Netflix Prize Data Recommender

DSC478 Final Application Report

Winter 2021

Robert Kaszubski

Eric Vistnes

Xiaojing Shen

I. Introduction

This final report serves to outline our processes and the techniques we used in building our movie recommendation system. Movies have been one of the most popular forms of entertainment for more than a century. Millions journey to movie theaters worldwide to see the latest blockbusters, dramas, comedies, musicals, westerns, you name it, there's a movie about it. However, since the new millennium, there has been a growing demand for the capability of watching and enjoying movies from home. Netflix is currently the leading provider of movie and television streaming in the world. While they may be a streaming giant now, in the era before high speed internet and in-home streaming, they were known for their by-mail home delivery service. Customers could rent their desired films online and have the DVD conveniently shipped to their home. Once the customer had finished watching their films, they would mail them back and were asked to rate each film based on how much they enjoyed it. Netflix captured this data to utilize in their recommendation system in order to suggest movies to their customers that they believed they would enjoy. In 2009, Netflix opened a million dollar competition to create a better collaborative filtering algorithm utilizing the data they've provided. We have used that same dataset to devise our own recommendation algorithms utilizing techniques such as clustering, nearest neighbor and standard value decomposition.

II. Data Overview

The Netflix Prize dataset can be currently found and downloaded on Kaggle. The primary dataset is stored on four text files labeled combineddata(1,2,3,4) . This data list a MovieID (ranging from 1 to 17770 sequentially), followed by a list containing each CustomerID (ranging from 1 to 2,649,429, with gaps, in total there are 480,189 customers) that rated that movie along with their rating on a scale from 1 to 5 and the date when the film was rated. All together there are 8,532,958,530 ratings. The dataset also contains a separate csv file that contains the movie titles corresponding to each MovieID along with the release year.

Our first step was to combine the four text files into one cohesive dataset that we stored in a csv. This was accomplished using a regular expression that upon reading the original text files line by line was able to match when a new MovieID was presented and know the following lines were ratings for that movie. Upon storing and reopening our newly created csv, we realized the size of the dataset would pose memory and storage issues when running our desired functions on it. We took steps to reduce the memory usage of the dataset by altering the data types of our variables.

```
movie.memory_usage()
```

```
Index              128
MovieID       803844056
CustomerID    803844056
Rating        803844056
Date          803844056
dtype: int64
```

```
movie.dtypes
```

```
MovieID        int64
CustomerID     int64
Rating         int64
Date          object
dtype: object
```

Making these changes reduced the dataset from nearly 3 gigabytes to a more manageable sub one gigabyte dataset.

```
movie.memory_usage()
```

```
Index              128
MovieID       200961014
CustomerID    401922028
Rating        100480507
Date          201060390
dtype: int64
```

```
movie.dtypes
```

```
MovieID          int16
CustomerID       int32
Rating            int8
Date          category
dtype: object
```

The next challenge came with pivoting our dataset to create a Customer by Movie matrix storing all of the different rating values. Due to the size of the resulting matrix, we resorted to splitting

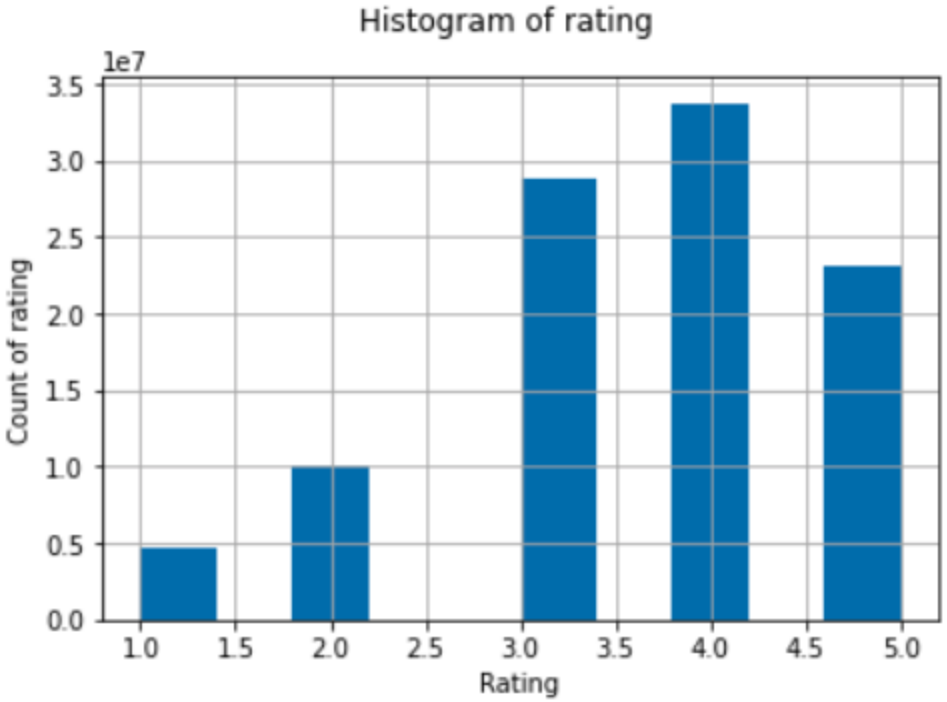
the data into chunks, creating eight separate pivot tables with those chunks and then appending the data together to one DataFrame. This new dataframe contained nearly 66 billion values, largely due to the high amount of missing values of the movies that customers have not seen yet and we would be predicting and recommending. This dataframe and later our models would be stored using pickle.

