Results from IMF and User Based Recommender systems on little users clusters

Claudio Cimarelli

University of Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg

Abstract. In this paper we will use a dataset of movies ratings,separating an ”expert user base” from a ”private user base” which will be clustered in groups. Then we’ll construct the Matrix Factorization model on the expert base in offline mode to serve with online updates method(Incremental MF) predictions for each cluster. We will also implement a collaborative filtering recommender system, the user based model, to predict ratings within each cluster.

1 Introduction

In this paper we will use the Movielens 1M ratings, a matrix of 6040 user x 3952 item(movies).
We will use 4000 user with more ratings as an ”expert user base” and the remaining 2040 as a ”private user base”. We will calculate with k-means algorithm 10 cluster on the ”private user base”, run IMF and User-based to give predictions and show the results for each one.

2 Clustering

To cluster it’s been used the library from Scikit-learn that provides numerous algorithm for machine learning, among which K-Means.
In this case we configured the class *Kmeans* selecting the number of clusters, number of trial to find the minimum and number of iterations per each trial.
Then, passing to the method fit a users matrix to be clustered, we obtain the groups of users that minimize the euclidean distance between each users vector in the clusters.

In this case, passing the user matrix as is, hopefully we will have in the same cluster users that rates items in similar way; this means high rates for same items and low for others. Instead using a mask of the user matrix, in which we have just 1 in place of a rating and 0 where no rating is given, we will end with cluster in which users are similar if just they have rated similar set of items, no matter what rate the give to them. We have different result depending on this choice and will be shown both.

After this we will add a little randomization to this clustering selecting a random pool of users, the 40% per each cluster, putting them together and finally redistributing an equal portion in each cluster.

```
import numpy as np
from sklearn.cluster import KMeans
def cluster_from(users_matrix, n_cluster=10):

    k_means = KMeans(n_clusters=n_cluster, n_init=10,
        max_iter=350).fit((users_matrix))
    clusters = []
    clusters_index = []

    for i in range(n_cluster):
        ci_index = np.where(k_means.labels_ == i)[0]
        clusters_index.append(ci_index)

    random_pick = np.empty([1, 0], dtype='int64')
    for index, cluster_index in enumerate(clusters_index):
        random_rows = np.sort(np.random.choice(len(cluster_index),
            int(len(cluster_index) * 40 / 100), replace=False))
        random_pick = np.append(random_pick, cluster_index[random_rows])
        clusters_index[index] = np.delete(cluster_index, random_rows)

    split = np.array_split(random_pick, n_cluster)
    for index in range(n_cluster):
        clusters_index[index] = np.sort(np.append(clusters_index[index],
            split[index]))
        cluster_i = users_matrix[clusters_index[index], :]
        clusters.append(cluster_i)
    return clusters, clusters_index
```

3 User Based Implementation

First Step: calculate cosine similarity with the following formula:

$$Sim_{x,y} = \langle \frac{\mathbf{R}_x}{||\mathbf{R}_x||}, \frac{\mathbf{R}_y}{||\mathbf{R}_y||} \rangle$$

That is simply implemented with this code:

```
    def cos_sim(users_matrix):
        norm = np.linalg.norm(users_matrix, axis=1).reshape((-1,1))
        with np.errstate(divide='ignore', invalid='ignore'):
            div = np.nan_to_num(np.divide(users_matrix, norm))
        sim = np.dot(div, div.T)
        return sim
```

Second Step: use the similarity between users as weight to calculate predictions:

$$p_{x,i} = \overline{r_x} + \frac{\displaystyle\sum_{y \in \mathcal{G} \,\wedge\, r_{y,i} \neq 0} Sim_{x,y}(r_{y,i} - \overline{r_y})}{\displaystyle\sum_{y \in \mathcal{G} \,\wedge\, r_{y,i} \neq 0} Sim_{x,y}}$$

