a matrix R, where rows represent users, columns represent items, and entries represent user-item interactions (e.g., ratings).
2. **Matrix Factorization** : Decompose this matrix R into three matrices :
   - U : A user-feature matrix where each row represents a user in terms of latent factors.
   - Σ : A diagonal matrix with singular values, representing the strength of each latent factor.
   - V^T : An item-feature matrix where each column represents an item in terms of latent factors.
3. **Loss Function** : The objective is to minimize the reconstruction error between the original matrix R and the product of the decomposed matrices $U\Sigma V^T$. This is typically done using the following loss function :

This is done in 3 lines of code using the library

- SVD

```
from cornac.models import SVD
svdmodel=SVD()
svdmodel.fit(train_data)
```

3.5 Let's see how to get recommendations

Cornac makes it very easy to generate top-k recommendations. For any trained models, there are several methods you can apply to compute prediction scores, get rankings, etc. You can have a look at some of the methods below :

- Rating prediction for a user-item pair
- Rank
- Recommend

You task is to write some code to provide users with recommendations, according to an already trained model. You should compare the recommendations of different models.

```
itemcf= ItemKNN(k=10, similarity='cosine')
itemcf.fit(train_data)

user_id = '42'

recommendation_list= FIXME to get top-5 recommendations

you should also use the file 'movies.dat' to print movie title and genres instead of
    seeing just movieIds
```

4 Diversifying Recommendations

Now let's see how we can incorporate more diversity in the recommendation we get. In the following subsections, details about some diversification algorithms are presented. Within your group, you should pick one algorithm and try to implement it. The idea is compare the recommendations that are generated from the original algorithm and the recommendations that are diversified.

4.1 Maximal Marginal Relevance (MMR)

A widely used approach to combine relevance and diversity is the greedy MMR [3, 5]. Given the set of already recommended items S that is initialized to the most relevant item, MMR selects at each step an item i^* that maximizes a linear combination of the utility of the selected item and the gain in diversity that is achieved according to already selected items. More formally, it chooses i^* such that ($\alpha \in [0, 1]$) :

$$i^* = \underset{i \in \mathcal{I} \setminus (P \cup S)}{\arg\max} \left((1 - \alpha) \cdot utility(i) + \alpha \min_{j \in S} dist(i, j) \right)$$

where α is a parameter that tunes the relative importance of each of the two factors. Small values of α means more importance on the utility of recommended items and high α values means that more importance is put on diversifying the resulting set.

4.2 Max-Sum Diversification (MSD)

MSD is based on the $p-$ facility dispersion problem, where the goal is to select a set of p facilities on a network so that the minimum distance between any pair of facilities is maximized [4]. Like in MMR, the objective comprises two terms. Here, the first term is a modular function $g(.)$ that calculates the utility of a given set S and the second term measures the diversity of S ($\alpha \in [0, 1]$) :

$$S^* = \underset{|S|=k, S \subset \mathcal{I} \setminus P}{\arg\max} \left((1 - \alpha) \cdot g(S) + \alpha \sum_{i \in S} \sum_{j \in S - \{i\}} dist(i, j) \right)$$

The problem is known to be NP-Hard, but greedy MSD gives an approximation guarantee [2]. In our setting, $g(S) = \sum_{i \in S} utility(i)$. Small values of α means more importance on the utility of recommended items and high α values means that more importance is put on diversifying the resulting set.

4.3 SWAP

SWAP [6] is a re-ranking based approach that starts with ranking all items according to their achieved utility. It then picks the k items with the highest utilities and iterates through items and swaps the item from the top-k list that contributes the least to diversity with the next highest utility item only if the swap brings more diversity in the recommendation list. SWAP uses a utility upper bound ub as a condition to stop the swapping which corresponds to how much drop in utility is tolerated.

Here, are the steps to build the swap algorithm

- Generate initial recommendations based on utility scores
- Define a diversity metric : for example intra-list-distance
- Calculate the initial diversity score of the top-k recommendation list
- iterate through other items and identify pairs of items in the recommendation list that could be swapped. Swappable pairs are usually those where swapping could increase the overall diversity score without significantly reducing the relevance.

- After each swap, evaluate the new recommendation list's relevance and diversity. Ensure that the relevance of the list does not drop below a certain threshold ub
- Iterate until the allowed drop on relevance is achieved.

4.4 Diversity-weighted Utility Maximization (DUM)

DUM [1] tackles the diversity problem by maximizing the utility of the selected items weighted by the gain they provide in diversity. The utility of items is the main objective but it is subjected to increasing the diversity of the recommendation list. The gain in diversity is measured by the number of different topics that are covered by the recommendation list. The objective function of DUM is to find an optimal ordering of items S^* as follows :

$$S^* = \arg\max_{S \in \Theta} \sum_{k=1}^{|\mathcal{I}|} [f(S_{k-1}) - f(S_k)] \cdot u(i_k)$$

where $S = [i_1, i_2, ..., i_{|\mathcal{I}|}]$ is an ordering of items in the whole set $\mathcal{I}$, Θ is the set of all item permutations, $S_k = \{i_1, ..., i_k\}$ is the set of the first k elements in S. The term $f(S_{k-1}) - f(S_k)$ measures the gain in diversity that is achieved when i_k is added to S_{k-1}, and $u(i)$ measures the utility of item i. To recommend k items DUM, the proposed greedy algorithm first ranks all items according to their utility and then iteratively selects the next item if an only if it increases the diversity.

Références

[1] Azin Ashkan, Branislav Kveton, Shlomo Berkovsky, and Zheng Wen. Optimal greedy diversity for recommendation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[2] Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166, 2012.

[3] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, 1998.

[4] Erhan Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1) :48–60, 1990.

[5] Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 421–432, 2012.

[6] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. It takes variety to make a world : diversification in recommender systems. In *Proceedings of the 12th international conference on extending database technology : Advances in database technology*, pages 368–378, 2009.