III. Feature Exploration/Analysis

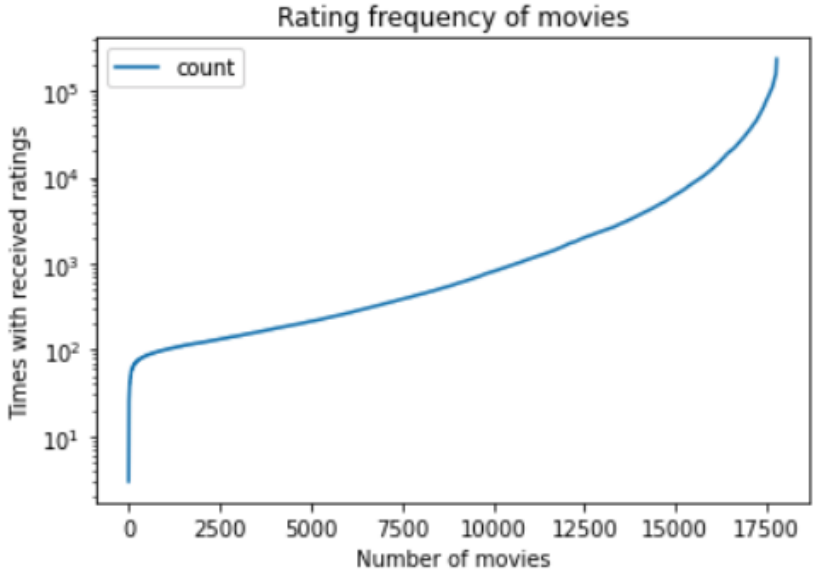
```
movie['Rating'].describe()
```

```
count    1.004805e+08
mean     3.604290e+00
std      1.085219e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: Rating, dtype: float64
```

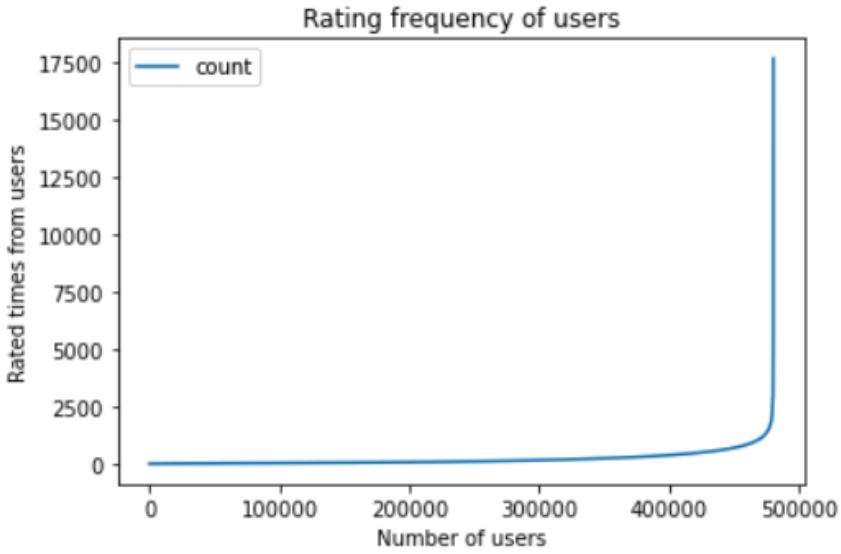
There are four variables presented in our raw database. Among those, CustomerID, Movie ID are nominal features. And date is not used in our model. Only rating is reviewed and analyzed. Above table states how the rating overall behaves and distributes. In total, more than 100 million ratings have been included and they fall in between 1 to 5, with mean and median at 3.6 and 4 respectively.



Here is a histogram map of rating. We could see the trend that the majority of received ratings are higher than 3. 4 points is the most popular selection and very few results are presented with score 1 or 2.