In code:

```
    def user_based_pred(users_matrix):
        sim = cos_sim(users_matrix)
        nz_us = non_zero_matrix(users_matrix)
        with np.errstate(divide='ignore', invalid='ignore'):
            avg_ratings = np.nan_to_num(np.sum(users_matrix, axis=1)/
                np.sum(nz_us, axis=1)).reshape((-1, 1))
        deviations = (users_matrix - avg_ratings)*nz_us
        num = np.dot(sim, deviations)
        den = np.dot(sim, nz_us)
        with np.errstate(divide='ignore', invalid='ignore'):
            base_pred = np.nan_to_num(np.divide(num, den))
        pred = avg_ratings + base_pred
        return pred
```

4 Predictions and Evaluation

To calculate predictions with the two recommender system methods, User-Based and IMF, we took 80% of the ratings by each user to construct the models. Then with the remaining 20% we can calculate RMSE.
Moreover we show for each cluster its number of user and density of ratings.
Each of these statistic it's inserted in an array of length 10.
Following there is the code in which we consider one cluster at a time and for each one:

- take a train and test mask,
- calculate user based prediction and rmse,
- train the vector U for every user i the cluster with IMF,
- calculate $U \times V + bias$ and the rmse versus the rating matrix of the cluster,
- save length and density of ratings.

```
for index, cluster in enumerate(clusters):
  ###train and test set per each cluster
  train_mask, test_mask = build_test_train_masks(cluster)
  ###User based predictions
  pred = user_based_pred(cluster*train_mask)
  pred = np.maximum(np.minimum(pred, 5), 1)
  rmse_user_based[index] = calc_rmse(cluster * test_mask, pred)
  ###IMF predictions
  users_cluster = private_users_index[clusters_index[index]]
  for user, i in enumerate(users_cluster):
      profile_u = np.nonzero(batch_matrix[i, :] * train_mask[user, :])[0]
      u_batch[i, :] = user_update(u_batch[i, :], v_batch[profile_u, :], bias,
          batch_matrix[i, profile_u])
  f = np.dot(u_batch[users_cluster, :], v_batch.T) + bias
  f = np.maximum(np.minimum(f, 5), 1)
  rmse_imf[index] = calc_rmse(cluster*test_mask, f)
  len_clusters[index] = len(cluster)
  cluster_dens[index] = np.sum(non_zero_matrix(cluster))/ cluster.size
```

5 Results

Recalling that we could cluster with two different metric of the distance between user vectors, we could have in CASE 1 a metric based on how similar two user rates, CASE 2 how similar is the rated item set :

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9	Group 10	Measure
1.141	1.144	1.095	1.054	1.077	1.098	0.986	1.039	1.064	0.953	UB RMSE
1.039	0.998	0.969	0.914	0.893	0.949	0.898	0.984	0.948	0.875	IMF RMSE
166	420	158	150	98	240	158	233	171	246	Length

Table 1. Results CASE 1

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9	Group 10	Measure
0.983	1.071	1.045	1.096	1.063	1.001	1.089	1.103	1.159	0.961	UB RMSE
0.876	0.937	0.951	0.967	0.916	0.901	0.964	0.993	1.002	0.861	IMF RMSE
150	147	230	159	161	153	238	174	401	227	Length

Table 2. Results CASE 2

6 Combining IMF and User Based predictions

After computing separately predictions by IMF and User Based we try to combine these with linear combination, varying the parameter alpha that shift the importance of the rating given by the two methods above.
We use the following formula to calculate the combined prediction r_{ij}^{Comb}:

$$r_{ij}^{Comb} = alpha * r_{ij}^{IMF} + (1 - alpha) * r_{ij}^{UB}$$

Then we change alpha and see how the combination changes the RMSE. Hence starting from 0, which will result in pure User Based predictions, we end with 1, pure IMF predictions, incrementing it with step of size 0.1.

