UNSW
THE UNIVERSITY OF NEW SOUTH WALES

School of Computer Science and Engineering

COMP9417 Group Project Report

Recommender system using collaborative filtering

Group Member

Yiwei Xue	Z5152673
Tianxiao Chen	Z5148945
Xun Zhang	Z5136876
Xing Wang	Z5150224

Abstract

With the rapid development of information technology, people have gradually entered the era of information overload from the period of lack of information. Nowadays, both information consumers and producers encounter significant challenges. For the aspect of consumers, it is hard for them to find the things they are interested in from a large amount of data stream. Plus, for the information producers, they are confused with how to make the information output stand out and catch users' eyes. The recommendation system provides an excellent choice to deal with this situation. It aims to coordinate the relationships between users and information as it helps users to find information that is useful to them. Besides, it allows the information to be exposed in front of users who are interested in it. Thereby it achieves a win-win situation for the two sides. Many shopping websites, like Taobao and Amazon, have their own recommendation system algorithms to improve the users' shopping experience. The main research objective of this paper is to develop a movie-based recommendation system, which is one of the critical applications of the recommendation system.

1. Introduction

1.1 Recommended Algorithm Classification

Generally, the recommendation system can be divided into two broad categories:

(1) Content-based recommendation

By mining the TF-IDF feature vector of the text to obtain users' preference, products that are similar to the consumers' previous favorite elements (such as meta-data, description, topic, etc.) are recommended to the users. Such a recommendation algorithm can help to find out the users' unique preferences successful.

(2) Coordinating filtering recommendation

A better recommendation effect can be obtained based on users' past behavior and statistical machine learning algorithm. In this paper, we will study and use the method of the coordinating filtering recommendation. Although the content-based recommendation is intuitive and easy to interpret, it relies only on past items and user content and lacks in diversity. Coordinating filtering recommendation system does not require any domain knowledge to discover new or different interests. Besides, it has high personalization and automation, so they are more in line with our recommendation system, which we will explain later.

1.2 Collaborative Filtering

Collaborative filtering is a typical method of harnessing collective intelligence. For example, if you want to watch a movie now but have no idea which one is excellent. You will usually ask your friends about what they have viewed recently. In particular, you prefer to get recommendations from friends who have similar tastes. This example is the core idea of collaborative filtering. Collaborative filtering discovers typically a small group of people whose taste is similar to you among all users. The filtering makes these users become neighbors, and then organize them into a sorted directory according to the things they like as recommendations.

Collaborative filtering can mainly be divided into two categories:

(1) User-based collaborative filtering

This kind of collaborative filtering discovers the users' preferences of the products or contents in terms of the users' past behavior data, and then measures and scores these likes. The relationship between users is calculated according to the attitudes and preferences of different users for the same product or content, and then the products are recommended to users who have similar preferences.

In step 1, cosine similarity and Pearson correlation coefficient are used to look for similar preferences among users.

Cosine similarity:

$$cos(\theta) = \frac{\sum_{k=1}^{n} x_{1k} x_{2k}}{\sqrt{\sum_{k=1}^{n} {x_{1k}}^2} \sqrt{\sum_{k=1}^{n} {x_{2k}}^2}}$$

The value of cosine is between [-1, 1]. The closer the value of cosine gets to 1, the more relevant the two users are.

Pearson correlation coefficient:

$$sim(x_1, x_2) = \frac{\sum_{k=1}^{n} (x_{1k} - \bar{x_1})(x_{2k} - \bar{x_2})}{\sqrt{\sum_{k=1}^{n} (x_{1k} - \bar{x_1})^2} \sqrt{\sum_{k=1}^{n} (x_{2k} - \bar{x_2})^2}}$$

```
def Pearson_Correlation_Similarity(self,vec1,vec2):
    """
        calculate the pearson similarity of two vectors
    """
    v1,v2 = self.removeZeros(vec1,vec2)
    if v1 == v2:
        return 1
    else:
        mean1,mean2 = sum(v1)/len(v1), sum(v2)/len(v2)
        sim = 0
        for i in range(len(v1)):
            sim += (v1[i]-mean1)*(v2[i]-mean2)
        t1 = math.sqrt(sum([(x-mean1)**2 for x in v1]))
        t2 = math.sqrt(sum([(x-mean2)**2 for x in v2]))
        return 0 if t1*t2 == 0 else  sim /(t1*t2)
```

The output range of the Pearson correlation coefficient is consistent with cosine similarity and also between -1 and 1. 0 means no correlation, a negative value means negative correlation, and a positive value means positive correlation.

