CSC 575 - Intelligent Information Retrieval

Winter `14

Final Project

Developing a Movie Recommendation Systems

BY:

Akhil Kumar Ramasagaram

Fizza Sajjad

Table of Contents

1. Introduction

A **recommendation engine** [1] is a feature that filters items by predicting how a user might rate them. It solves the problem of connecting your existing users with the right items in your massive inventory of products or content. Recommender systems have become extremely common in recent years, and are applied in a variety of applications. The most popular ones are probably movies (Netflix), music (Spotify), news, books, research articles, search queries, social tags, and products in general.

Recommender systems typically produce a list of recommendations in one of two ways - through collaborative or content-based filtering. Collaborative filtering approaches building a model from a user's past behaviour as well as similar decisions made by other users; then use that model to predict items that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties

In this project we are going to implement and compare item-based and user-based collaborative filtering systems. And also implement a recommendation approach that clusters users, then compares the target user to the centroid of each cluster. The best match cluster is used as the neighbourhood to generate recommendations.

2. Algorithms

We experimented with a number of different types of algorithms to build recommender systems. To learn more about them, please click on a link below:

- Model based algorithms
    - (i) User based collaborative filtering
    - (ii) Item-based collaborative filtering
    - (iii) Clustering

Algorithms Criteria:

The three important criteria that our team used to determine how useful an algorithm is.

1. Quality of Prediction

This is a pretty obvious one - of course we want our recommender to make good recommendations. More so, we want it to perform better than any "dumb" prediction algorithm which just uses global data, such as an average rating for items.

2. Speed/Scalability

Most recommender systems work in a commercial and/or online setting, and so it is important that they can start making recommendations for a user almost instantly. This means that the algorithm cannot take too long to make any predictions - it has to work, and work fast!

Directly related to speed is the scalability of the algorithm. Again, systems in a commercial and/or online setting can have a huge dataset. The algorithm must maintain its speed even if there are many billions of ratings.

a. Collaborative systems

i. User-Based

The underlying steps in a user-based collaborative filtering systems can be reduced to two steps:

1. Look for users who share the same rating patterns with the active user.
2. Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user.

For our project we chose the 100k movie ratings dataset from the movielens.org [2]. MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.

This data set consists of:

* 100,000 ratings (1-5) from 943 users on 1682 movies.

* Each user has rated at least 20 movies.

* Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up – users who had less than 20 ratings or did not have complete demographic information were removed from this data set. Detailed descriptions of the data file can be found at the end of this file.

We use two datasets here.

- U.data: The full u data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of
  user id | item id | rating | timestamp.

  The time stamps are unix seconds since 1/1/1970 UTC

- U.item: Information about the items (movies); this is a tab separated list of movie id | movie title | release date | video release date | IMDb URL | unknown |

Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi |Thriller | War | Western |

The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data data set.

This concept will be implemented in R language. Below is the algorithm/pseudo code for this method.

Algorithm:

STEP 1: Create a User X Movies Data.

STEP 2: Select a user as a target variable to make recommendations

STEP 3: Calculate the user similarity with respect to the target user.

STEP 4: Select data where the movies are not yet rated.

STEP 5: Multiply the above matrix with the user similarity matrix to get the weighted matrix.

STEP 6: Calculate the sum of a particular movie ratings of all the users.

STEP 7: Calculate the sum of similarities of users who have rated for that particular movies.

STEP 8: Divide Step 6 with Step 7 to get the predicted ratings.

Repeat the steps from STEP 2 for each new user added to our new database.

