User-based Collaborative Filtering Recommender System

May 16, 2018

Abstract

User-based Collaborative Filtering(UserCF) algorithm recommends movies based on groups of users who share similar test on movies. A simple UserCF recommender system has been impletemented. The report is about the algorithm used in the simple recommender system and the future work to do.

1 User-based Collaborative Filtering Algorithm

UserCF algorithm has 2 steps.

- Find neighbors for each target users
- Recommend movies that the neighbors like but the target user has not known

1.1 User Similarity

The recommender system adopt an easy version definition of user similarity. It used Cosine Similarity to compute the user similarity.

Given a User u and a User v, let $N(u)$ represents the movie set that User u given positive feedback and $N(v)$ represents the movie set that User v given positive feedback.

Then

$$User\_sim_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u) \parallel N(v)|}} \tag{1}$$

The code is given here. Our train set is a sparse numpy array that train[u][m] is the rating that the User u gives to a Movie m and 0 means data missing in this train set.

```
def UserSimilarity(train):
    # build movie_users matrix from user_movies matrix
    mov_user = train.transpose()

    #caculate co-rated movies
    co = {}
    for i in range(np.shape(mov_user)[0]):
        #users who rate the same movie
        users = np.where(mov_user[i]>0)[0]
        for u in users:
            #print(type(u))
            if u not in co:
                co[u] = {}
            for v in users:
                if u==v:
                    continue;
                if(v not in co[u]):
                    co[u][v] = 0
                #the num of movies co-rated by u and v
                co[u][v] += 1

    # caculate similarity matrix user_sim
    user_sim = {}
    for u, co_u in co.items():
        if u not in user_sim:
            user_sim[u] = {}
        for v, times in co_u.items():
            u_len = np.count_nonzero(train[u,:])
            v_len = np.count_nonzero(train[v,:])
            user_sim[u][v] = times/math.sqrt( u_len * v_len )

    return user\_sim
```

To distinguish positive feedback from the ratings data rangeing from 1 to 5, each user's avrage rating should be computed as the base line. This job belongs to data preprocessing and we have not done it.

1.2 Top-N Recommendation

We take k neighbors with top-k user similarity and select the movies they like but the target user does not know. Then predict the ratings that the target user may give for those movies and rank them. Finally, return the top-N movies as our recommendation.

```
def Recommend(u,train,user_sim):
#   Find K similar users and recommend N movies for user
    k = 10
    n = 5

    rank = {}

    if np.count_nonzero(train[u]) == 0:
        print('The data of User ',u,' does not in this train set')
        return

    m_u = np.where(train[u]>0)[0]
    for v, sim in sorted(user_sim[u].items(), key=itemgetter(1),
         reverse=True)[0:k]:
#         print("top k: u",v)
        m_v = np.where(train[v,:]>0)[0]
        for m in m_v:
            if m in m_u:
                continue;
            if m not in rank:
                rank[m]=0
            rank[m] += sim*train[v,m]
#     print("rank",rank)
    return sorted(rank.items(), key=itemgetter(1), reverse=True)[0:n]
```

2 Future Work

2.1 Data Preprocessing

We are going to follow the Part 4 of the paper *Differentially Private Recommender Systems*[1]. The jobs include adding noise and data centralization.

2.2 Evaluation

Precise and *Recall* are used to evaluate the performance our recommender system.

References

[1]McSherry, F. and Mironov, I. Differentially private recommender systems. [online] Available at: https://dl.acm.org/citation.cfm?id=1557090 [Accessed 10 Apr. 2018].