ECEN 765-600
Machine Learning with Networks

Academic Project Report

Recommendation System based on Collaborative Filtering with focus on Cold Start

Submitted by:
Ankit Yadav(UIN 726006435)
ankityadav270796@tamu.edu

Project Guide: Dr. Xiaoning Qian

Department of Electrical and Computer Engineering
Texas A&M University
College Station, TX – 77843

Abstract

This work has been done as a part of the academic project for subject **"Machine Learning with Networks".** A lot of research has been performed on the recommendation system based on collaborative filtering, but the cold start issue seems to have been ignored in most of the studies. Hence, I wish to dedicate this project to address the cold start problem. The purpose of this project is not to improvise any existing techniques but to identify if the user's demographic scores or any explicit information can have any impact while addressing the cold start problem in the collaborative filtering system.

First, a basic recommendation system was implemented, it is being used as the base model. Any of the implemented models are compared to this model to see if there has been any improvement. Overall 4 models were implemented including the basic one.

Also, apart from the project implementation, I have dedicated time to learn about the recent research work being done on "Cold Start problem in Recommendation System" and identify the areas where this project can be further taken to.

Introduction

Over the last decade, there has a rapid growth in internet users. The growth in internet users between 2000-2018 is more than 1000 percent and it' still growing. Companies like Amazon, Flipkart, eBay, Netflix, and YouTube introduce new stuff every single day.

There is a high probability of the user being deluged with the content getting uploaded into these websites if the content was not personalized, and soon users would lose their interest. Here comes the recommendation system for the rescue. In terms of volume, the recommendation system is able to handle a big amount of data. Although there has been a good amount of research on this topic, still this remains a topic of interest given the significance.

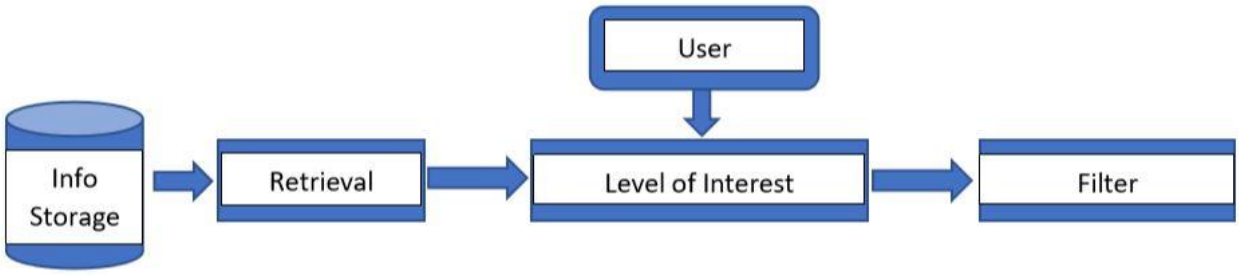
While looking for the similar projects, I noticed that most of them have not tried to consider the "Cold Start Problem" i.e. showing recommendation for a new user or new product that our system has no data available for. In this project, I expect to deal with that part of the problem. Apart from this, my plan is to do a literature survey on "Cold Approach Problem" in the recommendation system.

1.1 Recommendation System

Recommendation system suggests items to a user based on his/her past behaviors, their personal data, their physical location or based on some form of similarity to others. I seem to date back the presence of commendation system in my life since I had my first mobile phone in 2004, it was capable to show me favorite contacts based on who I talk to more or message.

Therefore, a recommendation system is a system that is capable of collecting information, processing and showing personalized results to the user. Some most common examples of the recommendation system are: YouTube & Facebook news feed or friend suggestion, LinkedIn job suggestion or TV series or movies recommendation on the platform like HULU or Netflix.

As stated, that a recommendation system works on the data from the user e.g. their search pattern, visited places or pages etc. In many situations, the system is unaware of this information. That makes it very difficult for the system to predict the data for a new user. This is called cold approach problem.



1.2 Cold Start Issue in the recommendation system

Any recommendation system works on the data collected from the user e.g. their search pattern, visited places or pages etc. In many situations, the system is unaware of this information; e.g. A new user visiting an e-commerce website, a new person joining a job portal.

In the collaborative filtering based recommendation system, the recommendations are made based on the past ratings from the user, these ratings are not available from the users.

That makes it very difficult for the system to predict the data for a new user. This is called cold approach problem.

1.3 Purpose

The whole project revolves around the cold start problem in Collaborative filtering-based recommendation systems. First, a basic collaborative filtering-based recommendation system was built and then one with the variations to address the cold start problem. The purpose of this project is not to improvise the existing collaborative filtering-based recommendation system but to see if the available information from the user can be used to ease the recommendation process.

Also, I did a literature survey on some recent papers in the reputed journals. This was done to keep up with the recent research on this area and identify somewhere this project can be extended in the future.

2. Basics of a Recommendation System

As mentioned in the introduction that a recommendation system is used to ease the problem of information overload, information overload is when the user finds it really difficult to find the right thing. The information overload problem is addressed by either one of them:

1. Information Retrieval
2. Information Filtering

Therefore, the solution to the information overload problem can be dealt with one of them.

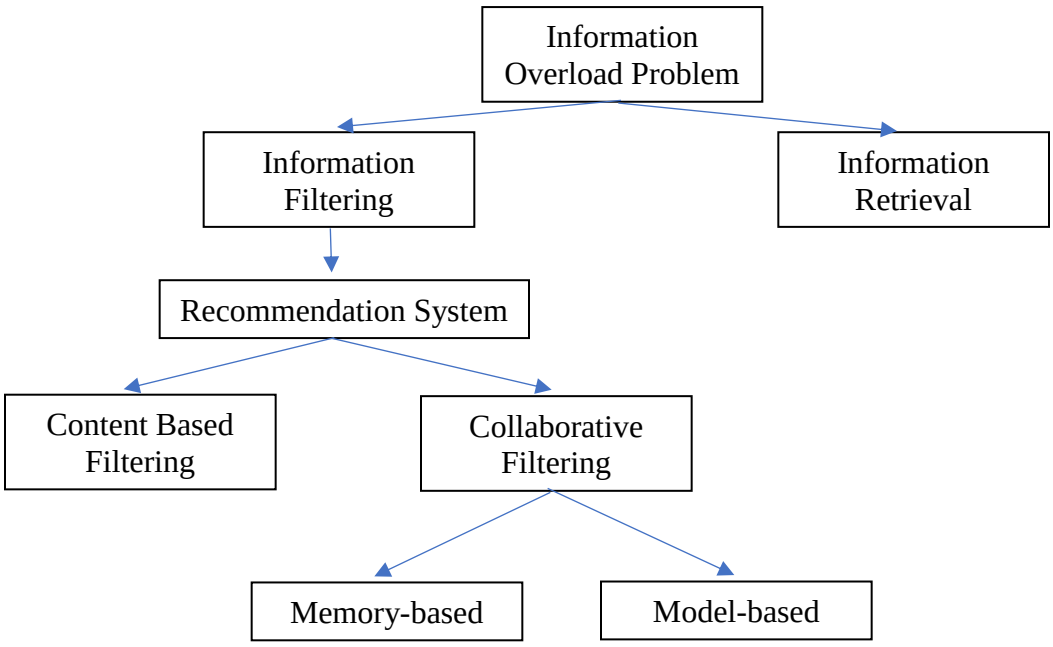


Figure: Information Overload Problem

Information Retrieval Systems directly ask users for the information they need to narrow down the results while the Information Filtering Systems learns it from the user's profile, e.g. his past search results, age etc.

As the collaborative filtering is a part of the Information Filtering, so the focus of this report would be on Information Filtering Side.

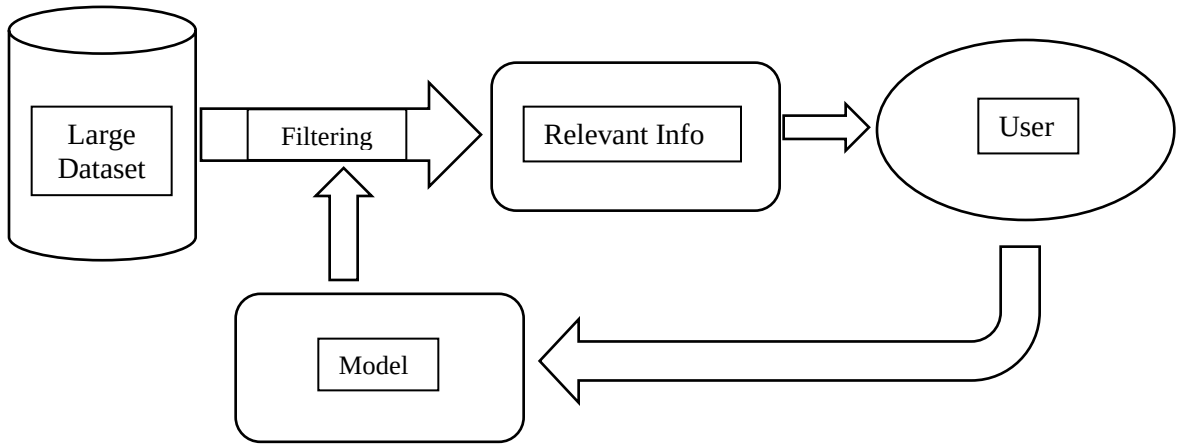


Figure: Information Filtering

The first recommendation system was built by Goldberg in 1992, this recommendation system was based on solving the problem of numerous amounts of emails presented in the inbox. This system was a type of collaborative filtering where users rate their emails that were sent to the user's inbox.