	Item1	Item2	Item3	Item4
User1	3		4	Recommend
User2	1	3		2
User3	2	3	3	4

Table1: User Based

In step 2, items which are not duplicates will be recommended to the users who showed interest in them. Therefore, we need to recommend products that users have not viewed or purchased.

For example, if we would like to recommend item to user1, we need firstly find the users similar to him, then weight the similarity by their ratings of different products. Finally, we sort and recommend them to user1.

The method above is a user-based collaborative filtering algorithm. This algorithm relies on the user's historical behavior data to calculate relevance. And for some new websites or sites with less data, we can use the item-based collaborative filtering algorithm.

(2) Item-based collaborative filtering

The item-based collaborative filtering algorithm is very similar to the user-based collaborative filtering algorithm, which interchanges goods and users. The relationship between the items is obtained by calculating the scores of different items rated by different users. This is a recommendation for similar items to the user based on the relationship between the items.

During step 1, similar items are asked to be found. As above, we can also use the cosine similarity and the Pearson correlation coefficient to calculate the similarity.

	User1	User2	User3
Item1	3	4	3
Item2		3	
Item3	3	5	recommend

Table2: Item Based

In Step 2, after obtaining similar items, it will predict the things that the current user has not indicated the preference according to the preference of the user's history, and then calculating a sorted item list as a recommendation.

As shown in the figure above, for item 1, according to the historical preference of all users, all users who like item 1 like item 3. it can be concluded that item 1 is similar to item 3. While user 3 likes item 1, so it can be inferred that user 3 may also like item 3.

2. Analysis of Algorithms

2.1 KNN Introduction

KNN is the basic algorithm for recommending systems. That is, given a training data set (i.e., the data set corresponding to the user movie rating above us). For the new input data, it could find the K instances closest to the instance in the training data set.

In the recommendation system, KNN is mainly divided into two categories(user-based and item-based), which have been explained in detail above.

The main steps of the algorithm are divided into three stages:

In the first step, we need to calculate the similarity. In the whole module, the cosine similarity and the Pearson correlation coefficient have to be chosen.

In the next step, through the similarity calculated, finding K-neighbours which are the most similar(or close) to the target, then use these K-neighbours to predict the score.

$$\hat{r}_{u,m} = \frac{\sum_{n \in N(u,m)} S_{mn} R_{u,n}}{\sum_{n \in N(u,m)} S_{mn}}$$

Finally, we calculate the relative best RMSE by continually adjusting the number of K. In order to speed up the search for k-nearest neighbors and speed up the operation efficiency, we used two different strategies, Kmeans and Tree structure to compare them.

2.2 Kmeans

K-means is a clustering algorithm. The sample data set is divided into K clusters by constant convergence. Meanwhile, the cluster points need to be as close as possible, and the distance between the clusters becomes larger.

Algorithm flow:

Let Sample set D ={x1,x2,x3,x4…...Xm}, we could first take a K value and determine the number of iterations N and Cluster C = {c1,c2,c3,c4….Ck}.

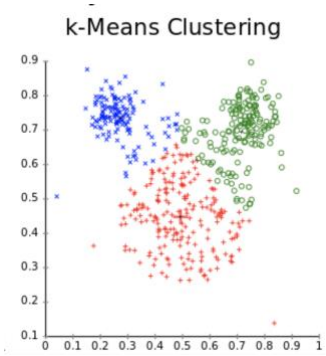
Step1: Randomly select k as the initial center from the data set： {μ1,μ2...μk}

Setp2: For n=1, 2, …N, we first initialize Ci = []. Then we calculate the distance from each sample Xi to each centroid: $d_{ij} = \|x_i - \mu_j\|^2$, mark it to the nearest cluster class Ci so that all clusters could be updated in this way.

Then recalculate the new centroid for all samples in Cj: $\mu_i \frac{1}{|C_i|}\sum_{x\in c_i} x$.

Our ultimate goal is to minimize the square error: E = $\sum_{i=1}^{k}\sum_{x\in c_i}\|x-\mu_i\|^2$.

Here we converge through constant iterations, and the primary goal of k-means is to minimize E. Usually we can't guarantee that we will get the global minimum, but in general, the local optimum reached by k-means has already met the requirements.




```
def KMeans_runner(self):
    for i in range(self.iteration):
        self.distribution = self.Distribution()
        self.cluster_center = self.update_new_centers()
        print(self.cal_var())
```

Figure1: K-Means

2.3 Kd-Tree

Kd-Tree(K-dimensional tree) is a high-dimensional index tree data structure commonly used for Nearest Neighbour and Approximate Nearest Neighbour in large-scale high-dimensional data spaces. Therefore, we use Kd-Tree to implement Knn.

