Recommendation System

Group6: Thomson Batidzirai, Sen Dai, Jie Jin, Jiyuan Liu, Qiuyu Ruan, Xiyi Yan

In this project, we use matrix factorization methods for recommender system. Our goal is to recommend the most appropriate movies to each individual user. Our matrix factorization has 3 parts: factorization algorithm, regularization and postprocessing. For factorization algorithm, we use Alternating Least Square(ALS). For regularization, we use temporal dynamic. And for postprocessing, we compare two different methods, which are KNN and kernel ridge regression respectively.

Step 1 Load Data and Train-test split

The dataset we used is 'rating.csv'. This dataset contains 4 columns, which are userId, movieId, rating and timestamp.First of all, we removed the movies with less than 5 ratings,and reduced the number of movies to 3268 from 9724. In order to consider temporal dynamic part, we needed to split timeline into bins. Considering the computational burdern, we splitted the time into 15 parts. Then we splitted the original data into train and test data. In this step, we have to ensure that each userId, movieId and bins appear at least once in train data. Otherwise, there wil exists zero in the latent factor while implementing ALS algorithm. The training set contains 71470 observations and the testing set contains 16894 observations.

We use R to choose subdata and splite test and train set. (The following are R code.):

movie_subset <- movie$movieId[movie$n > 5]

extract index of unique userid

index1 <- duplicated(rating_sort[,1])

data_test_1 <- rating_sort[!index1,]

rating_sort <- rating_sort[index1,]

extract index of unique movie

index2<-duplicated(rating_sort[,2])

data_test_2 <- rating_sort[!index2,]

rating_sort <- rating_sort[index2,]

extract index of unique timebin

index3<-duplicated(rating_sort[,6])

data_test_3 <- rating_sort[!index3,]

rating_sort <- rating_sort[index3,]

test_idx <- sample(1:nrow(rating_sort), round(nrow(rating_sort)/5, 0))

train_idx <- setdiff(1:nrow(rating_sort), test_idx)

data_train <- rating_sort[train_idx,]

data_test <- rating_sort[test_idx,]

data_train <- rbind(data_test_1,data_test_2,data_test_3,data_train)

In [0]:

```
from google.colab import files
uploaded = files.upload()
```

Choose File No file selected

No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data_test_subset.csv to data_test_subset (1).csv
```

In [0]:

```
from google.colab import files
uploaded = files.upload()
train = pd.read_csv('data_train_subset.csv')
test = pd.read_csv('data_test_subset.csv')
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data_train_subset.csv to data_train_subset (1).csv
```

Step 2 Matrix Factorization

Step 2.1 Algorithm(Alternating Least Squares) and Regularization(Temporal Dynamic)

Because the original user-item matrix is a sparse matrix. So we need to do matrix factorization. Our goal is to minimize the objective function, where q is the user matrix and p is the item matrix. For Regularization, We need to consider the temporal dynamic. we only need to consider the movie-related temporal effects. To make it efficient, We split the movie-bias into a stationary part(b_i) and a time changing part($b_{i,Bin(t)}$). As a result, We code the predictor as $\hat{r}_{ui}(t) = \mu + b_u + b_i + b_{i,Bin(t)} + q_i^T p_u$. For the Algorithm, we need to use Alternating Least Squares. Alternating Least Squares is a matrix factorization algorithm:

- Step1. Initialize matrix p, q
- Step2. fix q, p, b_u, b_i solve $b_{i,Bin(t)}$ by minimizing the objective function
- Step3. fix q, p, b_u, $b_{i,Bin(t)}$ solve b_i by minimizing the objective function
- Step4. fix p, q, $b_{i,Bin(t)}$,b_i solve b_u by minimizing the objective function
- Step5. fix $b_{i,Bin(t)}$,b_i, b_u, p solve q by minimizing the objective function
- Step6. fix $b_{i,Bin(t)}$,b_i, b_u, q solve p by minimizing the objective function
- Step7. Repeat until a stopping criterion is satisfied.

In [0]:

```
import pandas as pd
uploaded = files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving rmse_result.csv to rmse_result (2).csv
```

In [0]:

```
from google.colab import files
uploaded = files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data_test_subset.csv to data_test_subset.csv
```

In [0]:

```
drive = GoogleDrive(gauth)
your_module = drive.CreateFile({"id": "1yDFzLbF_otIow2aUH5C8zgI7abDIrGhv"})   #
```

```
"your_module_file_id" is the part after "id=" in the shareable link
your_module.GetContentFile("ALS_TD.py")          # Save the .py module file to Colab VM
import ALS_TD
```

Tune Parameter

In [0]:

```
## ALS_TD.ALSTD_CV (test,5,10,0.01,5)
## ALS_TD.ALSTD_CV (test,5,10,0.1,5)
## ALS_TD.ALSTD_CV (test,5,10,10,5)
```

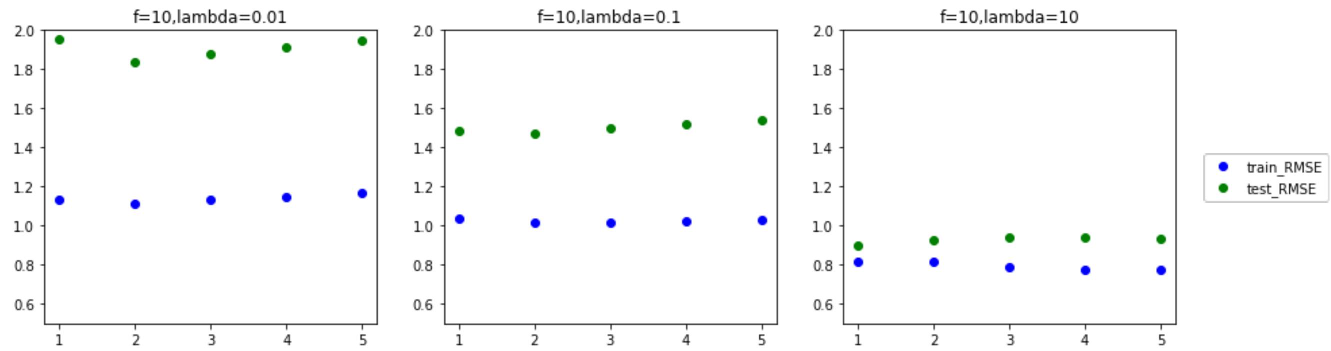
In [0]:

```
rmse_result=pd.read_csv('rmse_result.csv')
import matplotlib.pyplot as plt
x=(1,2,3,4,5)

fig, (ax1,ax2,ax3) = plt.subplots(1, 3, figsize=(15,4))

l1=ax1.plot(x,rmse_result.iloc[:,1],'bo')[0]
l2=ax1.plot(x,rmse_result.iloc[:,2],'go')[0]
ax1.set_ylim([0.5, 2])
ax1.title.set_text('f=10,lambda=0.01')

l3=ax2.plot(x,rmse_result.iloc[:,3],'bo')[0]
l4=ax2.plot(x,rmse_result.iloc[:,4],'go')[0]
ax2.set_ylim([0.5, 2])
ax2.title.set_text('f=10,lambda=0.1')

l5=ax3.plot(x,rmse_result.iloc[:,5],'bo')[0]
l6=ax3.plot(x,rmse_result.iloc[:,6],'go')[0]
ax3.set_ylim([0.5, 2])
ax3.title.set_text('f=10,lambda=10')

line_labels = ["train_RMSE", "test_RMSE"]
fig.legend([l1,l2,l3,l4,l5,l6],labels=line_labels,loc="center right",borderaxespad=0.1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: UserWarning: You have mixed
positional and keyword arguments, some input may be discarded.
```

Out[0]:

```
<matplotlib.legend.Legend at 0x7f4d5dd62b00>
```



After parameter tune, we choose 10 latent factor, 1 iteration and lamda = 10.

In [0]:

```
## mu, q, p, bi, bu, bit_2, train_RMSE_2, test_RMSE_2 = ALS_TD.ALSTDfit(10,10,1, train, test)
```

Step 3 Postprocessing

Postprocessing can help us improve the accuracy of the prediction.

In [0]:

```
import pandas as pd
uploaded = files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving q_2.csv to q_2.csv
```

In [0]:

```
uploaded = files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data_train_subset.csv to data_train_subset.csv
```

In [0]:

```
uploaded = files.upload()
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving ratings_subset.csv to ratings_subset.csv
```

In [0]:

```
mat_q=pd.read_csv('q_2.csv')
data_train_subset=pd.read_csv('data_train_subset.csv')
ratings_subset=pd.read_csv('ratings_subset.csv')
```

Step 3.1 KNN

We used KNN method to update all ratings. For example, we choose the movie j from matrix q and we calculate the cosine similarity to compare the latent factor of the movie j with the latent factor of other movies. Then we select the closest movie i and calculate the average ratings of movie i. Finally we update the rating of movie j with this mean. So the output matrix of KNN indicates that for each movie, all users give the same rating.

In [0]:

```
!pip install pydrive                             # Package to use Google Drive API - not installed
in Colab VM by default
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth                    # Other necessary packages
from oauth2client.client import GoogleCredentials
auth.authenticate_user()                         # Follow prompt in the authorization process
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
```

In [0]:

```
drive = GoogleDrive(gauth)
your_module = drive.CreateFile({"id": "1ijdOUAgWSrh2eLqCVKSQOJPW9okmQP8I"})    #
"your_module_file_id" is the part after "id=" in the shareable link
your_module.GetContentFile("knn_postprocessing.py")          # Save the .py module file to Colab VM
import knn_postprocessing
```

In [0]:

```
knn_postprocessing.knn_postprocessing(data_train_subset,mat_q)
```

Step 3.2 Kernel Ridge Regression

In [0]:

```
########################
#####Step1: Run KRR#####
########################

def krr_postprocessing(mat_q,data,data_train):

    import numpy as np
    import pandas as pd

    from sklearn import preprocessing
    from sklearn.kernel_ridge import KernelRidge

    n_movies=np.unique(data.movieId).shape[0]
    n_users=np.unique(data.userId).shape[0]

    updated_rating_mat=np.zeros((n_users,n_movies))

    mat_q=mat_q.T

    #normalize q matrix
    q_normalize=preprocessing.normalize(mat_q)
    q_normalize.shape
    q_normalize=pd.DataFrame(q_normalize.T)
    q_normalize.columns=[np.unique(data.movieId)]

    for i in range(n_users):

        rating_i=data_train.loc[data_train['userId']==i+1,['movieId','rating']]
        movieId_i=rating_i.iloc[:,0]
        y_i=rating_i.iloc[:,1]#rating vector of user i

        #create X for user i
        X_i=q_normalize.loc[:,movieId_i]

        #predictions of krr
        krr = KernelRidge(alpha=0.5,kernel="rbf")
        krr.fit(X_i.T,y_i)

        pred_krr=krr.predict(q_normalize.T)
        updated_rating_mat[i]=pred_krr

    return(updated_rating_mat)

####################################################
#####Step 2: Function for Calculating KRR_RMSE#####
####################################################
def rmse_krr(rating,est_rating):

    import numpy as np
    import math

    def sqr_err(obs):
        sqr_error=(obs[2]-est_rating.iloc[int(obs[0]-1),int(obs[4])])**2
        return(sqr_error)

    return(math.sqrt(np.mean(rating.apply(sqr_err,1))))

############################
#####Step 3: CV for KRR#####
############################

def cv_krr(data,data_train,k):

    import pandas as pd
    import numpy as np
    from sklearn.utils import shuffle
```