Since the inception of the first recommendation, the basic idea has remained the same to offload the not required information, but the overall architecture of the system and applications have witnessed innumerable changes.

As the number of applications that use the recommendation system has been growing, the need for a precise and accurate recommendation has gone up too. Companies and the universities who deployed this technique into their system have reported that these techniques are really very beneficial.

A recommendation system can be summarized with the help of figure 1.1

2.1 Basic Recommendation Systems developed in past years

1. **Collaborative Filtering:** In this type of system, the recommendations are based on the ratings given by the user and the similarity between them.

2. **Content-Based Filtering:** It is based on the closeness between the products.

3. **Demographic Filtering:** This type of system uses the user's demographic information like age, occupation, and location etc.

4. **Social Filtering:** In this system, the user's social network's information is used to make the recommendation.

5. **Hybrid Filtering:** This can simply be a combination of one or more of the systems described above.

As the whole project is based on Collaborative filtering, therefore the next section has been devoted to exploring a little about them.

2.2 Content-Based Filtering

In the type of system, the items that have already been rated by different users are considered and similarity between them is computed. This makes the basis of the recommendation.

As mentioned earlier in the report Information retrieval and Information Filtering. In this user is not trying to make a search or query, the system has to learn about the user depending on his past behavior and selection and present the best recommendation. This technique is also known as cognitive filtering because the recommendation is based on implicitly learning the user behavior then directly asking for the required information. This type of filtering has taken some concepts from the information retrieval.

In the information filtering, a vector is generated from the user's past behavior, this vector represents the user profile to the system. Depending on this profile vector, the similarity with different products is calculated. Basically, this vector represents a user's behavior in a way that can be interpreted by the computer.

2.2 Collaborative Filtering

This type of system relies on the user's ratings, which is used to calculates the similarity. It is one of the most common techniques used these days because it is both easier to interpret & implement and gives good results.

Any collaborative Filtering based system focuses on two points: how to calculate a similarity vector and how to use it to make the predictions.

In simple words, this technique depends on the model that is generated from the user's profile. This model can be based on the profile of a single user or a group/collection of users' profiles as the name also suggests.

Collaborative filtering-based algorithms can be divided into two parts:

1. **Model-Based Filtering:** In this type of filtering the system doesn't have to go through the whole dataset instead a predictive dataset is generated from the available data, which is used whenever we have to make a recommendation.
   Example: Item-Based Filtering Algorithm

2. **Memory-Based Filtering:** In this type of filtering algorithm the system has to go through the whole dataset, hence it takes up a lot of resources to build and running it at runtime requires a lot of processing power.
   Example: User-Based recommendation system

2.2.1 Item-based Collaborative Filtering

As mentioned above Item-based Collaborative Filtering, also referred to as IBCF is an example of a Model-Based Filtering technique.

In this method, a similarity matrix is required to be calculated. While making the recommendation, the relationship between items is taken into consideration. Examples of the similarity measures: Cosine Similarity, Pearson Similarity.

In the layman terms, we maintain a matrix, that stores the k similar items which are similar to this product. To make the recommendation more accurate, the rating given by the similar user is preferred.

For the prediction, we just calculate the similarity score with one of the similarity measures. And the one with the highest score is considered.

2.2.2 User-based Collaborative Filtering

As described above, this comes in the category of the memory-based filtering technique, it is also referred to as UBCF. The whole behind this is based on that the users who have similar interests in the items would give a similar rating to the products.

Therefore, If we can find the users who have given a similar rating to some products would have a similar choice. This can be used to make predictions.

2.2.3 k Nearest Neighbor Algorithm

This algorithm has been widely used in the algorithm, that is why it requires a mention in the report. It is both easy to understand and gives a great performance. This algorithm divides the data into classes, the similarity between the point is simply the distance between them.

While making the prediction, this algorithm would find out the k most close point. It would find out to which class most of the points belong, the test item is said to belong to that class.

Because the kNN doesn't generalize, therefore this algorithm has to maintain the whole dataset for the testing phase. This is one of the reasons why the training phase is quite fast while testing is a little slow.

In simple terms, we pick a constant k. This denotes how many points would be there in a class. The k points that are very similar to each other are classified into a class. kNN can be simply interpreted with the help of a cartesian plot.

2.2.4 Hybrid Filtering Techniques

In the hybrid filtering technique, we combine two or more of the algorithms. This can definitely provide great results but comes with a cost of more processing.
Example: Content-based filtering based on demographics.

3. Literature Survey

This part of the report has been dedicated to the literature survey of existing research work which deals with the cold start problem.

Cold start problem has been one of the most challenging problems while building a recommendation system. Imaging opening a youtube page on incognito mode, everything looks so weird. If you manage to click one of those links, you whole feed gets messed up. Hence, it is very important to address the cold start issue.

Cold start problem can be divided into two parts, when a new user logs into the system and when a new product is introduced in the system.

When a new user logs into the system, very little information is present about that user so the predictions can be very inaccurate. Many of the recommendation systems deal with this problem by asking to rate technique, simply they ask the user to rate some of the products. While in case of the new product the collaborative filtering is quite hard because the collaborative filtering is based on the collaborative filtering and it's not present for a new product. Content-based filtering works quite better in this case.

In the coming sections, I have tried to address some of the existing research work published in the reputed journals

Using Demographic Information to Reduce the New User Problem in Recommender Systems

In this paper, authors have tried to use the demographics information present in the MovieLens 100K dataset. I am also using the same dataset in this project. This dataset includes the 100 thousand ratings from 963 users. All these ratings are for 1682 movies. Ratings are on a scale of 5.

Demographic information has the following format:

user id | age | gender | occupation | zipcode

User ratings are present in this format:
user id | item id | rating | timestamp

The dataset is then further divided into test and training datasets, a model with k different groups/clusters are created. This is done with the training dataset while in the training process. This model takes the user demographics into consideration while making predictions.

Given the plot shows the predicted ratings for a given user based on the information present in that database.

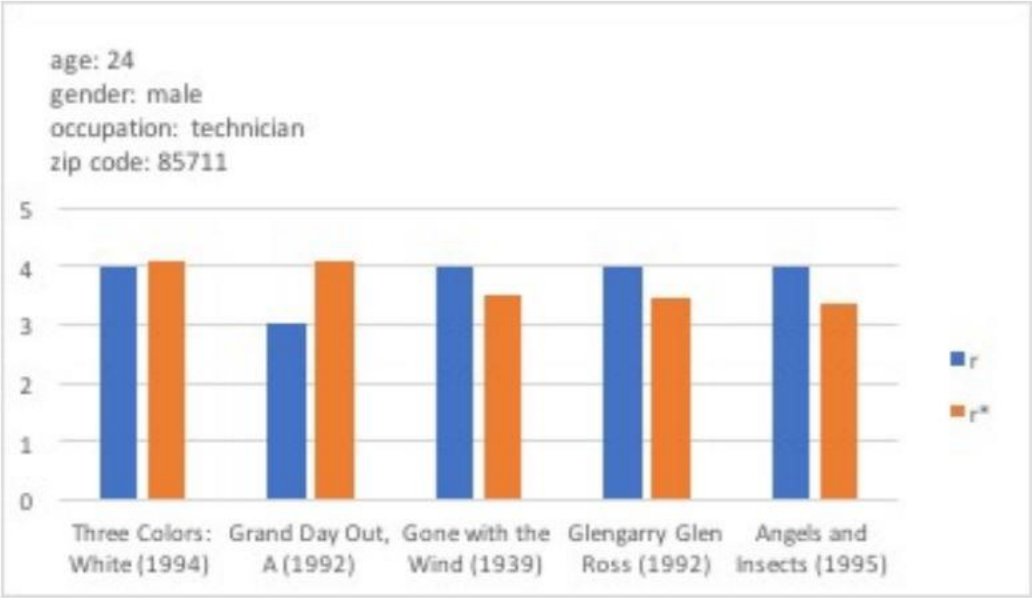


Figure: r* denotes the predicted rating using the given information in the plot that is occupation, gender, age, and zip code.