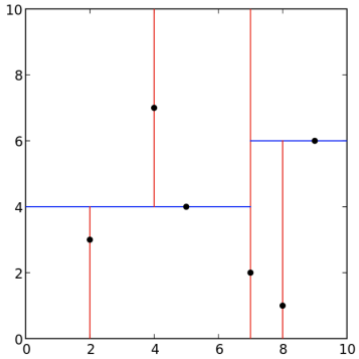


Figure2: Kd-Tree

During the construction, we first calculate the Pearson Correlation Similarity of each dimension data. Then we select the largest direction of the Pearson Correlation Similarity as our dimension (this is to ensure that our data can be separated as soon as possible). After selecting the divided dimensions, we sort the data of this dimension from small to large. Select the median as our root node. Then we loop this operation so that we can recursively divide the samples into groups. Meanwhile, the similarities within the same group are quite close. This method reduces the search time in the recommendation system prediction process (find K nearest neighbors).

2.4 Ball-Tree

In order to solve the problem of kd Tree inefficiency in high dimensions, we use k-d tree's improved method, ball tree. It uses nested hypersphere to split the data in place of the previous super-rectangular partition.

Here show the steps to build the Ball Tree:

- Step1: Construct a minimum hypersphere containing all the samples and select a point from the ball which is farthest from the center point of the ball.
- Step2: Select the second point farthest from the first point and assign all the points in the ball to the nearest one to the two cluster centers.
- Step3: Calculate the center of each cluster, and the cluster can contain the minimum radius required for all its data points.
- Step4: Recursively performing the previous step so that we can evenly divide the sample into several groups.

Here are the implementation steps of the ball-tree to retrieve the nearest neighbor (Take q as the search point to find the nearest neighbor within radius r).

- Step 1: From the root node Q recursively traverses from top to bottom each subspace Pi that may contain the final nearest neighbor
- Step 2: If the sum of the radius of the subspace Pi and R is less than the distance from the center point (Pi) to the target point (q)(i.e. $(radius(Pi) + r) \leq \|center(Pi) - q\|$).The recursive search within the subspace sample point that satisfies such a condition meets the point of $\|q - x\| \leq r$ is the nearest neighbor point we want.
- Step 3: Since the sub-super spheres a and b are truncated by q, and for the subspaces in a and b, d, h, and f are intercepted by q, so the linear search is performed in d, h, and f. Subspaces such as c, e, and g that are too far away will be discarded. Finally, $[X_4, X_7]$ is the final closest neighbor being obtained.

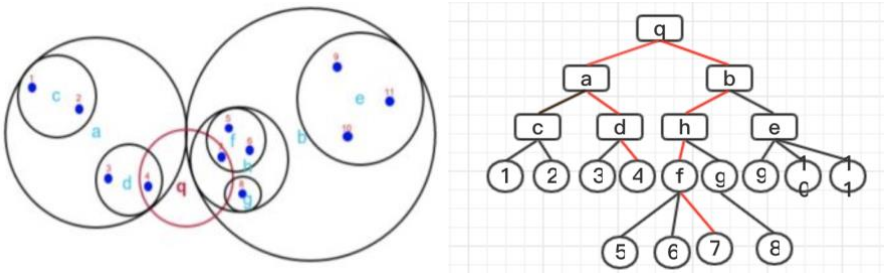


Figure3: Ball-Tree

3. RMSE Definition

Unlike most classic machine learning problems (such as classification/clustering/regression), Although the input data is simple and straightforward, Recsys is more like an integrated and complex system. Therefore, we cannot use some indicators such as Confusion Matrix or F1-score to evaluate our model. So here we apply the root mean square error (RMSE) as a performance test.

RMSE is the square root of the deviation of the observed value from the true value and the square root of the number of observations. In the actual measurement, the number of observations is always finite, and the true value can only be obtained with the most reliable (best) value. Instead, the square root error is sensitive to the extra large or small error in a set of measurements. Thus， if there is a large difference between a real value and a predicted value, the square operation will amplify the magnitude of the data. Therefore, compared with other methods such as mean square error (MSE) or mean absolute error (MAE), RMSE is more practical.