Code:

```
# Set the Working Directory to the folder where the data is stored
 setwd("C:/Users/Akhilkumar/Desktop/depaul/CSC 575/project/ml-100k/")

# Load the data
u.data <- read.delim("u.data", sep = "\t", header = FALSE)
colnames(u.data) <- c("User.ID","Movie.ID","Ratings","Time")

# Creating a User X Movies Matrix
spm <- matrix(NA, max(u.data$User.ID), max(u.data$Movie.ID))
for (i in 1:length(u.data[,1]))
{
  uid <- u.data[i,1]                # For Each User
  mid <- u.data[i,2]                # For Each Movie
  spm[uid,mid] <- u.data[i,3]       # Corresponding Rating
}

# Loading the u.item data which has the movies names
u.item <- read.delim("u.item", sep = "|", header = FALSE)
colnames(spm) <- u.item[,2]

# Transforming the NA in the User X Movies Matrix to Zero's
spm[is.na(spm)] <- 0

# This is the function which takes only one argument and display
#top 5 recommended movie based the movies which that user has rated.

recommend_user <- function(x){
  library(lsa)
  # user which we are going to predict the movie ratings
  target_user <- spm[x,]
  user_similarity <- matrix(NA, nrow(spm),1)
  for(i in 1:nrow(spm)){
    user_similarity[i,1] <- cosine(spm[i,], target_user)
  }
```

```
# this stores the indexes of the movie which that user has not rated
zero.movie_index <- which(target_user == 0)
movies_actually_rated <- which(target_user != 0)

# Creating a seperate matrix which has all the unrated movies of that user
movies.notrated <- spm[,zero.movie_index]

# Creating a new weighted matrix done by multiplying user similarities and movie ratings
weighted_matrix <- matrix(NA, nrow(movies.notrated), ncol(movies.notrated))
for(i in 1:nrow(user_similarity)){
  weighted_matrix[i,] <- user_similarity[i,1] * movies.notrated[i,]
}

# This is the sum of ratings of each movie
total.movie_rating <- colSums(weighted_matrix)


# This is the sum of similarity measures of users who have rated a particular movies
ss.users_rated <- data.frame()
for(i in 1:ncol(weighted_matrix)){
  # For each movies extracting indices where the movie is actually rated
  temp_index <- which(weighted_matrix[,i] >0)
  ss.users_rated[1,i] <- sum(user_similarity[temp_index])
}

# The predicted movie rating
predicted.ratings <- total.movie_rating/ss.users_rated
colnames(predicted.ratings) <- colnames(spm[,zero.movie_index])
ordered.index <- order(predicted.ratings, decreasing =T)
top5_index <- ordered.index[1:5]
movies_actually_rated <- which(target_user != 0)
rated.movies <- target_user[movies_actually_rated]
movies.predicted <- predicted.ratings[movies_actually_rated]
accuracy <- sum(abs(rated.movies - movies.predicted))/length(rated.movies)
print(paste0("The Mean Accuracy Error is: ",round(accuracy, 4)))
print("Top 5 Recommended Movie are:")
```

```
  return(predicted.ratings[,top5_index])
}
```

Result:

```
> recommend_user(48)
[1] "The Mean Accuracy Error is: 0.9771"
[1] "Top 5 Recommended Movie are:"
[1] "Great Day in Harlem, A (1994)"
[1] "They Made Me a Criminal (1939)"
[1] "Prefontaine (1997)"
[1] "Marlene Dietrich: Shadow and Light (1996) "
[1] "Star Kid (1997)"
```

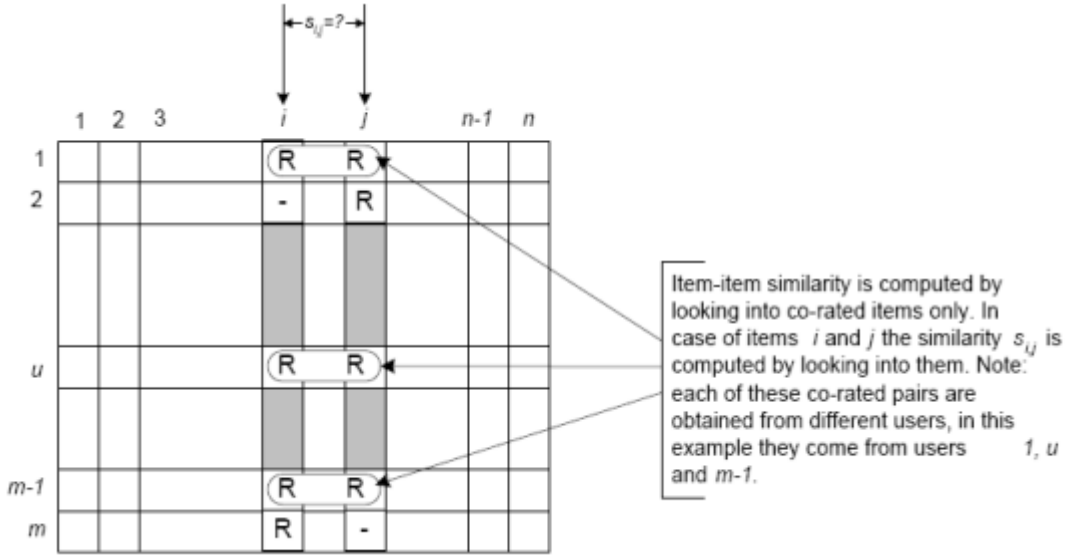
Here, we are generating recommendations for user with ID 48. The Mean Absolute Error for this user is 0.9771 which is quite good. The Top 5 movies for the user are also displayed.

ii. Item-Based

Item-based collaborative filtering is a model-based algorithm for making recommendations. In the algorithm, the similarities between different items in the dataset are calculated by using one of a number of similarity measures, and then these similarity values are used to predict ratings for user-item pairs not present in the dataset.

