Brandon Reyes

Final Project Report

IST 707

**Introduction**

Have you at any point thought about how Netflix suggests content, or how Amazon suggests you products they think you’d like? Maybe you've seen comparative suggestions with Instagram followers, or how Facebook will suggest new friends. These suggestions are made possible by the execution of recommender systems.

Recommender systems incorporate strategies and algorithms that can propose likeable things to users. They foresee future conduct dependent on past information through a large number of procedures including framework factorization. We presently live in what some call the "period of wealth". For some random item, there are now and again huge number of choices to look over. Think about the models above: web based recordings, long range informal communication, web based shopping; the rundown continues. Recommender frameworks help to customize a stage and help the client observe something they like.

The least demanding and easiest method for doing this is to suggest the most famous things. In any case, to truly upgrade the client experience through customized suggestions, we want committed recommender frameworks. From a business stance, the more important items a client finds on the stage, the higher their commitment. This frequently brings about expanded income for the actual stage. Different sources say that as much as 35–40% of tech goliaths' income comes from suggestions alone. Since we comprehend the significance of recommender frameworks, we should view kinds of suggestion frameworks, then, at that point, construct our own with publicly released information!

Collaborative-based filtering\*\*

**Analysis and Models**

About the Data

We pulled our data from <https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset>. The data was already given to us a small dataset to avoid any lags in our processing and modeling. The ratings data contained 105,339 observations for 10,329 movies. There were 668 users that made ratings. The data spans from the years 1996 to 2016. Originally the zip folder contained a links dataset, tags dataset, ratings dataset, and movies dataset. To build our recommender system, we used only the movies and ratings datasets.

Movies:

Text

Description automatically generated with low confidence

Text

Description automatically generated

Ratings:

A picture containing text, scoreboard

Description automatically generated

A picture containing text

Description automatically generated

Data Preparation

Firstly, we wanted to create a matrix that would allow us to search for a movie by genre. So we created a dataframe with just the genres column then split up all the genres into columns. There were 18 genres. We had to convert the genre datafram observations from characters to integers. Then we created a dataframe from the titles column of the movies dataset and separated the year out of the titles which you can see in the first six movies of the movies dataset above.

Next, we used several *for* loops to binarize our ratings data. We gave ratings of 4 or more a value of 1 and 3 or less a value of 0. Gave all NAs in the dataset a value of 0 with a *for* loop. This works best because many recommendation models work on binary data. We then removed the *movieId* column from the ratings dataset and also removed the movies that didn’t have ratings so that our models don’t get confused. After removing these movies from the ratings dataset we had to go in a remove those same movies from our genre dataset that we created earlier. To help match the two matrices we took the dot product of the genre matrix and the ratings matrix then converted them to dummy variables.

Exploratory Data Analysis

Chart, histogram

Description automatically generated

Here we can see there is an exponential growth in our movies dataset. This is probably due to the fact that the movie business has grown a lot over the years. The amount of movies released every year gets higher and higher every year. In or around 2016 there was a sharp decline and we think it’s because our dataset only dates until October of 2016. Additionally, the internet wasn’t as prevalent in 1996 as it is now. Keep in mind, the years axis shows the release dates of the movie in our dataset. The ratings is what dates from 1996-2016. The advancement of the internet makes it a lot easier for the broader audience to rate the movie.

Chart, bar chart

Description automatically generated

This is a nice heatmap of the top users and movies. It shows that only a few of the top users don’t have ratings for the top movies. Some have a few ratings. The closer to solid red we get, the better our model will be able to recommend a movie for these users because we have substantial data to use. The closer to solid blue, the harder it is to recommend movies based on these users’ ratings.

Chart, histogram

Description automatically generated

Here we can see the distribution of the average user rating. We see some left skewedness with a bit of normal distribution. Most user give a rating of 3.6-4 stars.

Text

Description automatically generated

Here is a wordcloud look at the most common genres found in our dataset. Those are: Comedy, Drama, Documentary, and Romance. Very interesting to see. Some movies are classified as being a combination of multiple genres. We see a few of these as well.

Models

We chose to build User-Based Collaborative Filtering and Item-Based Collaborative Filtering models. For our recommendation system, there are two kinds of functions we can use within our models: Pearson and Cosine. These functions calculate the similarity between the items.

