Understanding basics of Recommendation Engines

## **Introduction**

Ever wondered, “what algorithm google uses to maximize its target ads revenue?”. What about the e-commerce websites which advocates you through options such as ‘people who bought this also bought this’. Or “How does Facebook automatically suggest us to tag friends in pictures”?

The answer is Recommendation Engines. With the growing amount of information on world wide web and with significant rise number of users, it becomes increasingly important for companies to search, map and provide them with the relevant chunk of information according to their preferences and tastes.

Companies nowadays are building smart and intelligent recommendation engines by studying the past behavior of their users. Hence providing them recommendations and choices of their interest in terms of “Relevant Job postings”, “Movies of Interest”, “Suggested Videos”, “Facebook friends that you may know” and “People who bought this also bought this” etc.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/few-things2.jpg)

## **What are Recommendation Engines ?**

Often termed as Recommender Systems, they are simple algorithms which aim to provide the most relevant and accurate items to the user by filtering useful stuff from of a huge pool of information base. Recommendation engines discovers data patterns in the data set by learning consumers choices and produces the outcomes that co-relates to their needs and interests.

## **Types of Recommendation Engine:**

In this article, we will explain two types of recommendation algorithms that are also used by most of the tech giants like Google and Facebook in their advanced recommender system modules.

As a typical business problem,

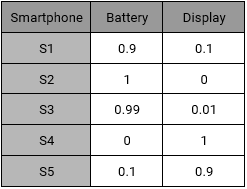
Consider a scenario of an e-commerce website which sells thousands of smartphones. With growing number of customers every day, the task in hand is to showcase the best choices of smartphones to the users according to their tastes and preferences.

To understand how recommendation engine works, let’s slice the data into a sample set of five smartphones with two major features “Battery and Display”. The five smartphones have following properties:

* S1 has good battery life but poor display
* S2 has an amazing battery performance but very rough display
* S3’s battery is one of the best but display lacks quality
* S4 & S5 are good in terms of display but poor in terms of battery performance.

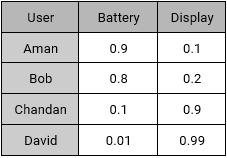
Using these characteristics, we can create an **Item – Feature Matrix**. Value in the cell represents the rating of the smartphone feature out of 1.

**Item – Feature Matrix**

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F1.png)  
Our sample set also consist of four active users with their preferences.

* **Aman**: He prefers battery over display as an ideal smartphone feature.
* **Bob**: He likes a long lasting battery.
* **Chandan**: For Chandan, display should be decent, battery should be normal.
* **David**: For David, Display is extremely important but not the battery.

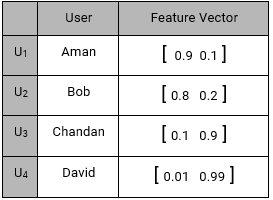
Using their interests, we can create a **User – Feature Matrix** as follows:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F2.png)

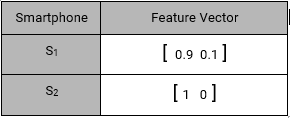
We have two matrices: Item – Feature and User – Feature. We can create the recommendation of smartphones for our users using following algorithms:

## **Content Based Recommendations**

Content based systems, recommends item based on a similarity comparison between the content of the items and a user’s profile. The feature of items are mapped with feature of users in order to obtain user – item similarity. The top matched pairs are given as recommendations, as demonstrated below: Representing every user by a feature vector:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F3.png)

Also, every item representation as a feature vector:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F4.png)

and so on…

Content Based **Item – User Mapping Recommendations** are given by the equation:

MAX ( U(j)T . I(i) )

                                i,j -> n,m

For User U1 (Aman), Smartphone recommendation is:

MAX( U1TS1, U1TS2, U1TS3, U1TS4, U1TS5)

MAX([0.9 0.1]T [0.9  0.1], [0.9  0.1]T [1  0], [0.9  0.1]T [0.99 0.01], [0.9  0.1]T [0.1 0.9],    [0.9  0.1]T [0.01  0.99])

MAX(0.82 , 0.9 , 0.89 , 0.18 , 0.10)

= S2(0.9), S3(0.89) & S1(0.82)

Smartphones S2, S3 and S1 has the highest recommendation scores, Hence S2, S3 and S1 are recommended to Aman.

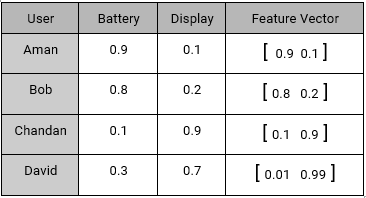
## **Collaborative Filtering**

Content-based recommendation lacks in detecting inter dependencies or complex behaviors. For example: People might like smartphones with Good Display, only if it has retina display and wouldn’t otherwise.