This is the line graph which describes the relationship between numbers of movies vs. numbers received rates. It is quite noticeable that most movies have received a fair amount of reviews- at least more than 100 comments.

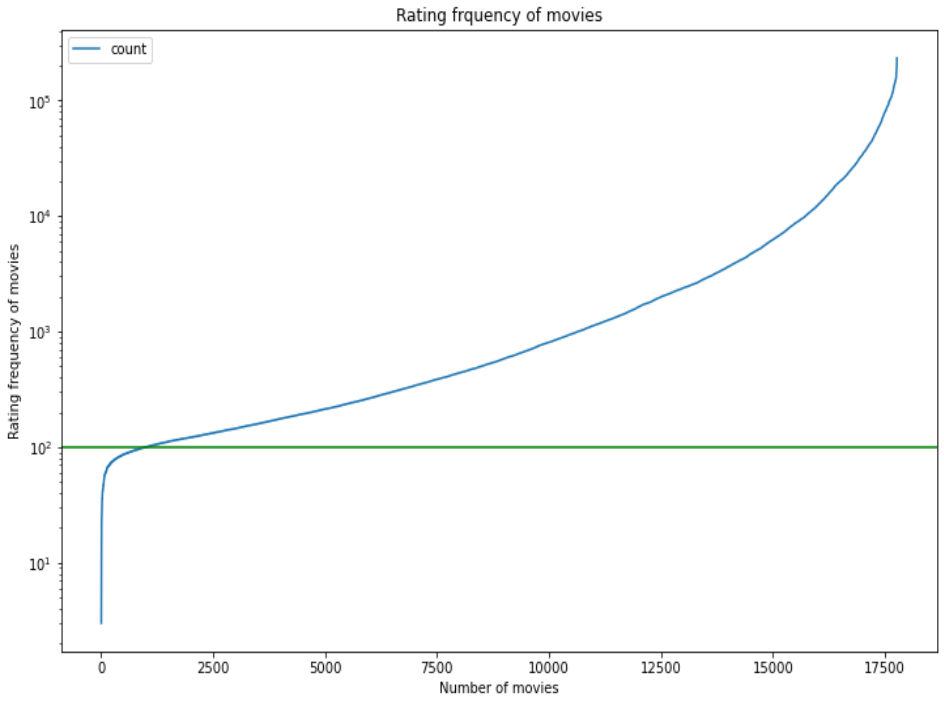


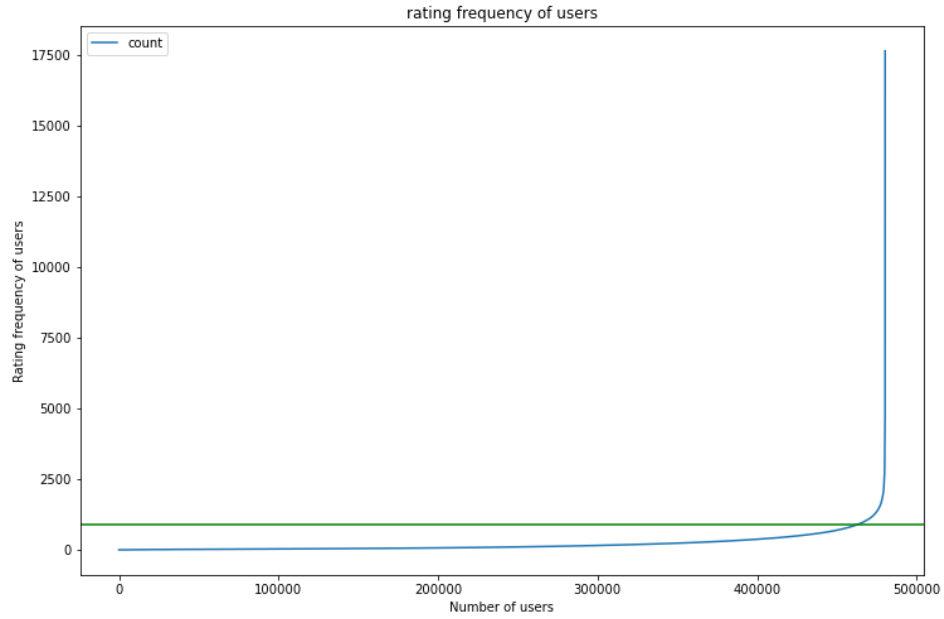
This is the line graph of rating frequency of users. It is interesting to see that most users (more than 400000) are not used to share feedback- very few leave with rates. But there are still a few-much smaller group users who are willing to share their perspective. They do seem to enjoy it very much, with ratings more than 1000 times.