Pearson correlation measures the association between our users and movies based on ratings and genre. The correlation ranges from -1 to 1. A correlation greater than 0 is a positive correlation. A correlation of 0 is no correlation. A correlation less than 0 is, that’s right, you guessed it, a negative correlation.

A Cosine similarity simply calculates the similarity between two vectors by taking the cosine of the angle between them. Think of a clock. If I want to know which hour is closest to the 12, the 3 or 10. I would take the cosine of the angle between both when the hands are pointing to 12 and 3, and then when they are pointing to 12 and 10. This value will tell me which hour is closer to 12 on the clock. In our case, the closer to 1 the cosine of the angle is, the more similar the items are.

**ITEM-based Collaborative Filtering Model (IBCF)** vs **USER-based Collaborative Model (UBCF)**

A picture containing text, different

Description automatically generated

This is a nice depiction of what the difference between the two models is. The UBCF model works like this: first you select a user with the movies the user has watched🡪 based on the user’s rating of the movies, you find the top k-nearest neighbors🡪 the model then gets the watched movie record of the user for each neighbor 🡪 it then calculates a score of similarity 🡪 and finally it recommends the items with the highest similarity score. The IBCF is similar but it selects the items(movies) with the users that have watched those movies instead.

Using a k-nearest neighbors of 30 for the UBCF model and the cosine method for our first user-profile:

rec\_model <- Recommender(ratmat\_norm,

method = "UBCF",

param=list(method="Cosine",nn=30))

model\_info <- getModel(rec\_model)

View(model\_info)

top\_10 <- predict(rec\_model,

ratmat[1],

n=10)

top\_10

#convert top\_10 to readable list

top\_10\_list <- as(top\_10,

"list")

top\_10\_list

# [1] "3567" "913" "55276" "30803" "27611" "4223" "106489" "5066" "42728" "55052"

#Get names of our recommendations

top\_10\_result <- matrix(0,10)

for (i in 1:10){

top\_10\_result[i] <- as.character(subset(movies,

movies$movieId == as.integer(top\_10\_list[[1]][i]))$title)

}

top\_10\_result

#[1,] "Bossa Nova (2000)"

#[2,] "Maltese Falcon, The (1941)"

#[3,] "Michael Clayton (2007)"

#[4,] "3-Iron (Bin-jip) (2004)"

#[5,] "Battlestar Galactica (2003)"

#[6,] "Enemy at the Gates (2001)"

#[7,] "Hobbit: The Desolation of Smaug, The (2013)"

#[8,] "Walk to Remember, A (2002)"

#[9,] "Tristan & Isolde (2006)"

#[10,] "Atonement (2007)"

We get these movies recommended for our first user-profile. After this, we built our second model using IBCF. After building and running our second model to predict some recommendations for the first item. We get these movie recommendations:

Text

Description automatically generated

Here is a plot of the IBCF Item Amount Distribution.

Chart, histogram

Description automatically generated

We can see our data is right-skewed.

**Results**

We implanted a total of four different models.

models\_comp <- list(

IBCF\_cos = list(name = "IBCF",

param = list(method = "cosine")),

IBCF\_pear = list(name = "IBCF",

param = list(method = "pearson")),

UBCF\_cos = list(name = "UBCF",

param = list(method = "cosine")),

UBCF\_pear = list(name = "UBCF",

param = list(method = "pearson")),

random = list(name = "RANDOM", param=NULL)

)

There was an error in running the UBCF\_pear model. So we stuck with the other three and also a random one and then plotted an ROC curve of them.

Chart, line chart

Description automatically generated

Here we can see that IBCF with cosine similarity has the greatest area under the curve. Let’s check out the precision-recall of IBCF.

Chart, line chart

Description automatically generated

IBCF\_k\_40 has the highest precision-recall. This means that the harmonized mean for this model is the highest compared to others.

**Conclusion**

This project consisted of implementing and evaluating a collaborative filtering recommender system. Let’s talk about the pros and cons of UBCF and IBCF. For the IBCF model, the recommender system is based on the content and it doesn’t require a lot of user data. With just the item data we can start giving recommendations to users. Because of this, it is possible to give recommendations to a user so long as we have the right amount of data to build their user profile. The problem with this is that the item data needs to be well distributed. The recommendation system won’t be practical if it’s content-based and the majority of the movies are the same genre.

For the UBCF model, we get recommendations that can be the perfect companion to