Collaborative Filtering algorithm considers “User Behaviour” for recommending items. They exploit behaviour of other users and items in terms of transaction history, ratings, selection and purchase information. Other users behaviour and preferences over the items are used to recommend items to the new users. In this case, features of the items are not known.

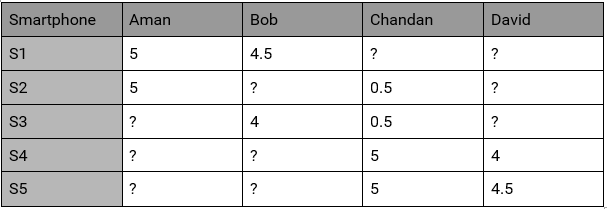
We have a similar **User – Feature Matrix** as content based:

**User – Feature Matrix**

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F8.png)

This time we don’t know features of the items but we have user behaviour. i.e. How the Users brought/rated the existing items.

**User- Behaviour Matrix**

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/10/F7.png)

where values of the behaviour matrix can be described as:

Bi,j = {r , if Uj has given “r” rating to a Si

?, if no rating is given

This user behavior matrix can be used to derive unknown features of the most liked items. Lets try to derive features of S1 using this behavior matrix.

S1 is rated 5 by U1

S1 is rated 4.5 by U2

S1 rating by U3 & U4 are not known

Using this information Feature Vector of S1 can be assumed as:

S1 : [x1 x2]

and the equations are:

U1TS1 = 5

U2TS1 = 4.5

[0.9   0.1]T [x1  x2] = 5

[0.8   0.2]T [x1  x2] = 4.5

0.9 \* x1 + 0.1 \* x2  = 5

0.8 \* x1 + 0.1 \* x2  = 4.5

solving these equations, gives x1 = 5.5 and x2 = 0.5

S1 = [5.5  0.5]

Similarly,

S2 = [5.5  0]

S3 = [5    0]

S4 = [0.5  5.5]

S5 = [2.7  5.25]

Now all the feature vectors are known. Hence the recommendations will be mappings of User Feature Vectors and Item Feature Vectors. Thus for Aman, based on his preferences and behaviours, recommendation will be:

MAX(U1TS1, U1TS2, U1TS3, U1TS4,U1TS5)

MAX([0.9  0.1]T [5.5  0.5] ,[0.9  0.1]T [5.5  0], [0.9  0.1]T [5   0], [0.9  0.1]T

[0.5 5.5],[0.9  0.1]T [2.7   5.25])

MAX(5, 4.99, 4.95, 1, 2.9)

> S1, S2 and S3

which comes out the be S1, S2 and S3 again. Since S1 and S2 are already rated by Aman, So we will recommend him a new smartphone S3.

In the above example where we assumed that there are two primary features of S1 as governed by the users who rated it. In real case, we end up with more number of features than this. For example, if we had data for all the N number of users who rated S1, then feature vector look like:

S1: [ x1 x2 x3 x4 x5 …  ]

# Quick Guide to Build a Recommendation Engine in Python

## **Introduction**

*This could help you in building your first project!*

Be it a fresher or an experienced professional in data science, doing voluntary projects always adds to one’s candidature. My sole reason behind writing this article is to get your started with recommendation systems so that you can build one. If you struggle to get open data, write to me in comments.

Recommendation engines are nothing but an automated form of a “shop counter guy”. You ask him for a product. Not only he shows that product, but also the related ones which you could buy. They are well trained in cross selling and up selling. So, does our recommendation engines.

The ability of these engines to recommend personalized content, based on past behavior is incredible. It brings customer delight and gives them a reason to keep returning to the website.

In this post, I will cover the fundamentals of creating a recommendation system using [GraphLab](https://www.analyticsvidhya.com/blog/2015/12/started-graphlab-python/" \t "_blank) in Python. We will get some intuition into how recommendation work and create basic popularity model and a collaborative filtering model.



## **Topics Covered**

1. Type of Recommendation Engines
2. The MovieLens DataSet
3. A simple popularity model
4. A Collaborative Filtering Model
5. Evaluating Recommendation Engines

Before moving forward, I would like to extend my sincere gratitude to the **Coursera’s**[**Machine Learning Specialization**](https://www.coursera.org/specializations/machine-learning)**by University of Washington**. This course has been instrumental in my understanding of the concepts and this post is an illustration of my learnings from the same.

## **1. Type of Recommendation Engines**

Before taking a look at the different types of recommendation engines, lets take a step back and see if we can make some intuitive recommendations. Consider the following cases:

### **Case 1: Recommend the most popular items**

A simple approach could be to recommend the items which are liked by most number of users. This is a blazing fast and dirty approach and thus has a major drawback. The things is, there is **no personalization**involved with this approach.