This plot shows that there exists a relationship between the demographic data and movie rating.

Collaborative Filtering Enhanced By Demographic Correlation

In this paper, the author has combined some of the existing algorithms with the demographic data. This paper introduced two algorithms named U-Demographic and I-Demographic. Where “U” in the first name comes from the user-based collaborative filtering while the ”I” in the second name comes from

the user-based collaborative filtering. The author in this paper has further divided the dataset into different age groups. It gives a better representation of the data and results are more accurate.

feature #	feature contents	comments
1	age <= 18	• each user belongs to a single age group, • the corresponding slot takes value 1 (true) • the rest of the features remain 0 (false)
2	18 < age <= 29	
3	29 < age <= 49	
4	age > 49	
5	male	• the slot describing the user gender is 1 • the other slot takes a value of 0
6	female	
7-27	occupation	• a single slot describing the user occupation is 1 • the rest of the slots remain 0

Figure: Demographic information used by the author.

A new user is molded into a vector form filling up all the fields in the above chart in binary format. The results show the performance is much better than the base model.

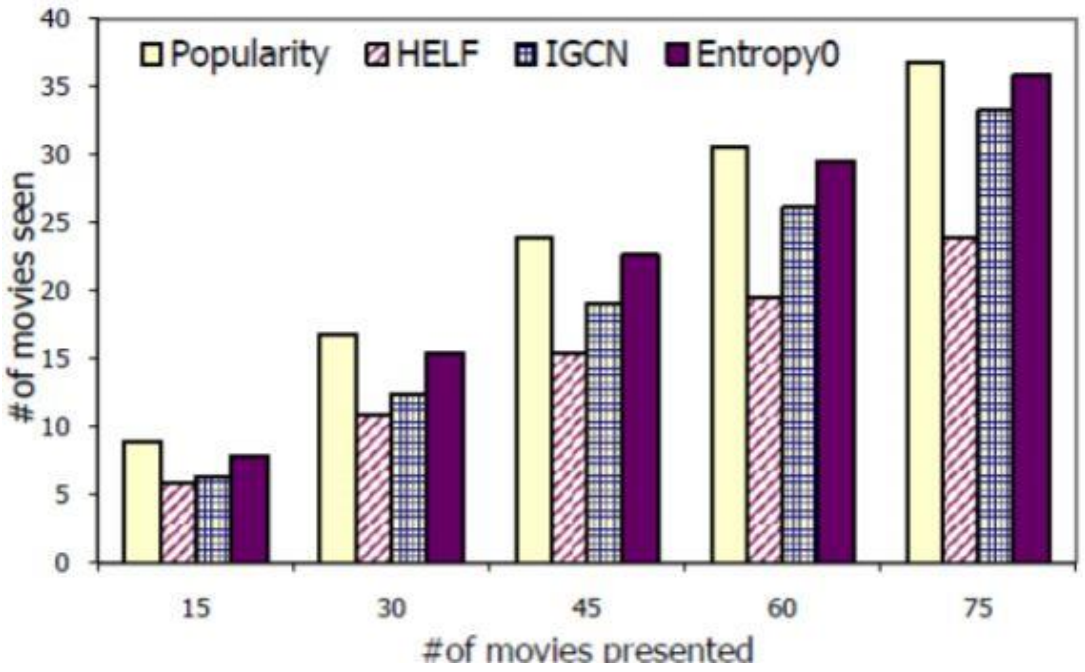
Cold-start Problem in Collaborative Recommender Systems: Efficient Methods Based on Ask-to-rate Technique

This paper tries to enhance the old ask to rate technique, depending on the rating first the similarity with a user in the dataset is calculated. Then from the active user list products are recommended.

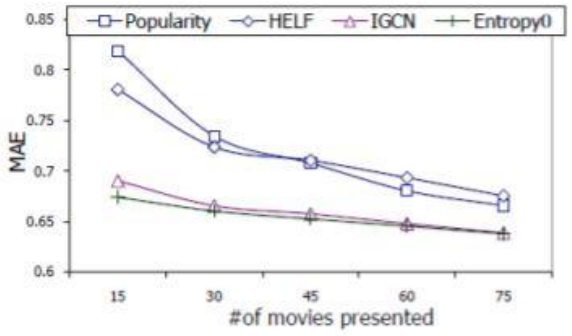
Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach

In this paper, the author has tried to address the cold start problem by calculating the effectiveness of different methods that are based on information theory.

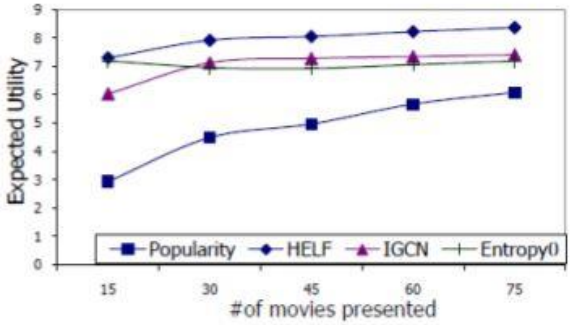
Some techniques were developed to extract information from new users to learn about them and make recommendations.



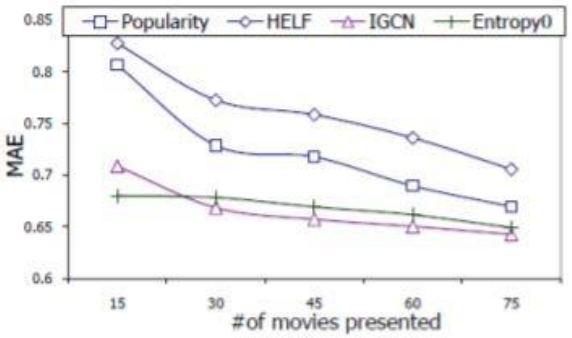
As observed in the previous section, the author also realized the problem with the entropy method, so he proposed a new variation named as entropy0 and HELF.
The author worked on 4 methods: HELF, Popularity, IGCN, and Entropy0. It comes out to be true that the popularity method is the most effective one.


Popularity
HELF
IGCN
Entropy0
MAE
0.85
0.8
0.75
0.7
0.65
0.6
15
30
45
60
75
#of movies presented

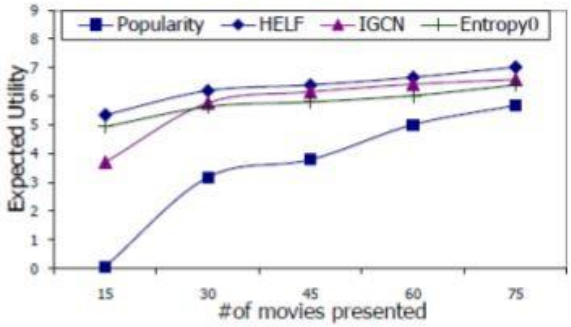
(a)


Expected Utility
9
8
7
6
5
4
3
2
1
0
Popularity
HELF
IGCN
Entropy0
15
30
45
60
75
#of movies presented

(b)


Popularity
HELF
IGCN
Entropy0
MAE
0.85
0.8
0.75
0.7
0.65
0.6
15
30
45
60
75
#of movies presented

(c)


Expected Utility
9
8
7
6
5
4
3
2
1
0
Popularity
HELF
IGCN
Entropy0
15
30
45
60
75
#of movies presented

(d)

4. System Architecture

This system is an idea developed on top of the ideas present in the literature survey. I am using ask to rate technique to handle the cold start issue, every time a new user joins in the website, they are shown an initial set of recommendation and asked to rate them. If the system can pick the initial set of movies which are more expected to be seen by a user, the system would easily get the initial ratings from the user, now these ratings can be used to do the collaborative filtering. The better our system is at predicting that initial set of movies, faster it would converge and better at solving the cold start problem.

Input: MovieLens 100k dataset(1683 movies rated by 982 users)
Output: Recommended movies:

1) **On a random basis:** In this type of recommendation system, the initial set of movies that were asked to be rated are picked up on a random basis.

2) **Movies for rating based on the demographic score:** In this type of recommendation system, the similarity between the users is calculated based on their demographics e.g. age, gender, location. This similarity helps us pick the initial set of movies that are presented to the user to be rated. presented based on the demographic information.

3) **Movies for rating based on the popularity:** This type of system assumes that popular movies have been seen by most of the people, hence easier to collect ratings for these. Later, we will see in the results that this assumption is incorrect.

kNN algorithm has been used to implement this system, also the bias (average rating given by the user is subtracted). This is done because some users are too generous while giving ratings while some are too harsh. Subtracting the average rating value before doing the calculating helps in removing the bias. The average is added back at the end after the filtering is done.