$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_{test}^{(i)} - \hat{y}_{test}^{(i)})} = \sqrt{MSEtest} \text{ =RMSE}$$

4. Results

	Model	Running time/s	RMSE
1	Basic KNN user by user	696s	1.173
2	Basic KNN item by item	1289s	1.180
3	Basic KNN user by user with bias	701s	1.168
4	Basic KNN item by item with bias	1289s	1.172
5	Kmeans user by user	208s	1.173
6	Kmeans item by item	590s	1.170
7	Kmeans user by user with bias	229s	1.166
8	Kmeans item by item with bias	540s	1.167
9	K-D tree user by user	38s	1.171
10	K-D tree item by item	47s	1.170
11	K-D tree user by user with bias	40s	1.168
12	K-D tree item by item with bias	72s	1.170
13	Ball Tree user by user	963s	1.172
14	Ball Tree item by item	1587s	1.174
15	Ball Tree user by user with bias	1029s	1.166
16	Ball Tree item by item with bias	1541s	1.245

Table3: Results

In order to facilitate the comparison between methods, we conducted model cross-validation for ml-100k data. And we set k at 10

And we considered the bias case here, with bias means taking consideration of movie global mean, user and movie bias.

```
def get_user_bias(self, user_index, movie_index):

    average_user_score = self.user_mean[user_index] if self.type == 1 else self.user_mean[movie_index]
    average_movie_score = self.movie_mean[movie_index] if self.type == 1 else self.movie_mean[user_index]
    return average_user_score + average_movie_score - self.global_average
```

From the table above we extract the best parts of each method for an intuitive comparison.

Method	Basic KNN user by user with bias	Kmeans user by user with bias	K-D tree user by user with bias	Ball Tree user by user with bias
RMSE	1.168	1.166	1.168	1.166

Table4: Best results for four methods

We can see that user by user with bias has the lowest error in each method compared with the other three. And from the table, we can see that Kmeans and K-D tree are relatively performed well in predicting effect(K-D tree is the fastest and not too far behind). Although Ball-tree also has a lower RMSE than the K-D tree, there is a significant shortage of time utilization. Maybe there is an improved method on Ball-tree, but due to the limited time, we will focus on Kmeans and K-D tree.

We take different k nearest neighbors as test parameter in model with Kmeans user by the user with bias and K-D tree user by the user with bias.

Num_neighbors	Kmeans		KD-tree	
	Time	RMSE	Time	RMSE
7	22	1.167	44s	1.170
10	229s	1.1669	39s	1.168
15	236s	1.1667	82s	1.168
20	241s	1.1664	48s	1.179
30	300s	1.1655	51s	1.168
50	469s	1.1668	54s	1.1687
80	750s	1.1673	53s	1.1699

Table5: Results by different K

Next, we represent the data in the line graph to visualize the data:

As shown in the figure below, the RMSE value decreases slightly with the increase of k value. However, when K value reaches a certain level, the RMSE value also stops falling and starts to rise. Besides, we can see that the RMSE values of Kmeans and K-D tree are not much different, but from the perspective of running efficiency, KDtree is obviously much faster. So KDtree is an excellent method by comparison.

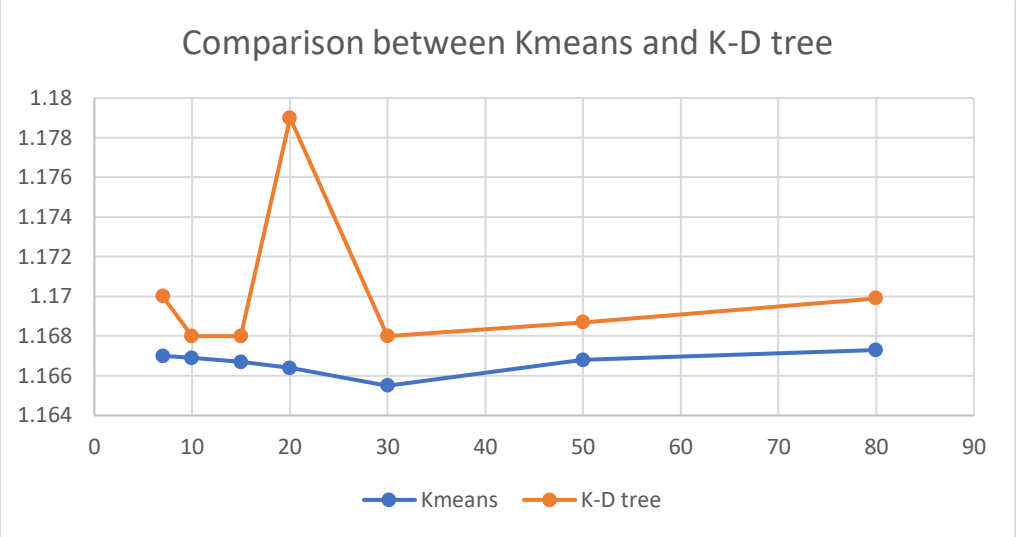


Figure4: Comparison between Kmeans and Kd-Tree

5. Discussion

This article discusses a variety of Recsys methods from the perspective of KNN. It applies many optimization algorithms to improve overall efficiency. For the basic KNN algorithm, it only needs to calculate the distance between all points and the target (similarity). Since KNN will go through all the points, thus the accuracy could be the highest. However, KNN is only suitable for a small data set. When talking to more massive data sets, the speed and efficiency of KNN will be greatly reduced.