Basically the most popular items would be same for each user since popularity is defined on the entire user pool. So everybody will see the same results. It sounds like, ‘a website recommends you to buy microwave just because it’s been liked by other users and doesn’t care if you are even interested in buying or not’.

Surprisingly, such approach still works in places like news portals. Whenever you login to say bbcnews, you’ll see a column of “Popular News” which is subdivided into sections and the most read articles of each sections are displayed. This approach can work in this case because:

* There is division by section so user can look at the section of his interest.
* At a time there are only a few hot topics and there is a high chance that a user wants to read the news which is being read by most others

### **Case 2: Using a classifier to make recommendation**

We already know lots of **classification algorithms**. Let’s see how we can use the same technique to make recommendations. Classifiers are parametric solutions so we just need to define some parameters (features) of the user and the item. The outcome can be 1 if the user likes it or 0 otherwise. This might work out in some cases because of following advantages:

* Incorporates personalization
* It can work even if the user’s past history is short or not available

But has some major drawbacks as well because of which it is not used much in practice:

* The features might actually not be available or even if they are, they may not be sufficient to make a good classifier
* As the number of users and items grow, making a good classifier will become exponentially difficult

### **Case 3: Recommendation Algorithms**

Now lets come to the special class of algorithms which are tailor-made for solving the recommendation problem. There are typically two types of algorithms – Content Based and Collaborative Filtering. You should refer to our [previous article](https://www.analyticsvidhya.com/blog/2016/03/exploring-building-banks-recommendation-system/) to get a complete sense of how they work. I’ll give a short recap here.

1. **Content based algorithms:**
   * **Idea:** If you like an item then you will also like a “similar” item
   * Based on similarity of the items being recommended
   * It generally works well when its easy to determine the context/properties of each item. For instance when we are recommending the same kind of item like a movie recommendation or song recommendation.
2. **Collaborative filtering algorithms:**
   * **Idea:** If a person A likes item 1, 2, 3 and B like 2,3,4 then they have similar interests and A should like item 4 and B should like item 1.
   * This algorithm is entirely based on the past behavior and not on the context. This makes it one of the most commonly used algorithm as it is not dependent on any additional information.
   * For instance: product recommendations by e-commerce player like Amazon and merchant recommendations by banks like American Express.
   * Further, there are several types of collaborative filtering algorithms :
     1. **User-User Collaborative filtering:** Here we find look alike customers (based on similarity) and offer products which first customer’s look alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every customer pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.
     2. **Item-Item Collaborative filtering:** It is quite similar to previous algorithm, but instead of finding customer look alike, we try finding item look alike. Once we have item look alike matrix, we can easily recommend alike items to customer who have purchased any item from the store. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new customer the algorithm takes far lesser time than user-user collaborate as we don’t need all similarity scores between customers. And with fixed number of products, product-product look alike matrix is fixed over time.
     3. **Other simpler algorithms:** There are other approaches like [market basket analysis](https://www.analyticsvidhya.com/blog/2014/08/visualizing-market-basket-analysis/), which generally do not have high predictive power than the algorithms described above.

## **2. The MovieLens DataSet**

We will be using the MovieLens dataset for this purpose. It has been collected by the GroupLens Research Project at the University of Minnesota. MovieLens 100K dataset can be downloaded from [here](http://grouplens.org/datasets/movielens/100k/). It 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)
* Genre information of movies

Lets load this data into Python. There are many files in the **ml-100k.zip** file which we can use. Lets load the three most importance files to get a sense of the data. I also recommend you to read the readme document which gives a lot of information about the difference files.

import pandas as pd

# pass in column names for each CSV and read them using pandas.

# Column names available in the readme file

#Reading users file:

u\_cols = ['user\_id', 'age', 'sex', 'occupation', 'zip\_code']

users = pd.read\_csv('ml-100k/u.user', sep='|', names=u\_cols,

encoding='latin-1')

#Reading ratings file:

r\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp']

ratings = pd.read\_csv('ml-100k/u.data', sep='\t', names=r\_cols,

encoding='latin-1')

#Reading items file:

i\_cols = ['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']

items = pd.read\_csv('ml-100k/u.item', sep='|', names=i\_cols,

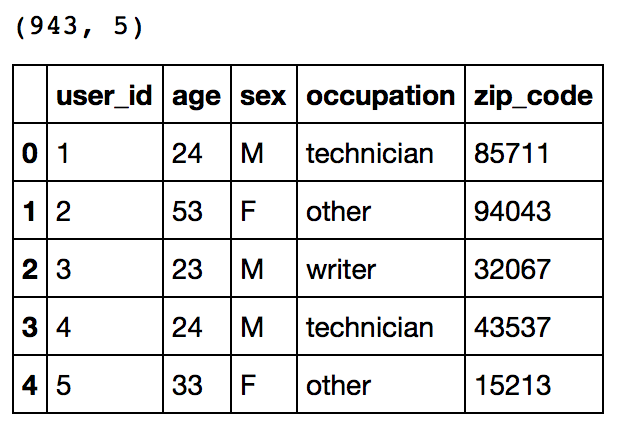
encoding='latin-1')

Now lets take a peak into the content of each file to understand them better.