4.1 Dataset

As mentioned in the earlier chapter, MovieLens 100k dataset has been used for this project. This data contains a lot of information, so the required information has been taken from it.

The first needed information is user demographics. That is present in the u.user file. The actual format looks like this:
User id | age | gender | occupation | Zipcode

It was converted to this format because the zip code information is simply not required.
User id | age | gender | occupation

For the precise calculation, the same format has been followed as the [4]. All this information is converted into a vector form.

1. Age has been divided into the following group: **0-18, 19-24, 25-30, 31-40, 41-50, 51-60, 61-70, 71-100**
2. 0/1 corresponding to **M/F** for gender identification.
3. The occupation has **21** categories in the dataset.

Age vector is a binary row vector of size [1*8]
Gender is another binary row vector of size [1*2]
Occupation is a binary row vector of size [1*21]

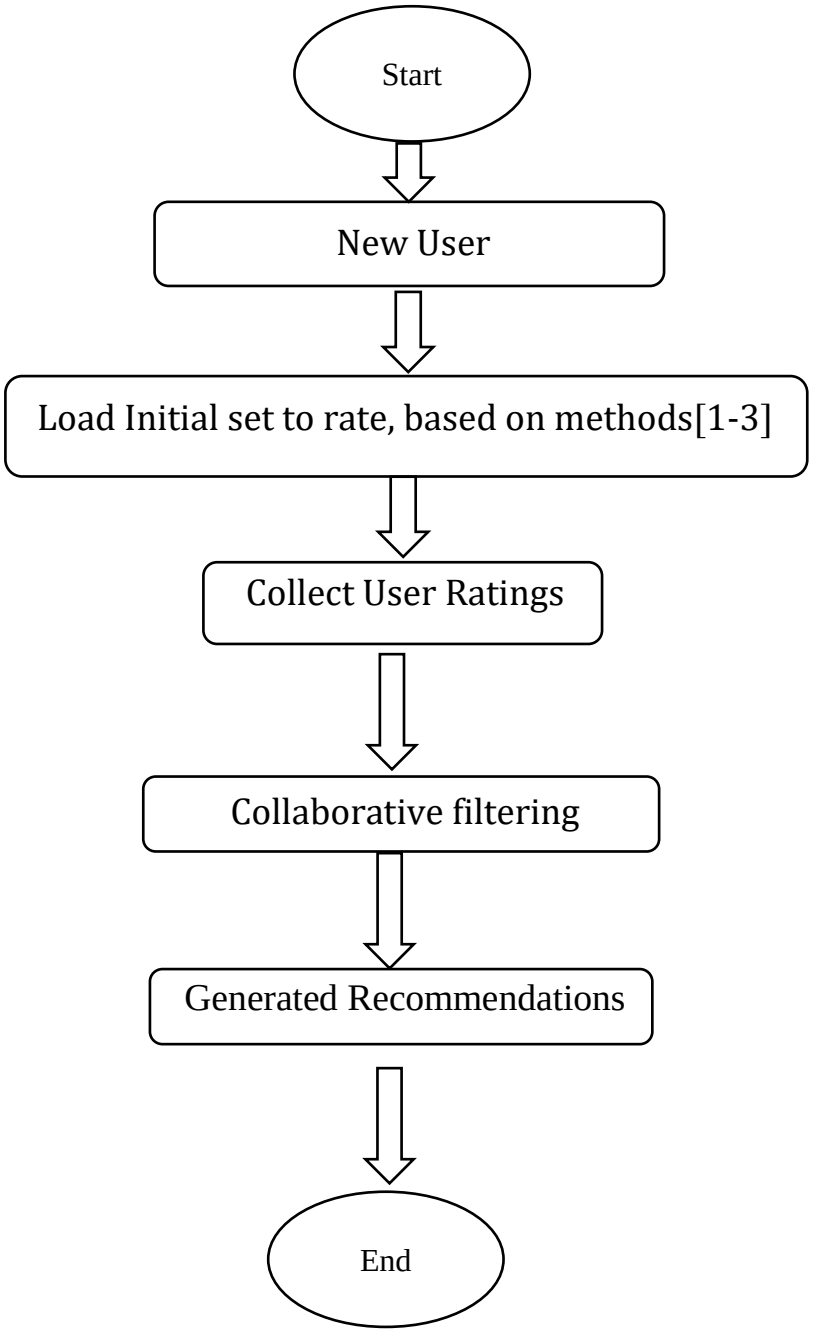
A completed demographic feature looks like: [Age | Gender | Occupation] is of size [1*31]

Example:
1. A 10 year old guy would look like this in the given format:
   [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
2. A 23 year old female scientist would look like this:
   [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

The second required information in the dataset is the user and their corresponding ratings data.
This information is present in the u.data file. It looks like this.
User id | movie id | rating value

Information related to the movie has been given in the u.item file but only movie name is of use when we use the interactive system, otherwise in case of the non-interactive system this dataset is of no use.

4.2 Architecture

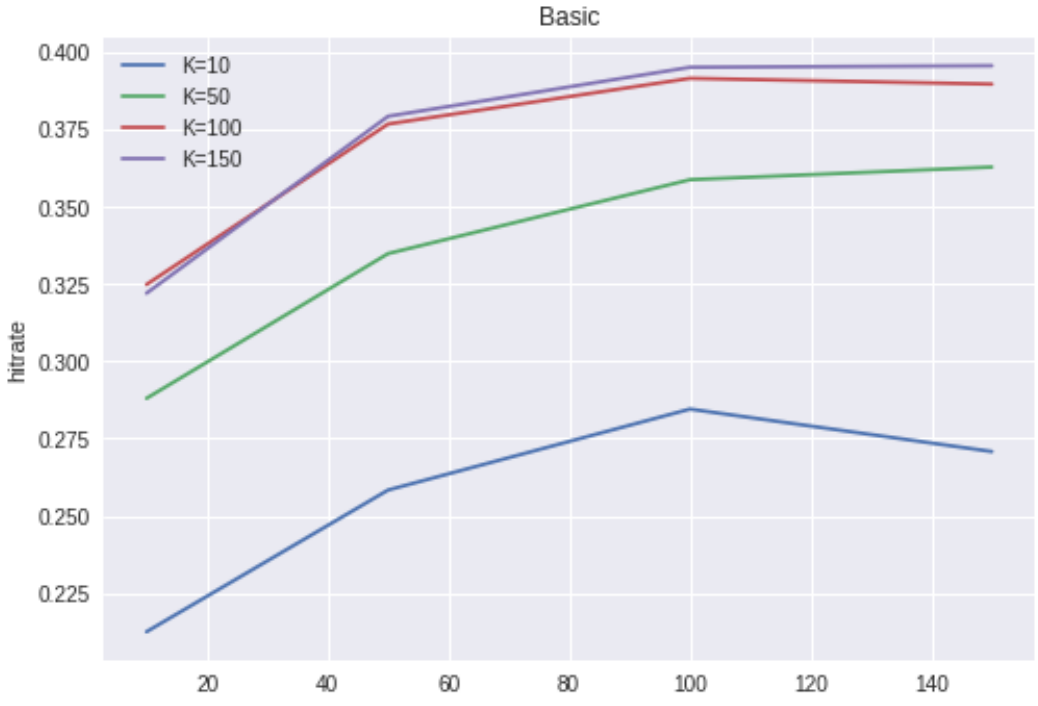


5. Results and Conclusion

To understand the effect of the different parameters on the system's performance, I have considered different combination of Neighborhood size that was used to make the recommendations and an Initial number of movies to rate.