IV. Data Preprocessing and Transformation

The original dataset from Kaggle provided us with a DataFrame with a shape of (100480507, 4). The amount of data is far too large to properly manipulate and analyze on our personal computers, and so we found wants to minimize the data by cutting out unnecessary noise, reducing data usage, and narrowing the dataset into Customers and Movies with the highest number of ratings.

Based on attempts at testing our algorithms on varying data sizes, we found an upper limit on the amount of data that could be processed at a time. That limit is around 20% of the total dataset, leaving us a dataset with shape (20000000, 3). In a dataset of that size, we would have far fewer data points than we would have from a dataset of just the frequent movies and customers. Therefore, in order to include the most data possible in our calculations, we can pass in only the most frequent CustomerIDs and MovieIDs, and cut out CustomerIDs and MovieIDs that don't contribute a meaningful amount to the calculation of our algorithms.





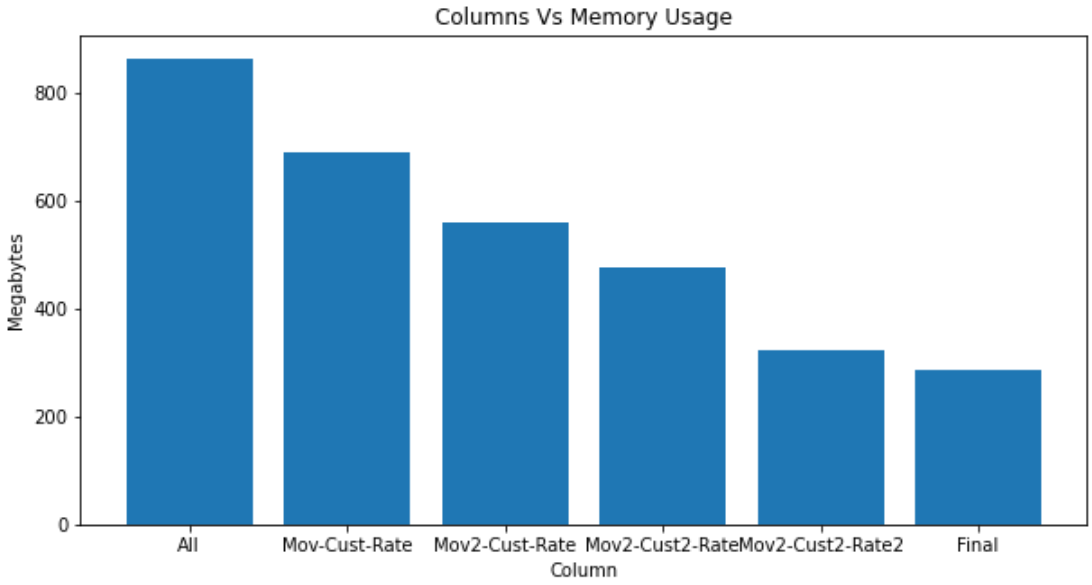
The graphs above show the thresholds that we use for selecting customers and movies: Customers who have rated 900 times or more and Movies that have been rated 100 times or more. We are left with a dataset of shape (22605955, 4) after this threshold is applied.

Our new dataset can be reduced further- we do not use the Dates feature in our algorithms, so we can remove a whole column, and with it, a full quarter of the dataset. The rest of the columns are stored in int64 format, but each can be reduced. We can reduce the data type of CustomerID from int64 to int32, of MovieID from int64 to int16, and of Rating from int64 to int8. The max values of each column does not exceed the maximum value of the data types, so no data is lost in this transformation. This is shown in the values below.

```
print("Max value of CustomerID: ", movie['CustomerID'].max())
print("Max value of int32: 2,147,483,647")
print("Max value of MovieID: ", movie['MovieID'].max())
print("Max value of int16: 32,767")
print("Max value of Rating: ", movie['Rating'].max())
print("Max value of int8: 127")
```

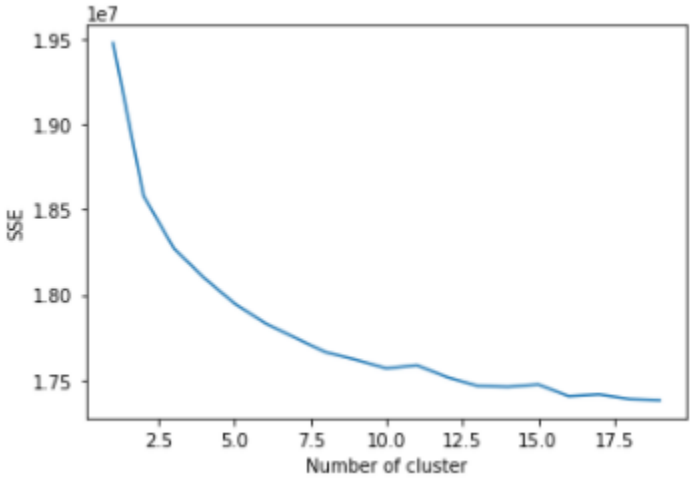
```
Max value of CustomerID:  2649429
Max value of int32: 2,147,483,647
Max value of MovieID:  17770
Max value of int16: 32,767
Max value of Rating:  5
Max value of int8: 127
```