7 Hybrid Results

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9	Group 10
UserNum	150	147	230	159	161	153	238	174	401	227
Density %	1.0093	0.9159	0.9692	0.9566	0.9164	0.9723	0.8905	0.8334	0.7911	0.8982

Table 3. Information of Groups

Having 10 clusters as described above, we compute the RMSE with three algorithms on each cluster. One is IMF setting, where a user Alice's predictions are generated by the Incremental MF algorithm. The second is User-Based Collaborative Filtering method , where a user Alice's predictions are based on data from her group. The last one is the hybrid recommender, where Alice's predictions are generated by combining the results of the previous two algorithms, varying the weight with parameter alpha.

Group 1	Group 2	Group 3	Group 4	Group 5
0.8372	0.9358	0.9418	0.9413	0.9770
Group 6	Group 7	Group 8	Group 9	Group 10
0.9091	0.9733	0.9907	1.0224	0.8898

Table 4. RMSE in IMF Setting

Group 1	Group 2	Group 3	Group 4	Group 5
0.9539	1.0712	1.0651	1.0367	1.1249
Group 6	Group 7	Group 8	Group 9	Group 10
0.9891	1.1133	1.1376	1.1585	0.9857

Table 5. RMSE in User-based CF Setting

The results of the combination are in the following table which is decorated with the values of Density and Length of the cluster, plus the value of alpha that gives the best RMSE.

Group N.	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	best	Density	Length
1	1.0000	0.9743	0.9511	0.9309	0.9137	0.8997	0.8892	0.8821	0.8786	0.8788	0.8826	0.8	0.0101	150
2	1.0863	1.0534	1.0237	0.9977	0.9756	0.9578	0.9443	0.9355	0.9314	0.9322	0.9377	0.8	0.0092	147
3	1.0241	0.9999	0.9789	0.9611	0.9468	0.9361	0.9292	0.9261	0.9269	0.9315	0.9400	0.7	0.0097	230
4	1.0713	1.0429	1.0176	0.9955	0.9768	0.9618	0.9506	0.9434	0.9403	0.9412	0.9462	0.8	0.0096	159
5	1.1324	1.1021	1.0751	1.0516	1.0319	1.0163	1.0048	0.9977	0.9950	0.9968	1.0030	0.8	0.0092	161
6	1.0485	1.0179	0.9905	0.9666	0.9466	0.9307	0.9190	0.9118	0.9092	0.9112	0.9177	0.8	0.0097	153
7	1.1205	1.0877	1.0580	1.0316	1.0089	0.9901	0.9754	0.9651	0.9591	0.9577	0.9608	0.9	0.0089	238
8	1.1444	1.1082	1.0757	1.0471	1.0229	1.0033	0.9886	0.9790	0.9747	0.9758	0.9822	0.8	0.0083	174
9	1.1386	1.1070	1.0790	1.0549	1.0351	1.0197	1.0089	1.0029	1.0019	1.0057	1.0143	0.8	0.0079	401
10	0.9775	0.9534	0.9318	0.9128	0.8967	0.8836	0.8736	0.8669	0.8635	0.8634	0.8667	0.9	0.0090	227

Table 6. Results Combination

Following there is the graphical representation of this data.

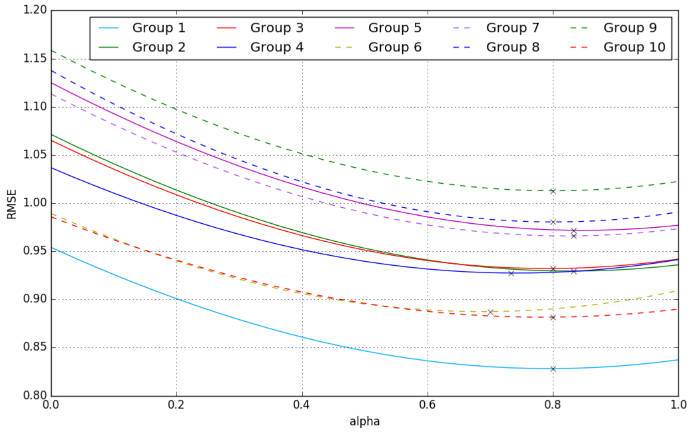

Fig. 1.