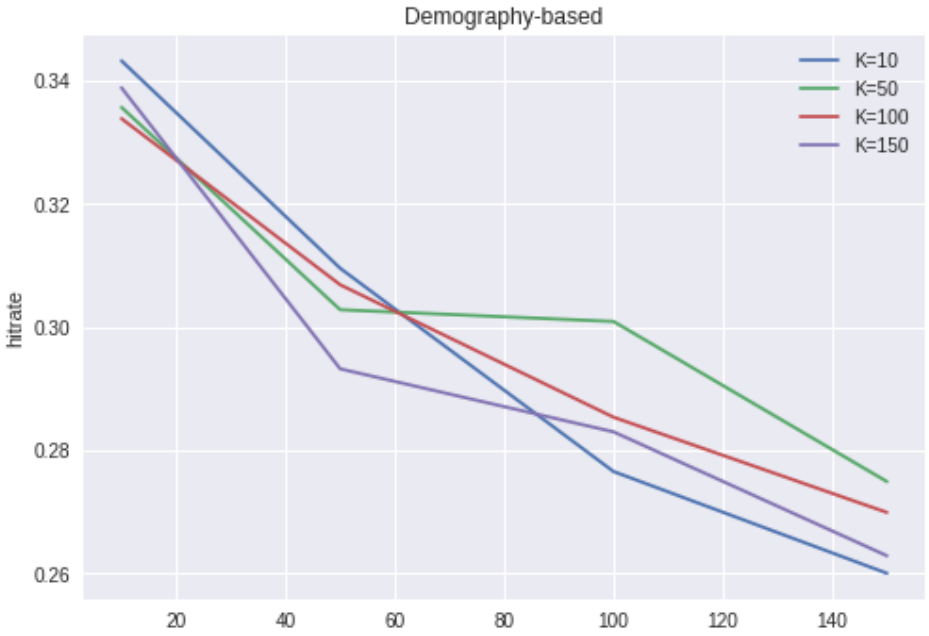
1. K denotes the neighborhood size in the plots.
2. hit-rate(fraction of movies watched from the final recommendation) is plotted on the y-axis
3. Number of movies shown during the ask to rate step is plotted on the x-axis

a) Basic model: This plot corresponds to the basic model explained in the 4.1. The results have been plotted for K=[10, 50, 100, 150]. As in this code, the movies are picked up on a random basis in the ask to rate stage, therefore we get lesser movies watched by the user, hence low hit rate is found. As we keep on increasing the number of movies in the ask to rate step, there is an increase in the final hit-rate because we system gets more information about the user, hence better accuracy. This is why the hit-rate increases with the given number of movies to rate.

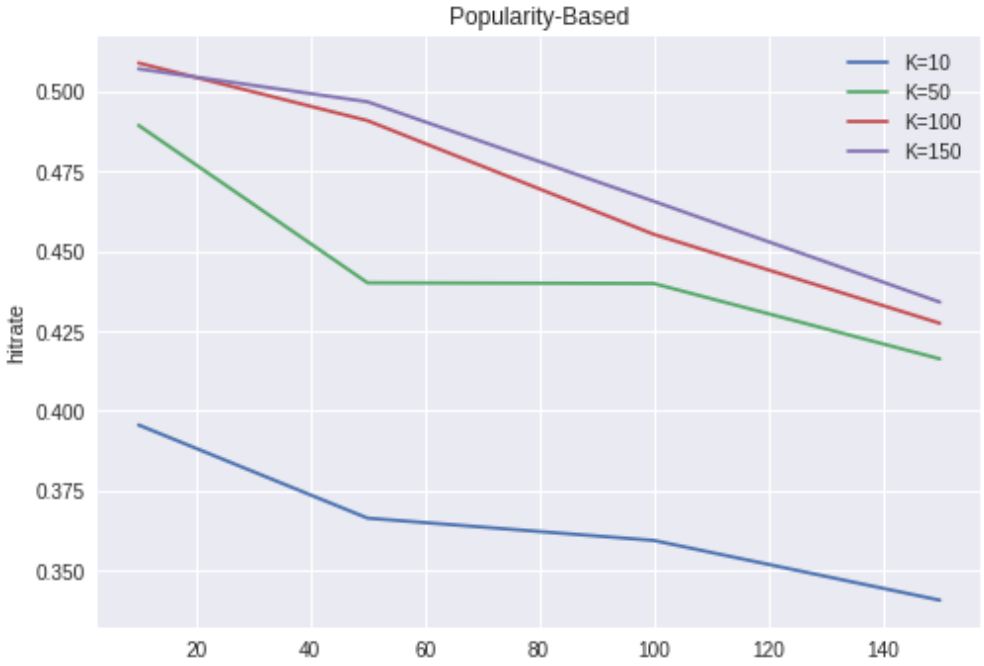


b) Demography Based: This plot corresponds to the collaborative filtering model based that handles the cold start problem using the demographic information of the user. The hit rate vs a number of movies to rate has been

plotted considering different neighborhood size. In this system, it can be observed that the hit-rate decreases as we increase the number of movies to rate because the system tends to exploit the correlation between the demography and item presented in the earlier stages. Increasing number of movies further forces the system to throw junk based on next higher demographic correlation values, hence performance decreases. It's because not every movie has been a user.



c) **Popularity-based:** This plot corresponds to the model explained in the 4.3. It is seen that as we increase the number of movies to rate the final hit rate decreases because the popularity of the movie is calculated by the number of people it has been watched by, once we start increasing the number of the system starts to include not so popular movies too.

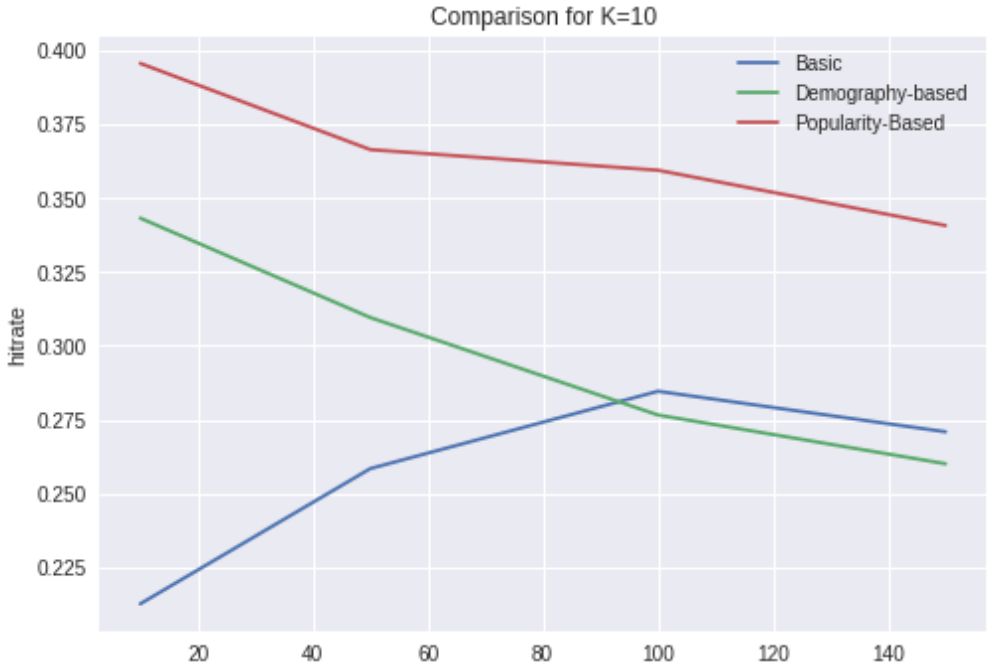


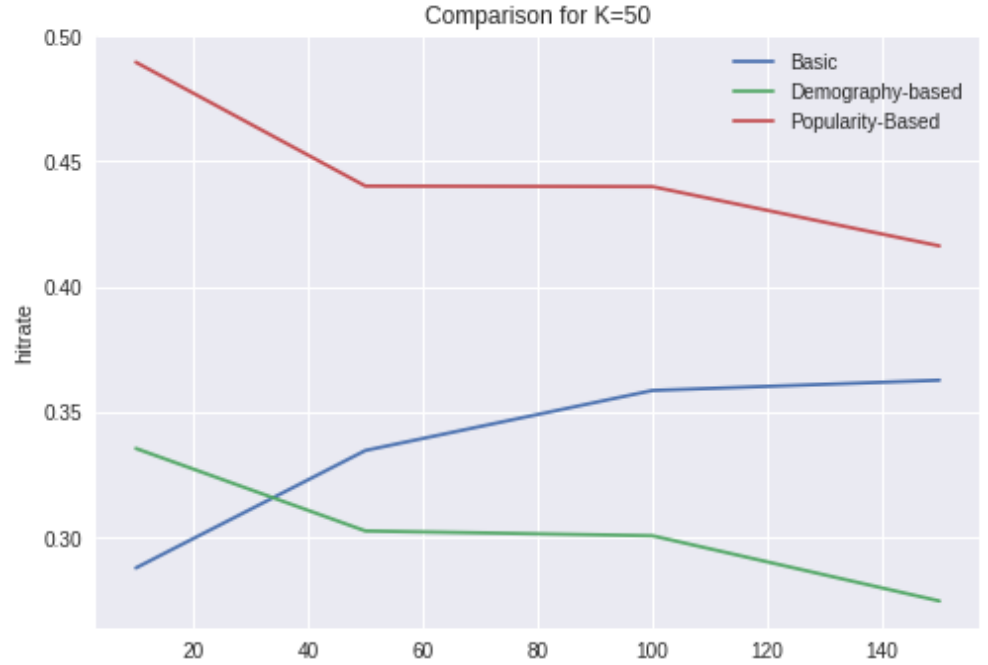
Note: From all the above 3 curves, it can be seen that as we increase the neighborhood size from 10 to 150, neighborhood size starts showing very less impact after it reaches 50 on hit-rate. Here is the comparison of all three methods for neighborhood size of 10 and 50.

d) Comparison for K=10 and K=50

For k=10, we observe that the performance increases with the increase in number of movies to rate for the basic method, because the user tend to find more of the movies watched hence system gets more details. While in case of the demography and popularity-based system, it decreases because the system already tends to use the correlated data, next it keeps throwing junk based on the next higher similarity coefficient hence, no new useful information is gained.