We can see a marked decrease in the size of the data set as we implement these changes. The dataset size is reduced to less than half the size even before we limit the data set to 20,000,000 data points. The reduction to an artificial limit of 20 million only reduces the dataset by approximately 11.5% further. This reduced data is the final data set that each of our models uses. Some may be further manipulated to match the model and the model capabilities, but each takes this as the initial data.



V. Clustering

Our first approach was to cluster our now reduced dataset in a way akin to document term clustering. To accomplish this we first had to create a customer by movie matrix with rating values. This proved challenging given the scale of the initial dataset. Our first attempt was only capable of running around 20% of the data due to memory limitations however with the reduced dataset we managed to pivot and run the full reduced dataset. The data was pivoted so that each row represented a customer and each column represented a movie. The next issue arose from the fact that no customer could have possibly viewed and rated every single movie in the dataset. This led to a majority of our data being compromised of missing values and we could not run clustering on our data. We took two approaches to eliminating missing values. The first was to replace all missing values with zeros. This proved to be reasonably effective as it allowed the clustering algorithm to be trained and then tested passing in our user's input. Using Sklearn kmeans clustering, we returned the centroids for each cluster. Each centroid has a value corresponding to each feature (movie) in the dataset. We were able to sort these in descending order and print out the top films for the cluster the user belonged to as our recommendations. Our second approach to handling missing values involved calculating an average for each film.



This rating was calculated by finding the sum of all ratings for a movie, adding it to the average of all the movies multiplied by 25 then divided by 25 + the total number of ratings of that film. This however did not prove to be very successful as just from a glance we saw the same films near the top of almost every cluster. We also explored various k values and plotted the results of our sum of square error. Using the elbow method we determined that 10 would be the best k value, although this wasn't too clear so we tested values ranging from 5 to 10. Once a model was generated using the dataset, we could pass in a new data row based on the user's inputs for the movies we initially prompted them to rate. This would cluster them and based on whichever cluster they ended up in, we would recommend the top movies from that cluster which they have not yet seen.

VI. K-Nearest-Neighbors

The K-Nearest-Neighbors method is also applied in this project. Two recommenders are introduced which are item-based, and user-based recommender. First of all, the reduced dataset was pivoted into a table, which describes ratings received from customers on each movie.

In addition, a cross validation function was designed to skip a certain percentage of rated score and use the remaining rating to find either k similar movies in item-based recommender model or k similar users in user-based recommender. Then we would refer to ratings from neighbors to predict a score for the skipped items. Later, mean absolute error between the predicted and true value would be calculated. To figure out the best k value for the model, three candidates– 10, 30, 50 were adopted and executed. The final result turns out to be- in the item-based recommender, minimum MAE happens while K is equal to 10. At the same time, k equal to 30 has been decided as the best parameter in user-based recommender.

By applying the most effective parameters to get the best model, we could then proceed to new users with their limited ratings to provide systematic recommendations. In item-based recommender, in order to figure out the top 3 picked movies, the ratings from new users were sorted. Next, the model will identify the k nearest-neighbors of the top 3 movies. The next step is to use ratings and the distance between the top 3 movies to decide the ratio of their neighbors. Movies with top 5 ratios then will be presented to the new customer. Similar to the previously discussed item-based system, here in the user-based engine, it would learn ratings from new users to locate users with similar taste from the dataset, then selecting their favorite movies from neighbors to recommend to new users.