8 Code

To generate the previous tables and image i used the following functions:

```
def hybrid_rec_test(batch_matrix,experts_index, private_users_index):

    private_users_matrix = batch_matrix[private_users_index, :]
    clusters, clusters_index = build_clusters(private_users_matrix)

    mask = np.zeros_like(batch_matrix)
    mask[experts_index, :] = 1

    N = len(batch_matrix)
```

```
M = len(batch_matrix[0])
K = 10
u_batch, v_batch = train(batch_matrix * mask, N, M, K,
    suffix_name='experts')
nz_ratings = non_zero_matrix(batch_matrix * mask)
bias = np.sum(batch_matrix * mask) / np.sum(nz_ratings)

#  Plotting data arrays
rmse_user_based = np.zeros((10, 1))
rmse_imf = np.zeros((10, 1))
clust_len = np.zeros((10, 1))
clust_dens = np.zeros((10, 1))
rmse_hyb = np.zeros((10, 11)) # array with all values of rmse with alpha
    from 0 to 1(step length 0.1)
rmse_hyb_best = np.zeros((10, 2)) # array with best rmse and value of
    alpha to obtain it

for index, cluster in enumerate(clusters):
    ###train and test masks per each cluster
    train_mask, test_mask = build_test_train_masks(cluster)

    # ###User based predictions
    ub_pred = user_based_pred(cluster * train_mask)
    rmse_user_based[index] = calc_rmse(cluster * test_mask, ub_pred)

    ###IMF predictions
    imf_pred = imf_pred_cluster(cluster*train_mask, v_batch, bias)

    rmse_imf[index] = calc_rmse(cluster * test_mask, imf_pred)

    ### Combined predictions for alpha form 0 to 1 with steps of 0.1
    hyb_rmse_alpha = np.zeros(11)
    for i, alpha in enumerate(np.linspace(0, 1, 11)):
        hyb_pred = alpha * imf_pred + (1 - alpha) * ub_pred
        hyb_rmse_alpha[i] = calc_rmse(cluster * test_mask, hyb_pred)
    rmse_hyb[index] = hyb_rmse_alpha

    ### find best alpha and rmse
    rmse_hyb_best[index, 0] = np.linspace(0, 1,
        11)[np.argmin(hyb_rmse_alpha)]
    rmse_hyb_best[index, 1] = np.min(hyb_rmse_alpha)

    ### save cluster density and length
    clust_dens[index] = 100 * (np.sum(non_zero_matrix(cluster)) /
        cluster.size)
    clust_len[index] = len(cluster)

    ###prepare the plot for the hybrid rec varying alpha
    if index > 4:
        linestyle = "--"
    else:
        linestyle = "-"
    plt.plot(np.linspace(0, 1, 11), hyb_rmse_alpha, label='Group ' +
        str(index + 1), ls=linestyle)

plt.plot(rmse_hyb_best[:, 0], rmse_hyb_best[:, 1], 'kx', label="min RMSE")
```

```
    plt.xlabel(’alpha’)
    plt.ylabel(’RMSE’)
    plt.grid(True)
    plt.legend(ncol=5)
    plt.savefig(’plots/rmse_combined_cluster.png’, bbox_inches=’tight’)
    plt.show()

    ###save data for latex tables
    np.savetxt(’data/rmse_hyb’, rmse_hyb, ’%.4f’, delimiter=’ &’,
        newline=’\\\\ \hline \n’)
    np.savetxt(’data/rmse_ub’, rmse_user_based.T, ’%.4f’, delimiter=’ &’,
        newline=’\\\\ \hline \n’)
    np.savetxt(’data/rmse_imf’, rmse_imf.T, ’%.4f’, delimiter=’ &’,
        newline=’\\\\ \hline \n’)
    np.savetxt(’data/clus_len’, clust_len.T, ’%.d’, delimiter=’ &’,
        newline=’\\\\ \hline \n’)
    np.savetxt(’data/clus_dens’, clust_dens.T, ’%.4f’, delimiter=’ &’,
        newline=’\\\\ \hline \n’)
    pass


def hybrid_rec(imf_pred, ub_pred, alpha=0.8):
    ### Combined predictions
    comb_pred = alpha * imf_pred + (1 - alpha) * ub_pred

    return comb_pred


def imf_pred_cluster(cluster, v_batch, bias):
    u_cluster = np.zeros((len(cluster), len(v_batch[0])))
    ###IMF predictions
    for user in range(len(cluster)):
        profile_u = np.nonzero(cluster[user, :])[0]
        u_i = new_user_update(v_batch[profile_u, :], bias, cluster[user,
            profile_u])
        u_cluster[user] = u_i
    imf_pred = np.dot(u_cluster, v_batch.T) + bias
    imf_pred = np.maximum(np.minimum(imf_pred, 5), 1)

    return imf_pred
```