K=50 The same as above is observed but the basic system takes over the demographic one because of the earlier reason.





Conclusion:

1. Popularity based system performs best in all the scenarios, can be verified from the papers presented in the literature survey part.
2. For smaller values of movies given to the users to rate, the demography-based system performs slightly better than the basic one.

Future work

As it is correctly visible that there is a correlation between the demographic information and the choices people make. Therefore, this project can be further expanded for other areas like job portals, e-commerce websites for different kind of products. Unfortunately, it couldn't be done because of the absence of the data, this dataset was just limited to the movies.

Also, if we could introduce more demographic information in the dataset e.g. religion, language, ethnicity information. The performance could be further improved.

References

1. https://machinelearningmastery.com/practical-machine-learningproblems/
2. https://www.ritchieng.com/machine-learning-project-customersegments/
3. Callvik, Johan, Liu, Alva, (2017) "Using Demographic Information to Reduce the New User Problem in Recommender Systems." Kth Royal Institute of Technology School Of Computer Science And Communication
4. Manolis Vozalis, Konstantinos G Margaritis, (2004). "Collaborative Filtering Enhanced By Demographic Correlation." AIAI symposium on professional practice in AI, of the 18th world computer congress.
5. MH Nadimi-Shahraki, M Bahadorpour, (2014). "Cold-start Problem in Collaborative Recommender Systems: Efficient Methods Based on Ask-to rate Technique". CIT. Journal of Computing and Information Technology 22 (2), 105-113 6. Rashid, A.M., Karypis, G., Riedl, J. (2002) "Learning preferences of new users in recommender systems: an information theoretic approach" 127– 134. ACM Press

```
import zipfile
import requests
import matplotlib.pyplot as plt

from numpy import array
from sklearn.metrics.pairwise import pairwise_distances
from random import randint
from scipy.stats import pearsonr
```

```
### Downloading the Database ###
#url = 'http://files.grouplens.org/datasets/movielens/ml-100k.zip'
#r = requests.get(url, allow_redirects=True)
#open('ml-100k.zip', 'wb').write(r.content)


#opener, mode = tarfile.open, 'r:zip'
#cwd = os.getcwd()
```

```
### Funtions ###
def readUserDataset(FilePath):
  head=['UserId','ItemId','Rating','Timestamp']
  return pd.read_csv(FilePath, sep='\t', names=head)

def NumUsers(Dataset):
  return Dataset.UserId.unique().shape[0]

def NumMovies(MovieDataset):
  return MovieDataset.ItemId.unique().shape[0]

def CreateUserItemMat(NumUsers,NumItems,Dataset):
  matrix=np.zeros((NumUsers, NumItems))
  for rating in Dataset.itertuples():

    matrix[rating[1]-1,rating[2]-1]=rating[3]    ### storing a rating corresponding to
  return matrix

def PearsonCorrelation(UserItemMatrix):
  similarity=1-pairwise_distances(UserItemMatrix, metric='correlation')

  similarity[np.isnan(similarity)]=0
  return similarity

def CalculateSimilarity(test, train, num_test, num_train):
  similarity=np.zeros((num_test,num_train))

  for i in range(0,num_test):
    for j in range(0,num_train):
      [p,r]=pearsonr(test[i,:],train[j,:])
      similarity[i,j]=p
      similarity[np.isnan(similarity)] = 0

  return similarity

def InitialRecRandom(test_ratings,num_movies,num_movies_rate):
  #### ratings given by the user to presented movies ####
  final=np.zeros(test_ratings.shape)
  for i in range(test_ratings.shape[0]):

    random_rec=random.sample(range(0, num_movies), num_movies_rate)
    for j in random_rec:
      final[i,j]=test_ratings[i,j]

  return final


def Predict(train_ratings, test_ratings, similarity, k): ### predicts with no bias t
```

```
def Predict(train_ratings, test_ratings, similarity, k):  ### predicts with no bias t
  pred = np.zeros(test_ratings.shape)
  test_user_bias= test_ratings.mean(axis=1)
  train_user_bias= train_ratings.mean(axis=1)

  train_ratings=(train_ratings-train_user_bias[:,np.newaxis]).copy()

  for i in range(test_ratings.shape[0]):

    KTopUsers=[np.argsort(similarity[:,i])[:-k-1:-1]]

    for j in range(test_ratings.shape[1]):
      pred[i,j]=similarity[i,:][KTopUsers].dot(train_ratings[:,j][KTopUsers])
      pred[i,j]/=np.sum(np.abs(similarity[i,:][KTopUsers]))

  pred+= test_user_bias[:,np.newaxis]
  return pred

def plot_comparison(x1,x2,x3,Kvector,name):

    fig = plt.figure()
    for j in range(0,x1.shape[0]):
      lab="K="+str(Kvector[j])
      plt.plot(Kvector, x1[j,:], label=lab)

    plt.xlabel('neighborhood size')
    plt.ylabel('hitrate')
    plt.title(name[0])
    plt.legend()


    fig = plt.figure()
    for j in range(0,x2.shape[0]):
      lab="K="+str(Kvector[j])
      plt.plot(Kvector, x2[j,:], label=lab)

    plt.xlabel('neighborhood size')
    plt.ylabel('hitrate')
    plt.title(name[1])
    plt.legend()

    fig = plt.figure()
    for j in range(0,x3.shape[0]):
      lab="K="+str(Kvector[j])
      plt.plot(Kvector, x3[j,:], label=lab)

    plt.xlabel('neighborhood size')
    plt.ylabel('hitrate')
    plt.title(name[2])
    plt.legend()

    fig = plt.figure()
    plt.plot(Kvector, x1[0,:], label=name[0])
    plt.plot(Kvector, x2[0,:], label=name[1])
    plt.plot(Kvector, x3[0,:], label=name[2])
    plt.xlabel('neighborhood size')
    plt.ylabel('hitrate')
    plt.title('Comparison for K=10')
    plt.legend()

    fig = plt.figure()
    plt.plot(Kvector, x1[1,:], label=name[0])
    plt.plot(Kvector, x2[1,:], label=name[1])
    plt.plot(Kvector, x3[1,:], label=name[2])
    plt.xlabel('neighborhood size')
    plt.ylabel('hitrate')
    plt.title('Comparison for K=50')
    plt.legend()

#### supporting functions for the demographics #####
def ReadDemography(path):
```

```
def ReadDemography(path):
    with zipfile.ZipFile(path) as datafile:
      return datafile.read('ml-100k/u.user').decode(errors='ignore').split('\n')

def CreateMetadeta(rawDemo, users_age, users_occup,users_meta_data):
  for user in rawDemo:
    if not user:
      continue

    splt=user.split('|')
    userid=int(splt[0])
    age = int(splt[1])
    gender = splt[2]
    occup = splt[3]

    i=0
    for m in users_age:
      if(age <= int(m)):
        break ##user belongs to this group
      else:
        i=i+1

    if(gender=='M'):
      j=8
    else:
      j=9

    k=10
    for temp in users_occup:
      if(occup==temp):
        temp
      else:
        k=k+1

    s= str(userid)+"|"
    for l in range (0,31):
      if(l==i or l==j or l==k):
        s=s+"1|"
      else:
        s=s+"0|"

    s=s[:-1]
    users_meta_data.append(s)

  return users_meta_data

def DemMatrix(users_meta,num_users):
  dem_matrix=np.zeros((num_users,30))
  i=0

  for user in users_meta:
    splt=user.split('|')

    for j in range (1,31):
      dem_matrix[i,j-1]=int(splt[j])
    i=i+1

  return dem_matrix

def InitialRec(train_ratings, test_ratings, similarity, k, num_movies_rate): #### demo
  pred = np.zeros(test_ratings.shape)
  #test_user_bias= test_ratings.mean(axis=1)
  #train_user_bias= train_ratings.mean(axis=1)
  #train_ratings=(train_ratings-train_user_bias[:,np.newaxis]).copy()

  for i in range(test_ratings.shape[0]):

    KTopUsers=[np.argsort(similarity[:,i])[:-k-1:-1]]