VII. Singular Value Decomposition

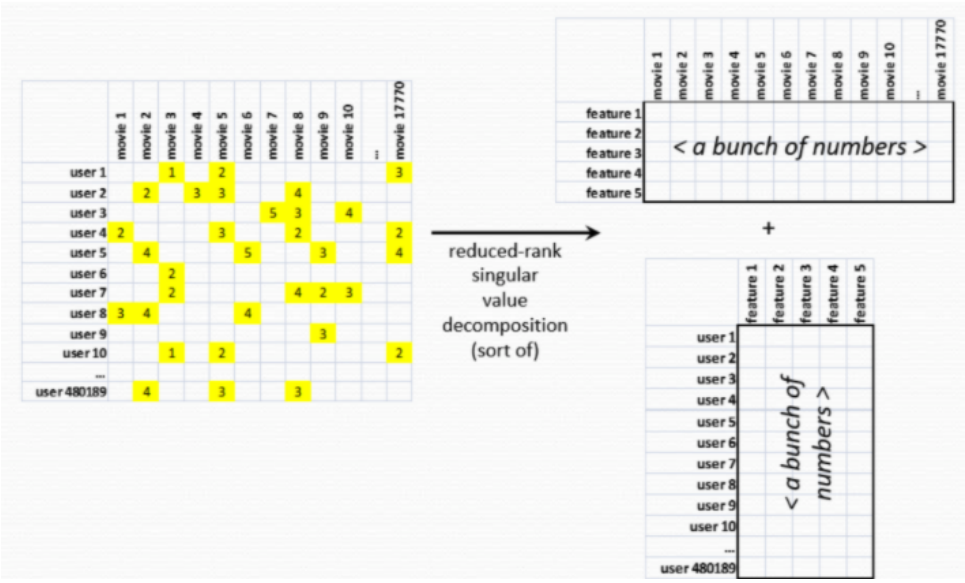
The Singular Value Decomposition model is a method that uses the SVD of the dataset, alongside averages of the data set, and movie and customer bias, to predict a rating from a Customer on a specific Movie. The SVD algorithm we use is an algorithm that was popularized during the Netflix Prize Competition. The prediction for each rating can be calculated via the formula below:

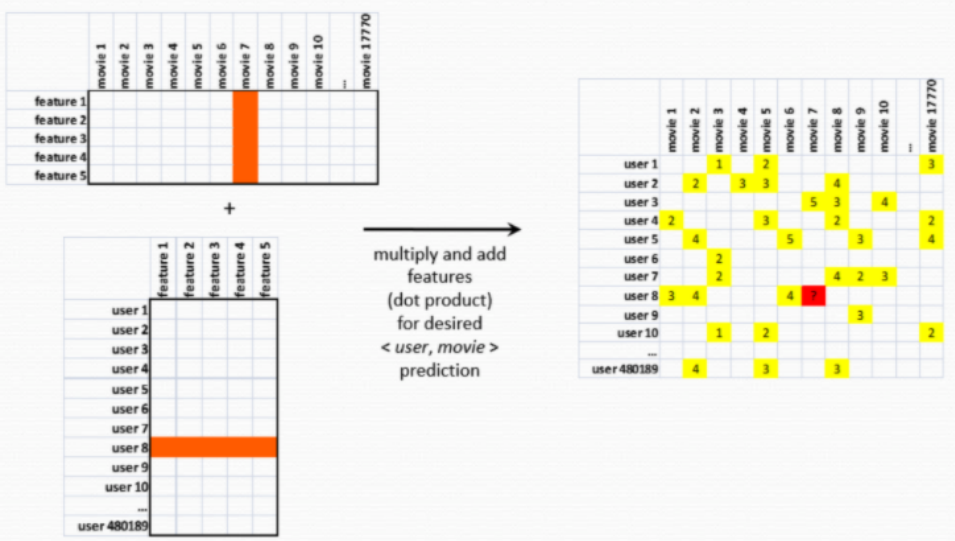
$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

In this formula, μ is the mean. Bu is the bias for a user, and Bi is the bias for an item. These indicate the variation from the global mean for the mean values of the user and item. Final, qi and pu are the factors making up the SVD of the users and items.

In order to create the Singular Value Decomposition of the data, we need to determine how many factors should be included in the final decomposition. In order to determine the optimal number of factors, we first run GridSearchCV on a list of factors and take the factor that returns the best RMSE score. The Grid Search is run with a 3-fold cross validation with RMSE accuracy.

Now that we have the number of factors to use, we can finally implement dimensionality reduction on the data in a manner that lets us run the dataset. The SVD model will handle the dimensionality reduction by matrix factorization. The SVD reveals the latent features in the data set that we can use to predict specific values, while also reducing the number of features in use by only taking the most significant features.





If we only use the SVD predictions from our data, we would be able to already approximate a user's rating of a movie. The features that are remaining after SVD are the features that capture most of the user's interest- positive or negative, in the different movies. That gives us the formula below, which is the beginning of the formula that we want to use to predict ratings. We can then use the mean and biases of customers and movies to finalize the formula to the one above.

$$\hat{r}_{ui} = q_i^T p_u$$

By using this method and algorithm, we were able to reach an RMSE of 0.829 in our test data. However, this test data was limited as we described earlier, which leads to bias within the data due to correlation between users who are likely to rate as frequently as we require. Nonetheless, this gives our model an extremely high accuracy for the Netflix Prize Data for users who input a large amount of ratings. When tested on the full data set of users, using only the most popular movies that we trained on, the RMS dropped to 0.99.

