COMP4121 Project Report

Course recommendation system

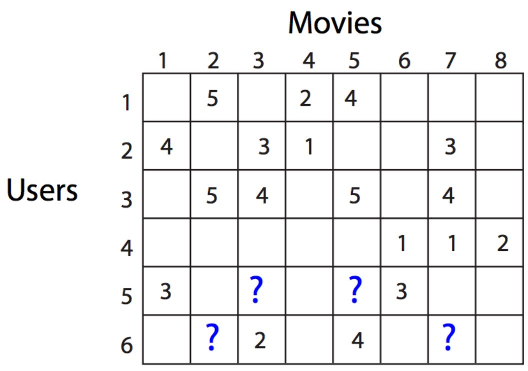
Abstract

This report of my major project would first introduce the background of the algorithm I used in my project. After introducing the background, the discussion would be mainly focus on how I applied the this magical algorithm to solve a real world problem in our live, to help students picking courses. Instead of dive into the mathematical definitions and proof, this report would mainly focus on how this substantial algorithm can be implemented step by step and only using very simple and efficient python code.

Introduction

Enlighten by the Netflix movie recommendation algorithm, maybe we can imitate this algorithm in order to help students to find their right course in university easier. This is my first idea about this project which is to help solve real problems as we have all met this before. Before start what work was carried out in this project, it would be better for us to introduce some background information about the Netflix movie recommendation first.

The input for the recommendation system would be table. We can think the table as a big sparse matrix which has each column represent a movie and each row represents a user.



Movies

Users

	1	2	3	4	5	6	7	8
1		5		2	4			
2	4		3	1			3	
3		5	4		5		4	
4						1	1	2
5	3		?		?	3		
6		?	2		4		?	

Most cells of this matrix would be empty and it would be our job to use the collaborative filtering method to predict the values in each empty cell. The output of the system would be table with all the missing values filled with predicted values. The way of we measuring this system would be using the Root Mean Square Error(RMSE) to measure for those (u, i) pairs where have both prediction and the actual rating. Assume there are C such pairs, then the RMSE would be:

$$RMSE = \sqrt{\sum_{(u,i)} \frac{(r_{ui} - \hat{r}_{ui})^2}{C}}$$

The smaller the RMSE, the better the system would be.

In order to make the system work, namely to predict user ratings, we would need to use the collaborative filtering methods. There are two main collaborative filtering methods which are the neighbourhood model and the latent factor model.

- The neighbourhood model is the intuitively simpler one which would try to find out users with similar tastes(neighbour). It would predict the missing values of a certain user by looking at the existing values in this users' neighbours. What's more, before we can start to use this method, we would need to find out the similarities scores between each pair of users first. Only after computing the matrix of pairwise similarities between users can we start to pick closest neighbours to predict ratings. Most importantly, this is the method that I used in my project.
- The second approach is called the latent factor model. This method would be a more abstract one comparing to the previous one. It would try to find out the most important underlying key factors that would influence users' ratings on the movies. Since I didn't apply this method in my major project, I am not going to spend much time in this.

Methodology

This part would start to dive into the major steps of my project. There are three phases in my project which are data preprocessing, model training and model evaluation.

Data Preprocessing:

The data used in this project is anonymised data from the UNSW Course Experience Survey of 2001-2006 CSE courses. It contains responses to 12 questions for ~30000 enrolments by ~6000 students in ~700 offerings of ~130 courses.

The columns are:

1) student number (mapped to consecutive integer)

2) session (e.g. 03s2)

3) course (mapped to consecutive integer)

4) student mark in course

5) student grade

6-19) responses to the 12 questions below

Response on 5-point scale where 1 = Strongly disagree 5 = Strongly Disagree or NA for not answered.

q01 It was always easy to know the standard of work expected in this course.

q02 I developed important skills in this course.

q03 The teaching staff of this course motivated me to do my best work.

q04 The workload in this course was too heavy.

q05 I usually had a clear idea of where I was going and what was expected of me in this co

q06 In this course I was tested on what I had memorised more than what I understood.

q07 I received helpful feedback on my work during this course.

q08 The teaching staff worked hard to make this course interesting.

q09 The topics covered in this course are important.

q01 I was generally given enough time to understand the things I had to learn this course.

q11 The assessment methods tested well what I'd learnt in this course.

q12 Overall, I was satisfied with the quality of this course.

Although there are quite a lot informative data in this file, we can only start with the data which most resemble the data format in the Netflix example. That means we would be mainly replying on the overall rating of a course instead of diving into the sub-scores of the survey.

Similar to the data format in the Netflix example, the input data should be m by n matrix where m is the number of users and n is the number of the courses. The rows stands for the students and columns stands for the courses. The values in entry (i, j) stands for the i-th student's rating of the j-th course. Above all, this matrix would carry the same meaning as the one in the Netflix example.