    for j in range(test_ratings.shape[1]):

      pred[i,j]=similarity[i,:][KTopUsers].dot(train_ratings[:,j][KTopUsers])
```

```
        pred[i,j]=similarity[i,:][KTopUsers].dot(train_ratings[:,j][KTopUsers])
        pred[i,j]/=np.sum(np.abs(similarity[i,:][KTopUsers]))
    #pred+= test_user_bias[:,np.newaxis]

    #### ratings given by the user to presented movies ####
    final=np.zeros(test_ratings.shape)
    for i in range(test_ratings.shape[0]):
      topKrec=[np.argsort(pred[:,i])[:-num_movies_rate-1:-1]]

      for j in topKrec:
        final[i,j]=test_ratings[i,j]

    final[np.isnan(final)] = 0
    return final

def CalcRMSE(V1,V2):
    temp=0
    V1[np.isnan(V1)] = 0
    V2[np.isnan(V2)] =0

    for i in range(0,V1.shape[0]):
      for j in range(0,V1.shape[1]):
        temp=temp+pow(V1[i,j]-V2[i,j],2)

    temp=temp/(V1.shape[0]*V1.shape[1])
    rmse=pow(temp,1/2)
    return rmse

def HitRate(test_ratings, predicted, k):
    total_ratings=(test_ratings.shape[0])*k
    miss=0
    for i in range(test_ratings.shape[0]):
      topK=[np.argsort(predicted[i,:])[:-k-1:-1]]
      zero_els = np.count_nonzero(test_ratings[i,topK]==0)
      miss=miss+zero_els

    hit_rate=1-miss/total_ratings
    return hit_rate

num_iter=3
```

```
##### Basic Part #####
#### reading the dataset #######
Dataset=readUserDataset("u.data")

#UserDataset.tail()
#UserDataset.head()

num_users=NumUsers(Dataset)
num_movies=NumMovies(Dataset)

#### creating the user item matrix ####
user_item_mat=CreateUserItemMat(num_users,num_movies,Dataset)

#### Splitting the dataset into 80:20 training vs test dataset ####
num_test_users=round(0.2*num_users)
num_train_users=num_users-num_test_users



k=[10,50,100,150] ###### Neighborhood size
movie_set_size=[15,30,45,60] ##### Number of Movies shown to the user
#### to store the result #####
basic_result=np.zeros((4,4))
hr1=np.zeros((4,4))

i=0
```

```
i=0
for group_size in k:
  j=0
  for num_movies2rate in movie_set_size:
    temp_rmse=0
    hit_rate=0
    for iter in range(0,num_iter):
      seperator=random.sample(range(0, num_users), num_users)

      #### grouping the users into test and training sets
      test_users=array(seperator[:num_test_users])
      train_users=array(seperator[num_test_users:])

      #### creating test and train user item matrix ####
      test_user_item=user_item_mat[test_users,:]
      train_user_item=user_item_mat[train_users,:]

      ##### Initial set of movies for ask2rate approach
      ask2rate=InitialRecRandom(test_user_item, num_movies, num_movies2rate)

      ##### creating similarity matrix #####
      similarity=CalculateSimilarity(ask2rate, train_user_item, int(num_test_users), i

      ##### calculating the predictions #####
      predictions=Predict(train_user_item, ask2rate, similarity, group_size)

      #temp_rmse=temp_rmse + CalcRMSE(predictions,test_user_item)

      hit_rate=hit_rate+HitRate(test_user_item, predictions, num_movies2rate)/num_iter

    print("Group size: {0}, No of Movies to rate: {1} Hit-Rate: {2}".format(group_size
    #basic_result[i,j]=temp_rmse

    hr1[i,j]=hit_rate
    j=j+1
  i=i+1
```

```
/usr/local/lib/python3.6/dist-packages/scipy/stats/stats.py:3010: RuntimeWarning: i
  r = r_num / r_den
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:60: RuntimeWarning: in
Group size: 10, No of Movies to rate: 15 Hit-Rate: 0.1969429747207525
```

```
##### Demography Based #####
demography=ReadDemography("ml-100k.zip")
```

```
#### reading the dataset #######
Dataset=readUserDataset("u.data")

Dataset.tail()
Dataset.head()

num_users=NumUsers(Dataset)
num_movies=NumMovies(Dataset)



##### create metadata for the demographics #####
users_age = ['18', '24', '30', '40', '50', '61', '70', '100']
users_occup = ['administrator', 'artist', 'doctor', 'educator', 'engineer', 'entertair
users_combined_features = ['18|0', '24|1', '30|2', '40|3', '50|4', '61|5', '70|6', '1(

users_meta=[]
users_meta = CreateMetadeta(demography,users_age,users_occup, users_meta)

DemVectors=DemMatrix(users_meta,num_users)

#print(DemVectors[354,:])

#### creating the user item matrix ####
user_item_mat=CreateUserItemMat(num_users,num_movies,Dataset)

#### Splitting the dataset into 80:20 training vs test dataset ####
num_test_users=round(0.2*num_users)
num_train_users=num_users-num_test_users

k=[10,50,100,150] ###### Neighborhood size
movie_set_size=[15,30,45,60] ##### Number of Movies shown to the user

#### to store the result #####
dem_result=np.zeros((4,4))
hr0=np.zeros((4,4))

i=0
for group_size in k:
  j=0
  for num_movies2rate in movie_set_size:
    temp_rmse=0

    hit_rate=0
    for iter in range(0,num_iter):

      seperator=random.sample(range(0, num_users), num_users)

      #### grouping the users into test and training sets
      test_users=array(seperator[:num_test_users])
      train_users=array(seperator[num_test_users:])

      #### creating test and train user item matrix ####
      test_user_item=user_item_mat[test_users,:]
      train_user_item=user_item_mat[train_users,:]

      #### dividing demographic data ####
      test_dem=DemVectors[test_users,:]
      train_dem=DemVectors[train_users,:]

      #### Calculating the demographic similarity ####
      DemSim=CalculateSimilarity(test_dem, train_dem, int(num_test_users), int(num_tra

      k=group_size #users considered.
      movies2rate=num_movies2rate

      #### Asked to rate based rating from user ####
      ask2ratings=InitialRec(train_user_item, test_user_item, DemSim, k, movies2rate)

      #### Collaborative filtering ####
```

```
      #### Collaborative Filtering ####
      similarity=CalculateSimilarity(test_user_item, train_user_item, int(num_test_use
      FinalRecommendation=Predict(train_user_item, ask2ratings, similarity, k)

      ##temp_rmse=temp_rmse + CalcRMSE(FinalRecommendation,test_user_item)/8
      hit_rate=hit_rate+HitRate(test_user_item, predictions, num_movies2rate)/num_iter
    print("Group size: {0}, No of Movies to rate: {1} Hit-Rate: {2}".format(group_size
    #basic_result[i,j]=temp_rmse
    hr0[i,j]=hit_rate

    j=j+1
  i=i+1
```

```
Group size: 10, No of Movies to rate: 15 Hit-Rate: 0.3289829512051734
Group size: 10, No of Movies to rate: 30 Hit-Rate: 0.29212228101116994
Group size: 10, No of Movies to rate: 45 Hit-Rate: 0.2748971193415638
Group size: 10, No of Movies to rate: 60 Hit-Rate: 0.26481481481481484
Group size: 50, No of Movies to rate: 15 Hit-Rate: 0.32475014697236915
Group size: 50, No of Movies to rate: 30 Hit-Rate: 0.28447971781305115
Group size: 50, No of Movies to rate: 45 Hit-Rate: 0.2630217519106408
Group size: 50, No of Movies to rate: 60 Hit-Rate: 0.26934156378600815
Group size: 100, No of Movies to rate: 15 Hit-Rate: 0.34838330393885947
Group size: 100, No of Movies to rate: 30 Hit-Rate: 0.30123456790123454
Group size: 100, No of Movies to rate: 45 Hit-Rate: 0.2908485204781501
Group size: 100, No of Movies to rate: 60 Hit-Rate: 0.26099353321575547
Group size: 150, No of Movies to rate: 15 Hit-Rate: 0.34591416813639037
Group size: 150, No of Movies to rate: 30 Hit-Rate: 0.3145796590241035
Group size: 150, No of Movies to rate: 45 Hit-Rate: 0.2921026846952773
Group size: 150, No of Movies to rate: 60 Hit-Rate: 0.2631981187536743
```