#### **Users**

print users.shape

users.head()

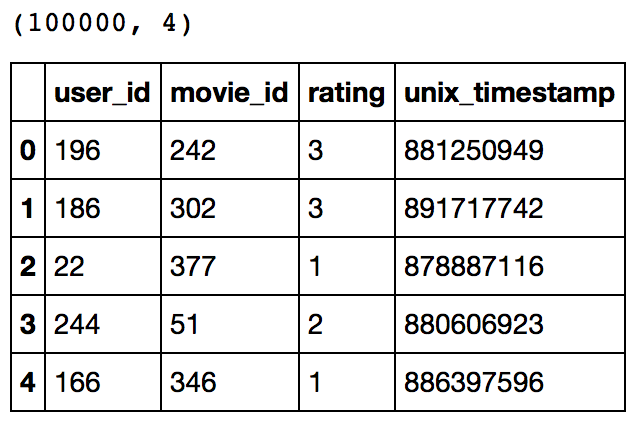
[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/1.-users.png)

This reconfirms that there are 943 users and we have 5 features for each namely their unique ID, age, gender, occupation and the zip code they are living in.

#### **Ratings**

print ratings.shape

ratings.head()

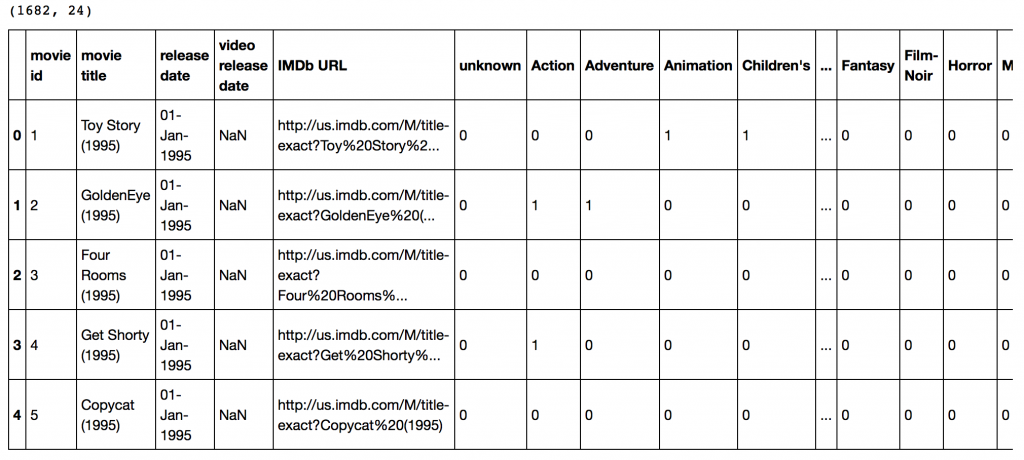
[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/2.-ratings.png)

This confirms that there are 100K ratings for different user and movie combinations. Also notice that each rating has a timestamp associated with it.

#### **Items**

print items.shape

items.head()

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/3.-items.png)This dataset contains attributes of the 1682 movies. There are 24 columns out of which 19 specify the genre of a particular movie. The last 19 columns are for each genre and a value of 1 denotes movie belongs to that genre and 0 otherwise.

Now we have to divide the ratings data set into test and train data for making models. Luckily GroupLens provides pre-divided data wherein the test data has 10 ratings for each user, i.e. 9430 rows in total. Lets load that:

r\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp']

ratings\_base = pd.read\_csv('ml-100k/ua.base', sep='\t', names=r\_cols, encoding='latin-1')

ratings\_test = pd.read\_csv('ml-100k/ua.test', sep='\t', names=r\_cols, encoding='latin-1')

ratings\_base.shape, ratings\_test.shape

Output: ((90570, 4), (9430, 4))

Since we’ll be using GraphLab, lets convert these in SFrames.

import graphlab

train\_data = graphlab.SFrame(ratings\_base)

test\_data = graphlab.SFrame(ratings\_test)

We can use this data for training and testing. Now that we have gathered all the data available. Note that here we have user behaviour as well as attributes of the users and movies. So we can make content based as well as collaborative filtering algorithms.

## **3. A Simple Popularity Model**

Lets start with making a popularity based model, i.e. the one where **all the users have same recommendation** based on the most popular choices. We’ll use the  graphlab recommender functions popularity\_recommender for this.