Therefore, the optimization algorithm for KNN is discussed in this paper, namely Kmeans, Kd-Tree and Ball Tree. First, Kmeans is a clustering algorithm for unsupervised learning, and it can be used to divide the sample set into K clusters in advance. Therefore, Kmeans can significantly increase the efficiency of KNN. Kmeans indeed has a faster convergence speed and a better clustering effect. However, the disadvantage is that the selection of k value is not easy to choose. Plus, due to it uses an iterative method, usually only the local optimum can be found. Therefore, Kmeans is not the best optimization of KNN. Then we take the Tree optimization method, which is Kd Tree and Ball Tree, respectively. The KD Tree utilizes the structure of the binary tree to improve the efficiency of the KNN search. Though in some cases, the efficiency is not high when the data is unevenly distributed, the efficiency of the k-d tree will decrease rapidly. Thus, we find another method called ball tree. This method is an improvement of the Kd Tree, which is based on the plane of the hypersphere. However, our conclusions show that for the low latitude data sets, the Kd Tree still performs better than the ball tree.

For other possible model optimization, we can also improve the recommendation efficiency by using some other popular Recsys method like Latent Factor Models and combining the methods to minimize RMSE in a shorter time.

6. Conclusion

In this project, we used the collaborative filtering with KNN and its optimization algorithm to predict the film score. The datasets here used100k and 1M datasets.

First of all, we can see that the user-based scoring algorithm is significantly better than the item-based one. Moreover, RMSE can be effectively reduced by introducing global mean plus user bias and movie bias.

We observe the prediction through three optimization methods here. We can see that Kmeans and Ball-tree have good performance in terms of prediction, but the time utilization rate of Ball-tree is too low. In the same time, KD-tree could make a prediction of larger data and the difference of predicted scores is not very big. Therefore, in terms of efficiency, KNN with KD-tree is a relatively good algorithm, which can guarantee relatively good prediction in the case of fast speed.

In real life, the time cost and efficiency of the recommendation system are the main reasons for users to choose it. Therefore, the optimization algorithm of this paper is indispensable, the structure of the tree in the report greatly reduces the classification time and improves the user experience.

7. Reference

1. Herlocker, J.L., Konstan, J.A., Terveen, L.G. and Riedl, J.T., 2004. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22(1), pp.5-53.

2. Schafer, J.B., Frankowski, D., Herlocker, J. and Sen, S., 2007. Collaborative filtering recommender systems. In The adaptive web (pp. 291-324). Springer, Berlin, Heidelberg.

3. Zhang, M.L. and Zhou, Z.H., 2007. ML-KNN: A lazy learning approach to multi-label learning. Pattern recognition, 40(7), pp.2038-2048.

4. Keller, J.M., Gray, M.R. and Givens, J.A., 1985. A fuzzy k-nearest neighbor algorithm. IEEE transactions on systems, man, and cybernetics, (4), pp.580-585.

5. Hartigan, J.A. and Wong, M.A., 1979. Algorithm AS 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), pp.100-108.

6. Karegowda, A.G., Jayaram, M.A. and Manjunath, A.S., 2012. Cascading k-means clustering and k-nearest neighbor classifier for categorization of diabetic patients. International Journal of Engineering and Advanced Technology, 1(3), pp.147-151.

7. Chang, R.I., Lin, S.Y., Ho, J.M., Fann, C.W. and Wang, Y.C., 2012. A novel content based image retrieval system using K-means/KNN with feature extraction. Comput. Sci. Inf. Syst., 9(4), pp.1645-1661.

8. Zhou, K., Hou, Q., Wang, R. and Guo, B., 2008, December. Real-time KD-tree construction on graphics hardware. In ACM Transactions on Graphics (TOG) (Vol. 27, No. 5, p. 126). ACM.

9. Nielsen, F., Piro, P. and Barlaud, M., 2009, March. Tailored Bregman ball trees for effective nearest neighbors. In Proceedings of the 25th European Workshop on Computational Geometry (EuroCG) (pp. 29-32).

10. Rashad, M. and Semary, N.A., 2014, November. Isolated printed Arabic character recognition using KNN and random forest tree classifiers. In International Conference on Advanced Machine Learning Technologies and Applications (pp. 11-17). Springer, Cham.