The programming language I used in this project is Python because it has some powerful external libraries like Pandas, Numpy, Scipy and Sklearn for scientific computing. In this data preprocessing part, I used the Pandas library to extract the data in "cse_anonymized.csv" and convert them into the matrix we mentioned above.

The major steps of data preprocessing:

1) Read file "cse_anonymized.csv" into a pandas table.
2) Construct an empty m by n matrix R_test.
3) Extract only the student number, course number and q12 score from the table. Fill the R_test with the by using the student number as the row number, course number as column number and q12 score as the value we need to put in.
4) Replace all the zeros in the R_test with np.nan(not a number).

Python code for data preprocessing:

```
import pandas as pd
import numpy as np

def convert_data_to_R_test(data):
    m = max(set(data["Student_number"]))
    n = max(set(data["Course_number"]))
    R_test = np.zeros((m+1, n+1))
    for k in data.index:
        i, j, score = data.loc[k][["Student_number", "Course_number", "q12"]]
        R_test[i, j] = score
    R_test[R_test==0] = np.nan
    return R_test

data = pd.read_csv("cse_anonymized.csv", sep='\t', header=None)
data.columns = [
    "Student_number", "Session", "Course_number", "Mark", "Grade",
    "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "q10", "q11", “q12"]

R_test = convert_data_to_R_test(data)
```

After the data preprocessing, we can find out that the shape of R_test is 6025 by 137 which means there are 6025 students and 137 courses. What’s more, this is also a really sparse matrix while almost 97% of the entries are empty. Good news is, the matrix in the Netflix is even much sparser than the one we have here.

Model Training:

In order to start to model training, we would need to first generate the R_train matrix which hide a certain percentage of entries in the R_test matrix.

Python code for generating R_train:

```
def get_R_train(percentage):
    m, n = R_test.shape
    R_train = R_test.copy()
    l = []
    for i in range(m):
        for j in range(n):
            if not np.isnan(R_test[i, j]):
                l.append((i, j))
    l = rd.sample(l, int(len(l) * (1 - percentage)))

    for (i, j) in l:
        R_train[i, j] = np.nan
    return R_train
R_train = get_R_train(0.8)
```

The argument provided to the get_R_train function indicates the fraction of data we would like to preserve. I chose to train the model by using 80% of the original data and evaluate the model by using the remaining 20%.

After acquiring the R_test matrix, we would need to generate the baseline model first. The baseline model can be acquired by removing the user bias and movie bias from the data. For any ratings in the matrix, it can be viewed as the sum of the average ratings of all movies, user bias and movie bias:

$$\hat{r}_{ui} = \overline{r} + b_u + b_i$$

The value of each user bias and movie bias can be calculated by minimizing resulting prediction's RMSE (for the training set's rating data), which is essentially:

$$\text{minimize}_{\{b_u, b_i\}} \sum_{(u,i)} (r_{ui} - \hat{r}_{ui})^2$$

This type of optimisation problem is called least squares which can be solved by using some methods in the Scipy library of Python. In order to work out all the values of user bias and movie bias, we would need to transform our data a little bit to make it able to read by the function in Scipy library.

A constant matrix **A** of shape C by (M + N) and C elements long constant vector **c** need to be construct(C is there number of all ratings, M is the number of users and N is the number of movies). The values in matrix **A** are zeros unless the corresponding user has rated the corresponding movie, in which it is 1. The values in vector **c** are the differences between the actual ratings and the average value of all ratings.

As least squares solutions often suffer from the overfitting problem, we would need to apply a technique called regularisation to avoid this problem. This is method would minimise a weighted sum of the sum of squared error and the sum of squared parameter values instead of the only minimising the squared error

$$\text{minimize}_{\{b_u, b_i\}} \sum_{(u,i)} (r_{ui} - \hat{r}_{ui})^2 + \lambda\left(\sum_u b_u^2 + \sum_i b_i^2\right)$$

where lambda is a tun-able parameter from zero to one.

Python code for generating calculating the user bias, movie bias and mean of all ratings:

```
def calculate_bu_bi_mean():
    mean = np.nanmean(R_train)
    m, n = R_train.shape
    c = []
    for j in range(n):
        for i in range(m):
            if not np.isnan(R_train[i][j]):
                c.append(R_train[i][j] - mean)
    c = np.array(c)

    A = np.zeros((len(c), m + n))
    count = 0
    for j in range(n):
        for i in range(m):
            if not np.isnan(R_train[i][j]):
                A[count][i] = 1
                A[count][m + j] = 1
                count += 1

    bu = sp.sparse.linalg.lsmr(A, c, damp=1)[0][0:m]
    bi = sp.sparse.linalg.lsmr(A, c, damp=1)[0][m:]

    return bu, bi, mean

bu, bi, mean = calculate_bu_bi_mean()
```

The next step is to preform the bias removal from the R_train matrix and generate the error matrix R_tilde.