We can train a recommendation as:

popularity\_model = graphlab.popularity\_recommender.create(train\_data, user\_id='user\_id', item\_id='movie\_id', target='rating')

Arguments:

* **train\_data**: the SFrame which contains the required data
* **user\_id**: the column name which represents each user ID
* **item\_id**: the column name which represents each item to be recommended
* **target:** the column name representing scores/ratings given by the user

Lets use this model to make top 5 recommendations for first 5 users and see what comes out:

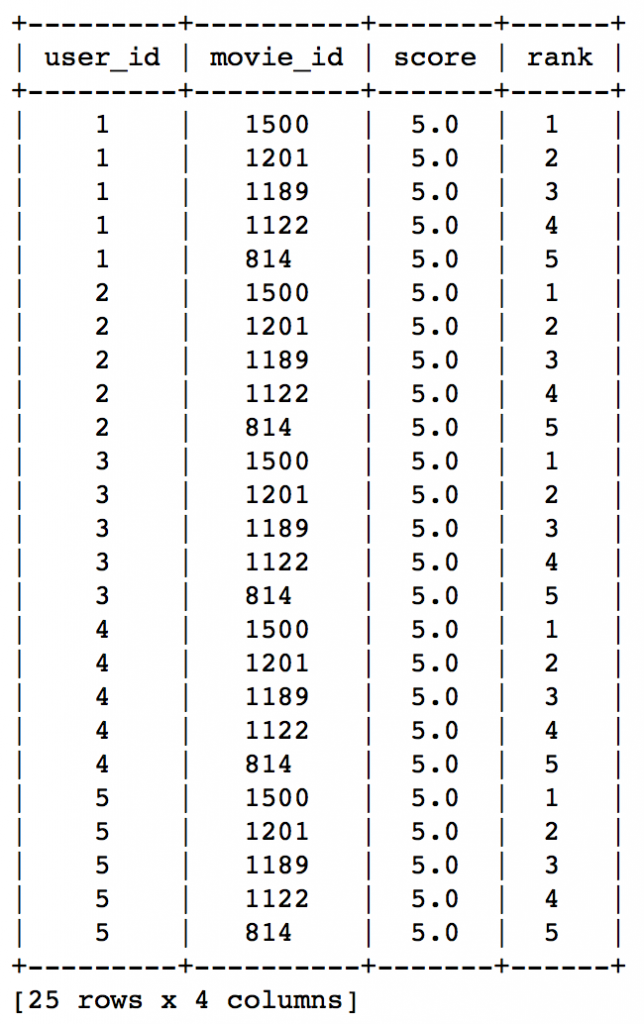
#Get recommendations for first 5 users and print them

#users = range(1,6) specifies user ID of first 5 users

#k=5 specifies top 5 recommendations to be given

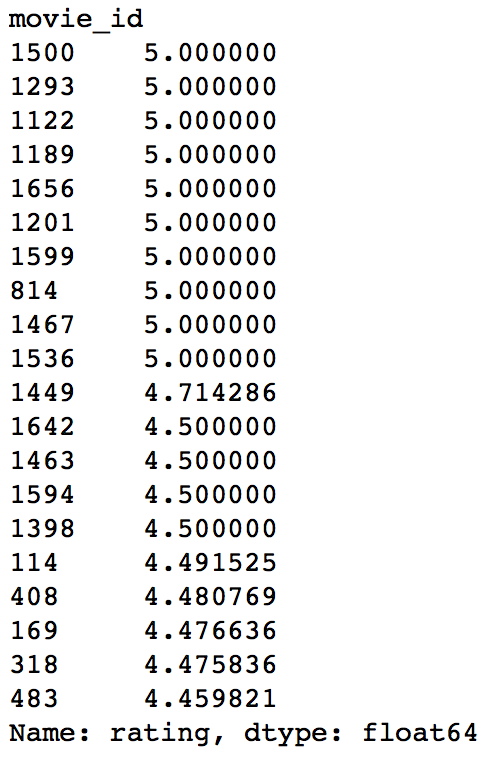
popularity\_recomm = popularity\_model.recommend(users=range(1,6),k=5)

popularity\_recomm.print\_rows(num\_rows=25)

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/4.-popularity-recomm.png)

Did you notice something? The recommendations for all users are same – 1500,1201,1189,1122,814 in the same order. This can be verified by checking the movies with highest mean recommendations in our ratings\_base data set:

ratings\_base.groupby(by='movie\_id')['rating'].mean().sort\_values(ascending=False).head(20)

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/5.-mean-ratings.png)

This confirms that all the recommended movies have an average rating of 5, i.e. all the users who watched the movie gave a top rating. Thus we can see that our popularity system works as expected. But it is good enough? We’ll analyze it in detail later.