Similarities between items

The similarity values between items are measured by observing all the users who have rated both the items. The similarity between two items is dependent upon the ratings given to the items by users who have rated both of them:



We have how User based collaborative filtering system worked on recommending movies. Now we will generate recommendation using Item-based Collaborative filtering.

The Main difference between user and item based collaborative filtering method is:

- **Item Based Collaborative Filtering** takes the similarities between items consumption history.
- **User Based Collaborative Filtering** considers similarities between user consumption history.

Algorithm:

Step 1: Create a User X Movies Data.

Step 2: Create a Movie X Movie Similarity Matrix.

Step 3: Select a target user for whom recommendations will be generated.

Step 4: For every un-rated movie, find the most similar movie.

Step 5: Assign the rating of the rated movie to the unrated movie.

Here, we are creating a Movie X Movie Similarity at first. Because if we think in terms of scalability, suppose if we implement this MovieLens 10M data which has ratings of 10,000 movies. If we make a Movies X Movies Matrix, it would include 100M Data Points. Generating the data with such data is going to take a huge amount of time. If we create the matrix at once and store in the memory. Then we can start recommending movie instantly.

Code:

```
 # Set the Working Directory to the folder where the data is stored
 setwd("C:/Users/Akhilkumar/Desktop/depaul/CSC 575/project/ml-100k/")

# Load the data
u.data <- read.delim("u.data", sep = "\t", header = FALSE)
colnames(u.data) <- c("User.ID","Movie.ID","Ratings","Time")

# Creating a User X Movies Matrix
spm <- matrix(NA, max(u.data$User.ID), max(u.data$Movie.ID))
for (i in 1:length(u.data[,1]))
{
  uid <- u.data[i,1]              # For Each User
  mid <- u.data[i,2]               # For Each Movie
  spm[uid,mid] <- u.data[i,3]        # Corresponding Rating
}

# Loading the u.item data which has the movies names
u.item <- read.delim("u.item", sep = "|", header = FALSE)
colnames(spm) <- u.item[,2]

# Transforming the NA in the User X Movies Matrix to Zero's
spm[is.na(spm)] <- 0

movies_sim <- matrix(NA, nrow = ncol(spm), ncol = ncol(spm))
for(i in 1:ncol(spm)){
  for(i in 1:ncol(spm)){
    movies_sim[i,j] <- cosine(spm[,i], spm[,j])
  }
}
colnames(movies_sim) <- u.item[,1]
samp <- movies_sim
for(i in 1:ncol(samp)){
  samp[,i] <- order(samp[,i], decreasing = T)
}
recommend_user_ib <- function(x){
  #library(lsa)
  # user which we are going to predict the movie ratings
  target_user_i <- spm[x,]
  movies_actually_rated_i <- which(target_user_i !=0)
  na.i.movies <- which(target_user_i == 0)
  pred_i <- data.frame()
  for(i in 1:ncol(spm)){
    temp_index <- i
    top_5_similar_movies <- samp[2:21,i]
    ratings_i <- max(spm[x,top_5_similar_movies])
    pred_i[1,i] <- ratings_i
  }
  p_order <- pred_i[,na.i.movies]
```

```
 for(i in 1:ncol(p_order)){
  p_order[,i] <- order(p_order[,i], decreasing = T)
 }
 colnames(p_order) <- colnames(spm[,na.i.movies])
 final_movies_it <- colnames(p_order[1,1:5])
 print(paste0("Top 5 Recommended Movie are:"))
 for(i in 1:5){
  print(paste0(final_movies_it[i]))
 }
 rated_movies_ac <- spm[x,movies_actually_rated_i]
 pred_movies_ac <- p_order[1,movies_actually_rated_i]
 accuracy <- sum(abs(rated_movies_ac - pred_movies_ac)) /
length(rated_movies_ac)
 print(paste0("The Mean Accuracy Error is: ",round(accuracy, 4)))
}
```

Result:

```
> recommend_user_ib(48)
[1] "Top 5 Recommended Movie are:"
[1] "Toy Story (1995)"
[1] "GoldenEye (1995)"
[1] "Four Rooms (1995)"
[1] "Get Shorty (1995)"
[1] "Copycat (1995)"
[1] "The Mean Accuracy Error is: 2.7879"
```

Here, we are generating recommendations for the same user “48”. In this model we are selecting top 20 movie which are supposed to be similar using the similarities measure. But still we have a lot of 0’s as ratings even after prediction. This is due to the sparsity of the data. Out of 1600+ movies, most of user have rated movies somewhere from 20 to 30. So, This is effecting us from generating recommendations.

If we compare the Accuracy of both the collaborative models, then we can see User-based collaborative model performs a lot better than Item-based collaborative filtering.

The User-based filtering model performs better on sparse data on the other hand Item-based filtering model is likely to perform better on the data with less sparsity.

b. Clustering

We have seen how collaborative filtering systems worked. Now we are going to use Clustering to generate recommendations. We are going to use hierarchical clustering.

