MA 691: Term Paper 1

Arabin K. Dey

Sarthak Agarwal
140123031

Contents

Problem 1

Predict the missing rating of user to movie using Simulated Annealing on Movie lens small data set.

Python code :

```
import pandas as pd
import numpy as np
import math
#100004 ratings and 671 users across 9125 movies.

def cost(movie_feat_vec, user_feat_vec, l1, l2, matrix, is_rated, no_movies,
no_users, no_feat):
     new_cost=np.sum(np.sum(np.square(matrix-np.multiply(np.dot(movie_feat_vec,
    np.transpose(user_feat_vec)), is_rated))))+np.sum(np.sum(np.square(movie_feat_vec)))*l1
    + np.sum(np.sum(np.square(user_feat_vec)))*l2

     return new_cost/100000

print("Reading the ratings file...")
Rating=pd.read_csv('ratings.csv')
no_users = Rating['userId'].max()
no_movies = Rating['movieId'].max()
matrix = np.zeros(shape = (no_movies, no_users))
for i in range(0,len(Rating)):

     matrix[Rating['movieId'].iloc[i]-1, Rating['userId'].iloc[i]-1] = Rating['rating'].iloc[i]
print("Normalizing the rating matrix...")
##normalizing the rating matrix
is_rated = matrix!=0
mean_rating = np.zeros(shape=(matrix.shape[0], 1))
norm_rating = np.zeros(shape=(matrix.shape))
for i in range(0, no_movies):
     t=is_rated[i,]==1
     indexes = [j for j, x in enumerate(t) if x]
     if not indexes:
          continue
     mean_rating[i]=np.mean(matrix[i,indexes])
     norm_rating[i,indexes] = matrix[i,indexes] - mean_rating[i];
no_feat = 3
s = np.random.normal(0, 1, no_movies*no_feat)
movie_feat_vec=np.reshape(s, (no_movies,no_feat))
t = np.random.normal(0, 1, no_users*no_feat)
user_feat_vec=np.reshape(t, (no_users,no_feat))
l1=np.random.normal(0, 0.1, 1)
l2=np.random.normal(0, 0.1, 1)

##applying simulated anneling
print ("Applying Simulated Annealing....")
```

Problem 1 continued on next page. . .

```
initial_cost=cost(movie_feat_vec, user_feat_vec, l1, l2, norm_rating, is_rated
, no_movies, no_users, no_feat)
no_iter=10
steps=0.1
for k in range(0, no_iter):
     print("Iteration :",k)
     temp=pow(1-steps, k)
     s=np.random.normal(0,1,no_movies*no_feat)
     t_movie=np.reshape(s, (no_movies,no_feat))
     t=np.random.normal(0,1,no_users*no_feat)
     t_user=np.reshape(t, (no_users,no_feat))
     t_l1=np.random.normal(0,1,1)
     t_l2=np.random.normal(0,1,1)
     new_cost=cost(t_movie, t_user, t_l1, t_l2, norm_rating, is_rated, no_movies, no_users, no_feat)
     print ("New cost=",new_cost, " Old cost=", initial_cost)
     if new_cost < initial_cost or np.random.uniform(1,0,1) < np.exp(-(new_cost-initial_cost)/temp):
          initial_cost=new_cost
          movie_feat_vec=t_movie
          user_feat_vec=t_user
          l1=t_l1
          l2=t_l2

predictions = np.dot(movie_feat_vec, np.transpose(user_feat_vec))
for i in range(0, no_movies):
     predictions[i,:] = predictions[i,:] + mean_rating[i]
MSE = np.sum(np.square(np.multiply(matrix, is_rated) - np.multiply(predictions, is_rated)))/100000
print ("MSE=", MSE)
RMSE = math.sqrt(MSE)
print ("RMSE=", RMSE)
```

Explanation :

Data Set: The dataset used in the problem is the Movie lens dataset. It contains the ratings of 943 users on 1682 movies. The scale of rating is from $0-5$. The dataset contains some movies which are not rated.
The data is partitioned into Training and Test following a **70% - 30%** split.

Representing the Data: A matrix called 'ratingMat' of size $1682x943$ is constructed where the rows represent movies and columns the users. Every cell (i,j) of the matrix represents the rating assigned by the user j to the movie i. Some entries are 0 which means that the particular user has not rated that movie. A second matrix called 'didRating' is defined having the same dimension as previous matrix with binary values. 0 means there is no rating for the movie by the user and 1 means the user has rated that movie.

The number of features taken here is assumed to be 3. But it can be easily changed. Corresponding to number of features, the movieFeatures and userPreference matrices are randomly generated from standard normal distribution of size 1682 x 3 and 943 x 3 respectively. The regularization parameters λ_1 and λ_2 corresponding to movieFeatures and userPreference are randomly generated from uniform distribution with lower and upper bound to be 0 and 0.005 respectively.

Cost Function: The cost function is defined as

$$Cost \quad = \sum_{i=0}^{nMovies} \sum_{j=0}^{nUsers} [(UV^T)_{ij} - m_{ij}]^2 \quad + \quad \lambda_1 \sum_{i=0}^{nMovies} \sum_{j=0}^{nFeatures} U_{ij}^2 \quad + \quad \lambda_2 \sum_{i=0}^{nUsers} \sum_{j=0}^{nFeatures} V_{ij}^2$$

$$where \ \ m_{ij} \ \ is \ \ an \ \ element \ \ of \ \ ratingMat.$$
$$ij^{th} \ \ entry \ \ denote \ \ the \ \ rated \ \ entries \ \ only.$$

The code snippet which defines the cost can be found in function calculateCost in line 36.

Simulated Annealing: We have used Simulated Annealing as an optimization algorithm for finding the minimum of cost function. With the new parameters generated from target distribution, the cost calculated as newCost.
For the acceptance/rejection of new parameters, we have used the following techniques.

$$1) \ if \ newcost < oldCost \quad then \ accept \ the \ newly \ generated$$

$$2) \ if \ newCost > oldCost \quad then \ accept \ with \ min(1, \alpha)$$
$$\alpha = e^{-(newCost - oldCost)/Temp}$$

Temperature is also reduced per iteration

$$Temp_{k+1} = \beta Temp_k$$

Based on the optimal parameters generated by the simulated annealing algorithm, we predicted the missing values in the movie-user rating matrix.
To Test our model, we have computed RMSE (Root Mean Sqaure Error) with the test data we have formed earlier.

$$The \ RMSE \ found \ \ is \ \ \mathbf{1.906480}.$$