However, the testing we ran on the full data set of users may not have provided accurate results as the model may not be able to accurately predict ratings for users that were not at all in the training data. For those users, they would receive a rating that does not take into account their personal preference or bias.

Finally, we created a separate recommender system for the SVD model that rates all the Movies for a given user, and then sorts the ranking to provide the user a dynamically sized list of

recommended movies. The model can run efficiently on such a small dataset as one CustomerID and all the MovieIDs. We create lists that link to each other to show the rating for each MovieID, and then we can print the Movie titles of the top movies. Each Movie title is also printed with the SVD model's predicted rating.

VIII. Application

The application is written entirely in python and must be run from a shell prompt. For all of our testing we have used Anacondas PowerShell Prompt by simply navigating to the app directory and running python main.py. Most of the code can be found in our main Jupyter Notebook with clustering, KNN, and SVD is slightly tweaked with each placed in it's own separate python file which are then called by main.py. The other files that were created solely for the application are attached in the appendix below. Main.py greats the user and asks for the user's name. We keep all users stored locally in our users folder. If this is the user's first time interacting with the application, the application runs initial.py and the user is prompted to either rate movies from a pool of 20 of the most popular or rate movies from a pool of 20 random movies. This number can be adjusted to provide a larger pool of movies and get more input data from the user but we felt like 20 was an adequate amount. The user rates the movies from 1 to 5 or can press enter if they have not seen the movie or do not wish to rate it. Once the user completes the initial rating process, they gain access to our full recommender system. That user created data is returned to main.py from initial.py and stored as a list of [[MovieID, Rating]]. The user can choose to receive movie recommendation via four different methods: clustering, k-nearest-neighbor name-based, k-nearest-neighbor item-based, and standard value decomposition. Each of these takes the user created ratings as a parameter. These do take a while to run as we are essentially adding a new user to our already massive dataset. Each of these methods will return recommendations based on the ratings the user has provided. They all return a list of 10 movies. The user is then prompted to rate any of the 10 recommendations that they may have already seen by calling addrating.py. This function returns a list in the same format as the original user created ratings data. This data is then appended to the user created ratings data. From there all subsequent runs of any of the algorithms will use the new users rating data complete with every movie that user has ever rated. Thus the more the user runs our system and rates movies, the better the recommendations will become. Finally upon exiting our system, the user created rating data will be stored in a pickle file. Any time they return to our

application and input their name, that data will be opened and used rather than the user having to start from scratch and re-rate all of the movies they have seen.

IX. Conclusion

In conclusion, by using a shortened data set of the most active customer and popular movies, we were able to create three different recommender systems that are able to all be utilized in an application. We achieved different levels of accuracy for each, but each serves a different purpose in a recommender system. Clustering and K-Nearest-Neighbors can recommend not only Movies, but other Customers that are similar to each other. KNN is the most optimal model for finding the most similar users, but does not have the capabilities to label a Customer as a certain type of user. SVD can be used in addition to both Clustering and KNN by predicting a specific rating for each Customer and Movie that differs even within clusters.

Each of these forms of information can be provided to a user who is looking for recommendations through our system. Additionally, it would be feasible to move forward by combining the different models together for a more accurate result. The SVD model of prediction incorporates user and movie bias, and both KNN and Clustering Analysis would provide even more accurate biases for the formula by specifying the Cluster or KNN bias. In future work that any of us may do on the Netflix Prize Data, this would be an efficient manner to improve our already existing algorithms.

X. Appendix

README

Must use a shell prompt to run. Won't work on Windows Command prompt.
We recommend using Anaconda Powershell Prompt which was installed with the Anaconda package and Jupyter Notebook.

Before running our files input this in the terminal to install sklearn suprise:
conda install -c conda-forge scikit-surprise

Our application also requires pandas, numpy, sklearn, and pickle
All of which should be included with Anaconda.