## **4. A Collaborative Filtering Model**

Lets start by understanding the basics of a collaborative filtering algorithm. The core idea works in 2 steps:

1. Find similar items by using a similarity metric
2. For a user, recommend the items most similar to the items (s)he already likes

To give you a high level overview, this is done by making an **item-item matrix** in which we keep a record of the pair of items which were rated together.

In this case, an item is a movie. Once we have the matrix, we use it to determine the best recommendations for a user based on the movies he has already rated. Note that there a few more things to take care in actual implementation which would require deeper mathematical introspection, which I’ll skip for now.

I would just like to mention that there are 3 types of item similarity metrics supported by graphlab. These are:

1. **Jaccard Similarity:**
   * Similarity is based on the number of users which have rated item A and B divided by the number of users who have rated either A or B
   * It is typically used where we don’t have a numeric rating but just a boolean value like a product being bought or an add being clicked
2. **Cosine Similarity:**
   * Similarity is the cosine of the angle between the 2 vectors of the item vectors of A and B
   * Closer the vectors, smaller will be the angle and larger the cosine
3. **Pearson Similarity**
   * Similarity is the pearson coefficient between the two vectors.

Lets create a model based on item similarity as follow:

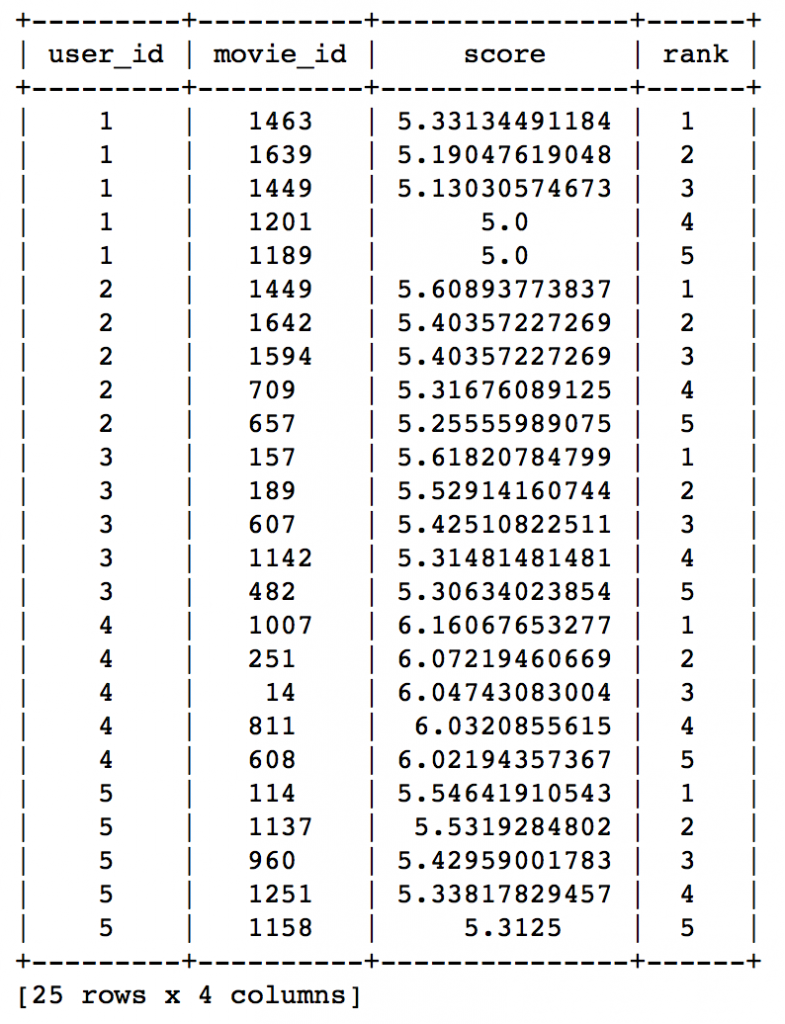
#Train Model

item\_sim\_model = graphlab.item\_similarity\_recommender.create(train\_data, user\_id='user\_id', item\_id='movie\_id', target='rating', similarity\_type='pearson')

#Make Recommendations:

item\_sim\_recomm = item\_sim\_model.recommend(users=range(1,6),k=5)

item\_sim\_recomm.print\_rows(num\_rows=25)

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/6.-similarity-model-1.png)

Here we can see that the recommendations are different for each user. So, personalization exists. But how good is this model? We need some means of evaluating a recommendation engine. Lets focus on that in the next section.

## **5. Evaluating Recommendation Engines**

For evaluating recommendation engines, we can use the concept of precision-recall. You must be familiar with this in terms of classification and the idea is very similar. Let me define them in terms of recommendations.