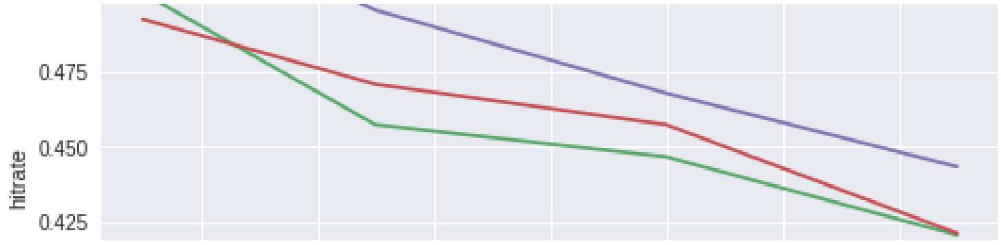
```
##### based on popularity
#### reading the dataset #######
Dataset=readUserDataset("u.data")

num_users=NumUsers(Dataset)
num_movies=NumMovies(Dataset)

#### creating the user item matrix ####
user_item_mat=CreateUserItemMat(num_users,num_movies,Dataset)
user_item_mat_copy=user_item_mat

#### most popular movie calculation ####
locations=np.where(user_item_mat_copy>0)
user_item_mat_copy[locations]=1
final_sum=user_item_mat_copy.sum(axis=1)

#### Splitting the dataset into 80:20 training vs test dataset ####
num_test_users=round(0.2*num_users)
num_train_users=num_users-num_test_users

k=[10,50,100,150] ###### Neighborhood size
movie_set_size=[15,30,45,60] ##### Number of Movies shown to the user
#### to store the result #####

hr2=np.zeros((4,4))

i=0
for group_size in k:
  j=0
  for num_movies2rate in movie_set_size:
```

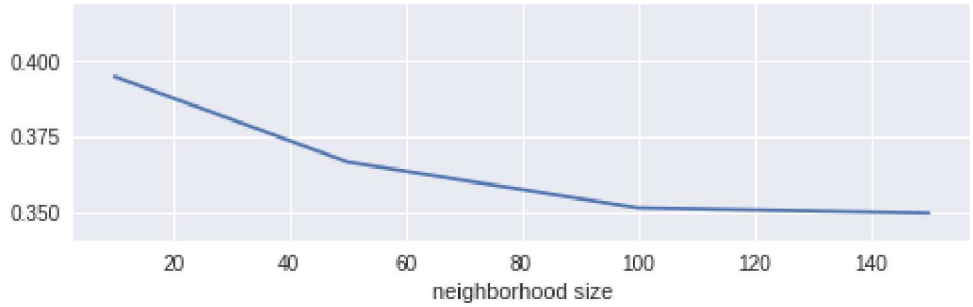
```
    temp_rmse=0
    hit_rate=0
    for iter in range(0,num_iter):

      seperator=random.sample(range(0, num_users), num_users)

      #### grouping the users into test and training sets
      test_users=array(seperator[:num_test_users])
      train_users=array(seperator[num_test_users:])

      #### creating test and train user item matrix ####
      test_user_item=user_item_mat[test_users,:]
      train_user_item=user_item_mat[train_users,:]

      k=group_size #users considered.
      movies2rate=num_movies2rate

      #### Asked to rate based rating from user based on popularity ####
      ask2ratings=np.zeros(test_user_item.shape)
      topK=[np.argsort(final_sum)[:-num_movies2rate-1:-1]]


      for l in range(0,test_user_item.shape[0]):
        for m in topK:
          ask2ratings[l,m]=test_user_item[l,m]


      #### Collaborative filtering ####
      similarity=CalculateSimilarity(ask2ratings, train_user_item, int(num_test_users)
      FinalRecommendation=Predict(train_user_item, ask2ratings, similarity, k)

      ##temp_rmse=temp_rmse + CalcRMSE(FinalRecommendation,test_user_item)/8
      hit_rate=hit_rate+HitRate(test_user_item, FinalRecommendation, num_movies2rate)/

    print("Group size: {0}, No of Movies to rate: {1} Hit-Rate: {2}".format(group_size
    #basic_result[i,j]=temp_rmse
    hr2[i,j]=hit_rate

    j=j+1
  i=i+1
```

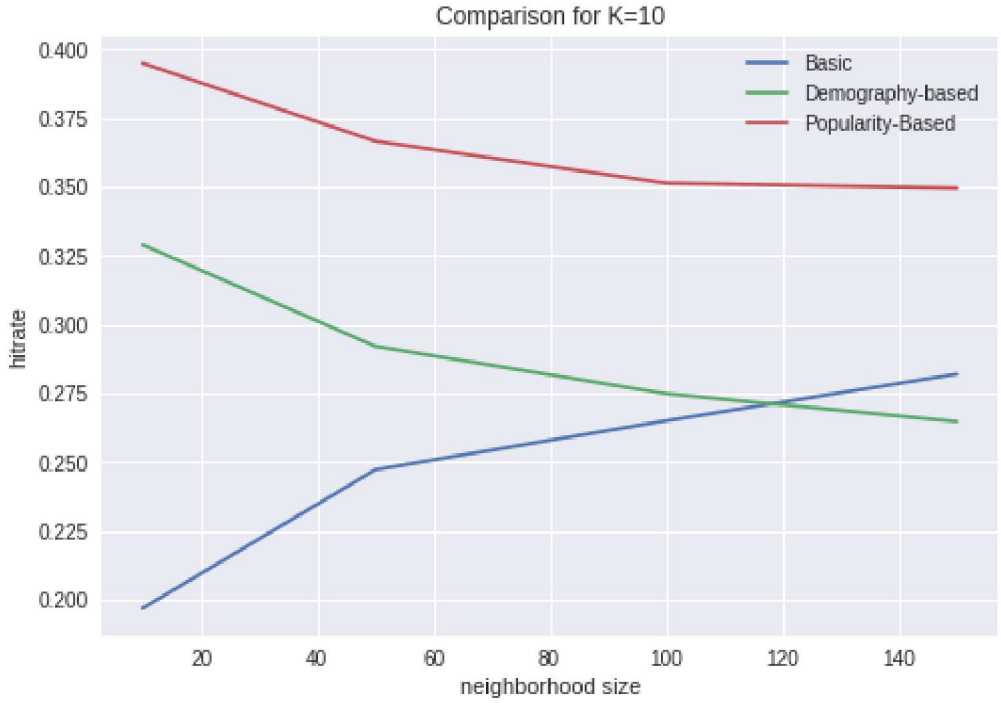
```
/usr/local/lib/python3.6/dist-packages/scipy/stats/stats.py:3010: RuntimeWarning: i
  r = r_num / r_den
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:60: RuntimeWarning: in
Group size: 10, No of Movies to rate: 15 Hit-Rate: 0.3949441504997061
Group size: 10, No of Movies to rate: 30 Hit-Rate: 0.3666666666666667
Group size: 10, No of Movies to rate: 45 Hit-Rate: 0.35140113658632177
Group size: 10, No of Movies to rate: 60 Hit-Rate: 0.349647266313933
```

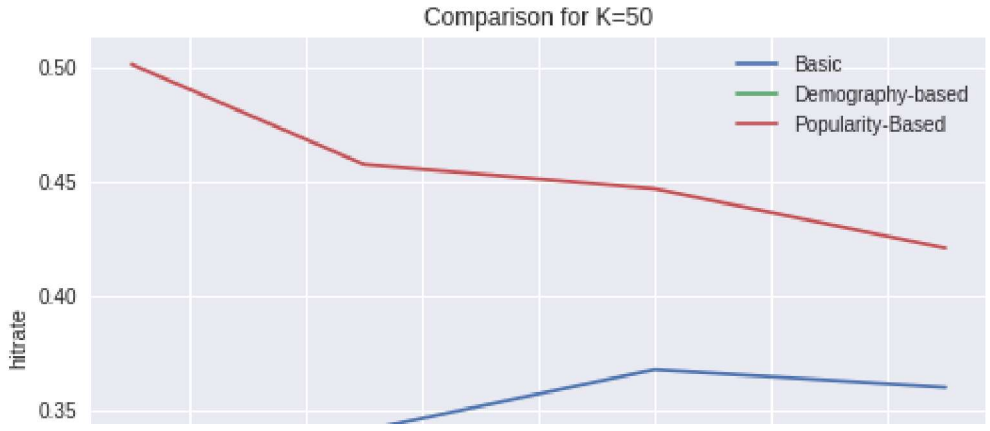
```
Group size: 50, No of Movies to rate: 15 Hit-Rate: 0.5009994121105232
```

```
1
2
3
4 K=np.array([10, 50, 100, 150])
5 name=["Basic","Demography-based","Popularity-Based"]
6 plot_comparison(hr1,hr0,hr2,K,name)
7
8
9
```




0.400
0.375
0.350
20
40
60
80
100
120
140
neighborhood size


Comparison for K=10
Basic
Demography-based
Popularity-Based
hitrate
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
20
40
60
80
100
120
140
neighborhood size


Comparison for K=50
Basic
Demography-based
Popularity-Based
hitrate
0.50
0.45
0.40
0.35