```
    #initialize train and test data and cv result
    n_movies=np.unique(data.movieId).shape[0]
    n_users=np.unique(data.userId).shape[0]
    n_col=data_train.shape[1]

    cv_result_mat=np.zeros((k,n_users,n_movies))
    n=data_train.shape[0]
    n_fold=int(n/k)

    data=np.zeros((k,n_fold,n_col))
    data_train=shuffle(data_train)

    krr_rmse_train=np.zeros(k)
    krr_rmse_test=np.zeros(k)


    for i in range(k):

        data[i] = data_train[i*n_fold:(i+1)*n_fold]
        vali = data[i]
        vali=pd.DataFrame(vali)
        train_new = data_train.drop(vali.index,axis=0)
        krr_result=krr_postprocessing(mat_q,ratings_subset,data_train_subset)
        cv_result_mat[i]=krr_result

        #calculate rmse for train and test
        krr_rmse_train[i]=rmse_krr(train_new,pd.DataFrame(krr_result))
        krr_rmse_test[i]=rmse_krr(vali,pd.DataFrame(krr_result))

    #get the predictions with smallest test error
    idx_min_rmse=krr_rmse_test.argmin()

    return(krr_rmse_test,cv_result_mat[idx_min_rmse])


test_error,best=cv_krr(ratings_subset,data_train_subset,5)
```

In [0]:

```
pd.DataFrame(best).head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	4.351820	4.213603	4.413049	4.349112	4.185339	4.507516	4.377710	4.159712	4.314976	4.347950	4.241032	4.244649	4.458720
1	3.984204	3.798626	3.811999	4.102197	3.833585	3.634537	3.607875	3.914802	4.111623	3.998452	3.542785	4.101274	3.896711
2	1.585993	1.514582	1.427432	1.225751	1.399430	1.439379	1.630687	1.497438	1.412664	1.519580	1.740790	1.278493	1.166803
3	3.883153	3.330078	3.610979	3.132044	3.696176	3.283170	3.696836	3.840751	3.585623	3.612926	3.373328	3.659608	3.042308
4	3.998687	3.299547	3.443292	3.388015	3.125498	3.985549	3.764177	3.035276	3.576003	4.009827	3.912953	3.072482	3.858053

5 rows × 3268 columns

In [0]:

```
test_error
```

Out[0]:

```
array([0.93742263, 0.94894305, 0.93595587, 0.94575991, 0.94906385])
```

With a 5-fold cross validation, our testing RMSEs are: 0.93742263, 0.94894305, 0.93595587, 0.94575991, 0.94906385, respectively.

Step 4 Combination using linear regression

After post-processing, we combine our algorithm, regularization and post-processing using linear regression. We treat these output of each step as input and calculate the coefficients. We train all algorithms on the training set. And then the predictions made by each algorithm for the test set are combined with linear regression on the test set. Add to the regression selected two-way interactions between predictors gives a small improvement.

In [0]:

```
drive = GoogleDrive(gauth)
your_module = drive.CreateFile({"id": "1QbnNeYyc0ajIsuOlUDJEAacKCX0VkRy7"})    # "your_module_file_id" is the part after "id=" in the shareable link
your_module.GetContentFile("linear_reg.py")          # Save the .py module file to Colab VM
import linear_reg

## KNN
## linear_reg.linear_regression_for_all(train,test,p,q,bi,bit,bu,pp,mu)
```

-0.03819894273294001

[6.56472085e+00 1.26445714e+00 1.00375197e+00 -3.11637031e-04]

In [0]:

```
## KRR
## linear_reg.linear_regression_for_all(train,test,p,q,bi,bit,bu,pp,mu)
```

In [0]:

```

```

Step 5 Results and Evaluation

We use RMSE as the evaluation for results.We obtained regression function for KNN and KRR, respectively. We got very similar results for them.

- Regression function for $KNN: y - 3.542 = -0.038 + 6.564pq + 1.265b_i + b_{i,bin(t)} + 1.003b_u - 0.00034KNN$
- Regression function fot $KRR: y - 3.542 = -0.04 + 6.565pq + 1.264b_i + b_{i,bin(t)} + 1.0038b_u - 0.000314KRR$

The traning and testing RMSE for regressing on KNN are 0.9000936 and 0.921986.The traning and testing RMSE for regressing on KRR are 0.9000965 and 0.921987.