* **Recall:**
  + What ratio of items that a user likes were actually recommended.
  + If a user likes say 5 items and the recommendation decided to show 3 of them, then the recall is 0.6
* **Precision**
  + Out of all the recommended items, how many the user actually liked?
  + If 5 items were recommended to the user out of which he liked say 4 of them, then precision is 0.8

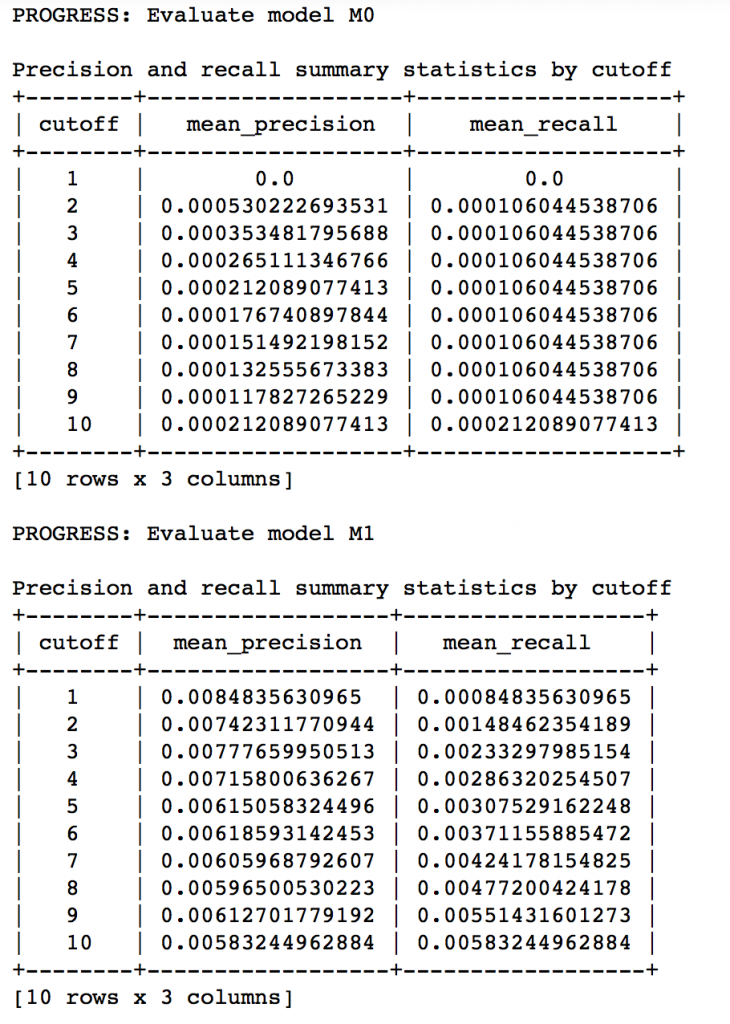
Now if we think about recall, how can we maximize it? If we simply recommend all the items, they will definitely cover the items which the user likes. So we have 100% recall! But think about precision for a second. If we recommend say 1000 items and user like only say 10 of them then precision is 0.1%. This is really low. Our aim is to maximize both precision and recall.

An idea recommender system is the one which only recommends the items which user likes. So in this case precision=recall=1. This is an optimal recommender and we should try and get as close as possible.

Lets compare both the models we have built till now based on precision-recall characteristics:

model\_performance = graphlab.compare(test\_data, [popularity\_model, item\_sim\_model])

graphlab.show\_comparison(model\_performance,[popularity\_model, item\_sim\_model])

[](https://www.analyticsvidhya.com/wp-content/uploads/2016/06/7.-evaluate.png)

Here we can make 2 very quick observations:

1. The item similarity model is definitely better than the popularity model (by atleast 10x)
2. On an absolute level, even the item similarity model appears to have a poor performance. It is far from being a useful recommendation system.

There is a big scope of improvement here. But I leave it up to you to figure out how to improve this further. I would like to give a couple of tips:

1. Try leveraging the additional context information which we have
2. Explore more sophisticated algorithms like matrix factorization

In the end, I would like to mention that along with GraphLab, you can also use some other open source python packages like the following:

* [**Crab**](http://muricoca.github.io/crab/).
* [Surprise](https://github.com/NicolasHug/Surprise)
* [Python Recsys](https://github.com/ocelma/python-recsys)
* [MRec](https://github.com/Mendeley/mrec)

## **End Notes**

In this article, we traversed through the process of making a basic recommendation engine in Python using GrpahLab. We started by understanding the fundamentals of recommendations. Then we went on to load the MovieLens 100K data set for the purpose of experimentation.

Subsequently we made a first model as a simple popularity model in which the most popular movies were recommended for each user. Since this lacked personalization, we made another model based on collaborative filtering and observed the impact of personalization.