Strategies for hierarchical clustering generally fall into two types:

- Agglomerative: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- Divisive: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In this project, we are going to use the Divisive approach towards hierarchical clustering.

Algorithm:

Step 1: Create a User X Movie Matrix.

Step 2: Calculate the euclidean distances of the movie genre w.r.t to the movies.

Step 3: Apply Hierarchical Clustering on the data.

Step 4: Divide the model into K parts.

Step 5: Select the target user for whom we generate recommendations.

Step 6: Find the movie with the highest rating.

Step 7: Look up in the cluster for that movies.

Step 8: Present the Top 5 movie from the cluster in Step 7.

For every new user repeat from step 5.

Here, i have used 10 cluster. If we check the amount/percentage that each cluster has w.r.t movie genres. For Example, let us consider which cluster has action movies.

```
> tapply(movies$Action, clusterGroups, mean)
        1         2         3         4         5         6         7         8
0.23323615 0.02040816 0.85714286 0.37500000 0.07017544 1.00000000 0.84000000 0.00000000
        9        10
0.08000000 0.00000000
```

As the movie genres are binary fields, where a 1 determines that the movies belongs to that genre and a 0 determines it doesn't belongs to that genre.

CODE:

```
# Set the Working Directory to the folder where the data is stored
 setwd("C:/Users/Akhilkumar/Desktop/depaul/CSC 575/project/ml-100k/")

# Load the data
u.data <- read.delim("u.data", sep = "\t", header = FALSE)
colnames(u.data) <- c("User.ID","Movie.ID","Ratings","Time")

# Creating a User X Movies Matrix
spm <- matrix(NA, max(u.data$User.ID), max(u.data$Movie.ID))
for (i in 1:length(u.data[,1]))
{
  uid <- u.data[i,1]              # For Each User
  mid <- u.data[i,2]               # For Each Movie
  spm[uid,mid] <- u.data[i,3]        # Corresponding Rating
}

# Loading the u.item data which has the movies names
u.item <- read.delim("u.item", sep = "|", header = FALSE)
colnames(spm) <- u.item[,2]
spm[is.na(spm)] <- 0
colnames(u.item) = c("ID", "Title", "ReleaseDate", "VideoReleaseDate", "IMDB",
"Unknown", "Action", "Adventure", "Animation", "Childrens", "Comedy", "Crime",
"Documentary", "Drama", "Fantasy", "FilmNoir", "Horror", "Musical", "Mystery",
"Romance", "SciFi", "Thriller", "War", "Western")


#Remove unnecessary attributes
u.item$ID = NULL
u.item$ReleaseDate = NULL
u.item$VideoReleaseDate = NULL
u.item$IMDB = NULL

# Saving only unique values.
movies = unique(u.item)
#calculate the distances.
distances = dist(movies[2:20], method = "euclidean")
clusterMovies = hclust(distances, method = "complete")
clusterGroups = cutree(clusterMovies, k = 10)

recommend_user_Clust <- function(x){
  user <- spm[x,]
  highest_rated_movie <- which.max(user)
  CG <- clusterGroups[highest_rated_movie]
  clusterX = subset(movies, clusterGroups==CG)
  final_movies <- clusterX$Title[1:10]
  for (i in 1:5){
    print(paste0(final_movies[i]))
  }
}
```

Result:

```
> recommend_user_Clust(48)
[1] "Four Rooms (1995)"
[1] "Get Shorty (1995)"
[1] "Copycat (1995)"
[1] "Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)"
[1] "Twelve Monkeys (1995)"
```

If you look at the recommendations generated by the Item-based collaborative filtering and compare with recommendation generated by clustering. Then we can see 3 movies are same. So we can say that the clustering model performed almost the same as Item-based collaborative filtering. There is no accuracy here, because we recommended movies based on their distance and not predicted any actual value.

3. Conclusion

We have generated recommendation using different methods. Now the question is when or which method is optimal? From the results we conclude that to design a recommendation engine, starting with clustering model at first. Because, it showed all the top 5 movies according to each genre. So if a new user is added to the system then all top 5 movies from each genre is shown. If a new user started to rate movies, then the system finds the most highly rated movie by that user and finds it in K clusters. It shows recommendation from that cluster. After that, when a user has rated a few movies but not quite many. In that case User-based Collaborative filtering will be used.

4.References

[1] . http://en.wikipedia.org/wiki/Recommender_system

[2]. The data set is availabale at http://files.grouplens.org/datasets/movielens/ml-100k.zip

[3]. Download R language from these mirror links. http://cran.r-project.org/mirrors.html

[4]. Download R Studio from http://www.rstudio.com/products/rstudio/