9 Experts size effects

```
    batch_matrix = load_data()
    experts_index, pvt_users_index = expert_base(batch_matrix)
    private_users_matrix = batch_matrix[pvt_users_index, :]
    clusters, clusters_index = build_clusters(private_users_matrix)
    train_mask, test_mask = build_test_train_masks(clusters[0])

    random_experts = np.random.permutation(experts_index)

    rmse_comb = np.zeros((3,8))
```

```
for i, usr_num in enumerate(np.linspace(500, 4000, 8)):
    experts_subset_index = np.sort(random_experts[-int(usr_num):])

    ###User based predictions
    #user_based_pool = np.append(batch_matrix[experts_subset_index, :],
         clusters[0] * train_mask, axis=0)
    ub_pred = user_based_pred(clusters[0] * train_mask)

    mask = np.zeros_like(batch_matrix)
    mask[experts_subset_index, :] = 1

    N = len(batch_matrix)
    M = len(batch_matrix[0])
    K = 10
    u_batch, v_batch = train(batch_matrix * mask, N, M, K,
         suffix_name=’experts_random’+ str(int(usr_num)))
    nz_ratings = non_zero_matrix(batch_matrix * mask)
    bias = np.sum(batch_matrix * mask) / np.sum(nz_ratings)
    imf_pred = imf_pred_cluster(clusters[0]*train_mask, v_batch, bias)

    result = hybrid_rec(imf_pred, ub_pred, alpha=0.8)

    rmse_comb[0,i] = calc_rmse(clusters[0] * test_mask, result)
    rmse_comb[1, i] = calc_rmse(clusters[0] * test_mask, imf_pred)
    rmse_comb[2, i] = calc_rmse(clusters[0] * test_mask, ub_pred)

plt.plot(np.linspace(500, 4000, 8), rmse_comb[0], label=’hybrid’)
plt.plot(np.linspace(500, 4000, 8), rmse_comb[1], label=’imf’)
plt.plot(np.linspace(500, 4000, 8), rmse_comb[2], label=’user based’)
plt.xlabel(’random experts length’)
plt.ylabel(’RMSE’)
plt.grid(True)
plt.legend(ncol=3)
plt.show()
pass
```

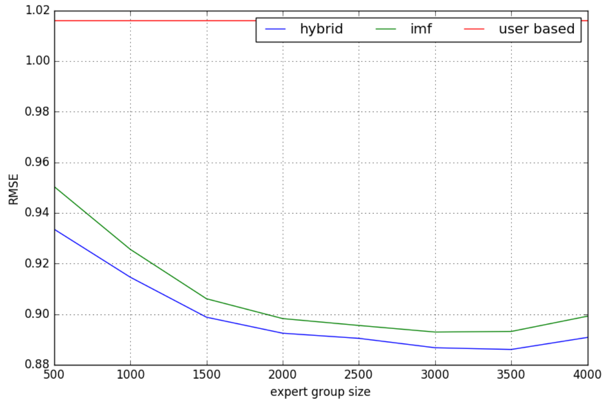

Fig. 2.