Finally, we discussed precision-recall as evaluation metrics for recommendation systems and on comparison found the collaborative filtering model to be more than 10x better than the popularity model.

# Exploring Recommendation System (with an implementation model in R)

## **How do we decide the performance metric of such engines?**

Good question! We must know that performance metrics are strongly driven by business objectives. Generally, there are three possible metrics which you might want to optimise:

1. **Based on dollar value:** If your overall aim is to increase profit/revenue metric using recommendation engine, then your evaluation metric should be incremental revenue/profit/sale with each recommended rank. Each rank should have an unbroken order and the average revenue/profit/sale should be over the expected benefit over the offer cost.
2. **Based on propensity to respond:** If the aim is just to activate customers, or make customers explore new items/merchant, this metric might be very helpful. Here you need to track the response rate of the customer with each rank.
3. **Based on number of transactions:** Some times you are interested in activity of the customer. For higher activity, customer needs to do higher number of transactions. So we track number of transaction for the recommended ranks.
4. **Other metrics:** There are other metrics which you might be interested in like satisfaction rate or number of calls to call centre etc. These metrics are rarely used as they generally won’t give you results for the entire portfolio but sample.

## **Building an Item-Item collaborative filtering Recommendation Engine using R**

Let’s get some hands-on experience building a recommendation engine. Here, I’ve demonstrated building an item-item collaborative filter recommendation engine. The data contains just 2 columns namely individual\_merchant and individual\_customer. The data is available to download – [Download Now](https://www.analyticsvidhya.com/wp-content/uploads/2016/03/data.csv).

The code is easy to understand. Hence, I haven’t explained it explicitly. If you find any part of code hard to understand, ask me in comments section below.

#load libraries  
> library(plyr)  
> library("arules")  
> library(readr)

#load data  
#This file has two columns inidividual\_merchant and inidividual\_customer  
> input <- read\_csv("Transaction\_file.csv")  
  
#Get the list of merchants/items  
> merchant <- unique(input$individual\_merchant)  
> merchant <- merchant[order(merchant)]  
> target\_merchants <- merchant  
> sno <- 1:length(target\_merchants)  
> merchant\_ident <- cbind(target\_merchants,sno)  
  
#Create a reference mapper for all merchant  
> colnames(merchant\_ident) <- c("individual\_merchant","sno")  
  
# Create a correlation matrix for these merchants  
> correlation\_mat = matrix(0,length(merchant),length(target\_merchants))  
> correlation\_mat = as.data.frame(correlation\_mat)  
> trans = read.transactions("Transaction\_file.csv", format = "single", sep = ",", cols =  
c("inidividual\_customer", "individual\_merchant"))  
> c <- crossTable(trans)  
> rowitem <- rownames(c)  
> columnitem <- colnames(c)  
> correlation\_mat <- c[order(as.numeric(rowitem)),order(as.numeric(columnitem))]  
> for(i in 1:9822) {  
        correlation\_mat[i,] <- correlation\_mat[i,]/correlation\_mat[i,i]  
   }  
> colnames(correlation\_mat) <- target\_merchants  
> rownames(correlation\_mat) <- merchant  
  
# Now let's start recommending for individual customer  
> possible\_slots <- 20  
> avail <- 21  
> merch\_rec <- matrix(0, nrow = length(target\_customers), ncol = avail)  
> merch\_rec[,1] <- unique(input3$Cust\_map)  
> correlation\_mat <- as.matrix(correlation\_mat)  
> position <- 1  
> for (i in 1:length(target\_customers)) {  
   been\_thr <- input[position : (position + customer\_merch\_ct[i] - 1),'individual\_merchant']  
   merging <- as.data.frame(merchant\_ident[merchant\_ident[,'individual\_merchant'] %in%          been\_thr,])  
   corel\_subset <- correlation\_mat[merging$sno,]   
   will\_go <- colSums(corel\_subset)   
   will\_go\_merch <- target\_merchants[order(-will\_go)]  
   not\_been\_there <- will\_go\_merch[!will\_go\_merch %in% been\_thr]  
   will\_go\_propensity <- will\_go[order(-will\_go)][!will\_go\_merch %in% been\_thr]  
   merch\_rec[i,2:avail] <- not\_been\_there[1:possible\_slots]   
   position <- position + customer\_merch\_ct[i]   
 }

## **End Notes**

Recommended engines have become extremely common because they solve one of the commonly found business case for all industries. Substitute to these recommendation engine are very difficult because they predict for multiple items/merchant at the same time. Classification algorithms struggle to take in so many classes as the output variable.

In this article, we learnt about the use of recommendation systems in Banks. We also looked at implementing a recommendation engine in R. No doubt, they are being used across all sectors of industry, with a common aim to enhance customer experience.