Python code for bias removal:

```
def get_R_tilde():
    m, n = R_train.shape
    R_tilde = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            if not np.isnan(R_train[i][j]):
                R_tilde[i][j] = R_train[i][j] - mean - bu[i] - bi[j]
    R_tilde[R_tilde==0] = np.nan
    return R_tilde
```

Actually we have already finished training the model up to this point. The values of in error matrix represent the error between the real value and the predicted value of the baseline model.

And the next step would be calculating pairwise similarities between the students. The similarity measuring metric I used in the project is the same one in the Netflix example which is also the well know one, cosine similarity. For each pair of students, the similarity of them can be calculated by using this formula:

$$Sim(i,j) = \frac{\sum \tilde{r}_{im}\tilde{r}_{jm}}{\sqrt{\sum \tilde{r}_{im}^2}\sqrt{\sum \tilde{r}_{jm}^2}}$$

m stands for number of the courses that student i and student j have both rated

Since there are 6025 students, a student pairwise similarity matrix of shape 6025 * 6025 would be constructed.

Python code for calculating student similarity matrix:

```
from sklearn.metrics.pairwise import cosine_similarity
import scipy as sp

def calculate_student_similarities():
    R_tilda_sparse = sp.sparse.csr_matrix(np.nan_to_num(R_tilda))
    return cosine_similarity(R_tilda_sparse, dense_output=False)

student_similarities = calculate_student_similarities()
```

This a huge sparse matrix, so using the traditional array like matrix data structure would be very inefficient (takes up to about 10mins to get the result). I took advantage of the Python Scipy library which contains a data structure especially design for the huge sparse matrix. The running time of constructing the similarity matrix has dropped down to less than 1 second. Also, rather than writing my own function to calculate the cosine similarity, I used the function in Python Sklearn library.

Lastly, the final step is to utilise both the student similarity matrix and the error matrix to generate the matrix R_hat which has all the entries as the predicted value of the neighbourhood method. Before we run the neighbourhood algorithm, we have to decide how to pick the best neighbours for a student. The criteria I chose to pick the closest neighbours is very straight forward which is just pick the candidates which have the top 10 largest absolute value of similarity.

After picking the top 10 neighbours, the next step which is also the most important step of the neighbourhood method, is to predict ratings by using the data from the neighbours and the baseline predictor. The predicted value of the i-th student's rating of j-th course is:

$$\hat{r}_{ui}^{N} = (\overline{r} + b_u + b_i) + \frac{\sum_{j \in L_i} Sim(i,j)\tilde{r}_{uj}}{\sum_{j \in L_i} \left|Sim(i,j)\right|}$$

L_i is the set of neighbours of student i.

Python code for generating the R_hat matrix:

```
def get_R_hat():
  m, n = R_tilde.shape
   R_hat = np.zeros((m, n))
   for i in range(m):
      neighbors = student_similarities[i].nonzero()[1]
      neighbors = sorted(neighbors, key=lambda k: student_similarities[i, k])[:10]
      for j in range(n):
         R_hat[i][j] += mean + bu[i] + bi[j]
         a = 0
         b = 0
         for k in neighbors:
            if not np.isnan(R_tilde[k][j]):
               a += student_similarities[i, k] * R_tilde[k][j]
               b += abs(k)
         if a != 0 and b != 0:
            R_hat[i][j] += a / b
            if R_hat[i][j] <= 1:
               R_hat[i][j] = 1
            if R_hat[i][j] >= 5:
               R_hat[i][j] = 5
   return R_hat
```

For those predicted values which are lower than 1(higher than 5), we just simply replace them with 1(5).

Model Evaluation:

Firstly, calculate the training error of the model by calculating the RMSE between the R_train matrix and the R_hat matrix.

Python code for calculating training error:

```
def train_error():
    m, n = R_hat.shape
    error = []
    for i in range(m):
        for j in range(n):
            if not np.isnan(R_train[i][j]):
                error.append((R_train[i][j] - R_hat[i][j]) ** 2)
    return np.sqrt(np.mean(error))
```

After running this function, the result we got is 0.89825

Secondly, calculate the testing error of the model by calculating the RMSE between the R_test matrix and the R_hat matrix but only consider the 20% of testing data which were hidden from the very beginning.

Python code for calculating testing error:

```
def test_error():
    m, n = R_hat.shape
    error = []
    for i in range(m):
        for j in range(n):
            if not np.isnan(R_test[i][j]) and np.isnan(R_train[i][j]):
                error.append((R_test[i][j] - R_hat[i][j]) ** 2)
    return np.sqrt(np.mean(error))
```

After running this function, the result we got is 1.07122

Further discussion

In my opinion, this course recommendation system can be further improved if we can squeeze out more information from the data, since we have only used the value of q12. Followings are some of my thoughts regarding to how the performance of this course recommendation system might be improved.

1) Try to utilise other information in the dataset (marks, grades, ratings of other questions)
2) Try the latent factor method
3) Use the course similarities instead of student similarities
4) Try other lambda values when doing regularisation

References

Mung Chiang Networked Life, 1st edition, Cambridge University Press, 2012, pp.60-86