Running the app:
Download and unzip.
Navigate to the app folder in Anaconda Powershell Prompt using cd "yourpathhere"/app
Now type: python main.py
to launch the application

Please follow along with the onscreen prompts!
Enjoy!

```
import initial
import addrating



import numpy as np
import pandas as pd

#ask user for their name which we use as an id of sorts to generate files storing their ratings
import pickle
print("Welcome to our movie recommender system!")
name = str(input("Please enter your name: "))

#user data is stored in the users folder in a pkl file
import os.path
save_path = 'users/'
file_name = name+".pkl"
completeName = os.path.join(save_path, file_name)

#if the file exists, open the ratings from it
if os.path.isfile(completeName):
```

```
    with open(completeName, 'rb') as f:
        user_data = pickle.load(f)
#else we know its a new user, so run the initializing to capture their first ratings
else:
    user_data = initial.run_file(20)

#print(user_data)
#loop to keep the application running
cont = True
while cont:
    print("")
    print("Our system supports Clustering, Knn User, Knn Item, and SVD, each offering different recommendations!")
    print("Which would you like to run?")
    print("a Clustering")
    print("b Knn User-Based")
    print("c Knn Item-Based")
    print("d SVD")
    #users can pick which recommendation algorithm/system they want to use
    choice = str(input("Please choose option a, b, c, or d "))
    if choice == "a" or choice == "A":
        print("Clustering...")
        print("Please Wait")
        import cluster
        #recs returns the top movies recommended from that cluster
        recs = cluster.cluster(user_data)
        #print(recs)
        #addmore allows the user to rate movies that were recommended thus strengthning their predictions
        addmore = str(input("Would you like to rate any of the recommendations? y/n"))
        if addmore == "Y" or addmore == "y":
            new = addrating.run_file(recs)
            user_data += new
            #print(user_data)
    #repeat code except for Knn-user
    elif choice == "b" or choice == "B":
        print("KNN user-based...")
        print("Please Wait")
        import knn_u
        recs = knn_u.classify(user_data)
        addmore = str(input("Would you like to rate any of the recommendations? y/n"))
        if addmore == "Y" or addmore == "y":
            new = addrating.run_file(recs)
            user_data += new
            #print(user_data)
    elif choice == "c" or choice == "C":
        print("KNN item-based...")
        print("Please Wait")
        import knn_i
        recs = knn_i.classify(user_data)
        addmore = str(input("Would you like to rate any of the recommendations? y/n"))
        if addmore == "Y" or addmore == "y":
            new = addrating.run_file(recs)
            user_data += new
            #print(user_data)
    elif choice == "d" or choice == "D":
```

```
        print("SVD...")
        print("Please Wait")
        import svd
        recs = svd.run_svd(user_data)
        addmore = str(input("Would you like to rate any of the recommendations? y/n"))
        if addmore == "Y" or addmore == "y":
            new = addrating.run_file(recs)
            user_data += new
            #print(user_data)
    #ask the user if they want to exit
    choice_cont = str(input("Run a different algorithm? y/n"))
    if choice_cont == "y" or choice_cont == "Y":
        continue
    else:
        cont = False

#save their rating data into their pickle file
with open(completeName, 'wb') as f:
    pickle.dump(user_data, f)




import pandas as pd
import numpy as np

#open reduced dataset and movie_titles

object = pd.read_pickle('data/cleanedMovie.pkl')
movies = pd.DataFrame(object)

terms = pd.read_csv('data/movie_titles.txt', sep='\t', encoding = "ISO-8859-1", header=None, index_col=0)
terms = terms.iloc[:,:2]
terms = terms.iloc[:,1:]

#function to convert movieID to title
def toTitle(MovieID):
    return terms.loc[ MovieID , : ][2]

#captures ratings of n movies when a new user joins app
def initial_rating(n, movielist, t):
    print("")

    ratings = []

    #if the user chooses to rate random movies
    if t == "r":
        print("These are",n, "random movies on our system:")
        order = np.random.randint(len(movielist), size=n)
    #if the user chooses to rate the most popular movies
    if t == "p":
        print("These are the top",n, "most watched movies on our system:")
        order = np.arange(n)
    print("Please rate at least 5 movies on a scale of 1 to 5 (1,2,3,4,5), you haven't seen it just hit Enter.")
```

```
    print("We can then start providing you recommendations based on your initial input!")
    print("")
    #iterates through n movies
    for mov in order:
        ID = movielist.index[mov]
        title = toTitle(ID)
        print("You are rating", title)
        #ensure proper input validation
        try:
            while True:
                b = int(input("Rate from 1 to 5: "))
                if b < 1 or b > 5:
                    print("Sorry, your response must be on a scale of 1 to 5.")
                    continue
                else:
                    break
            print("You rated", title, "as a", b)
            ratings.append([ID,b])
        #if the user presses enter of any non int values it skips the movie
        except:
            print("You skipped", title)
            print("")
            continue
        print("")


    return ratings

#function that asks how many movies to rate (n), returns a list of user created ratings back to main.py
def run_file(n):
    #print("Welcome to our Movie Recommender System!")
    print("We ask you to first rate some movies.")
    print("a Rate the", n,"most popular movies?")
    print("b Rate", n,"random movies?")
    choice = str(input("Please choose option a or b"))
    if choice == "a" or choice == "A":
        popular_movies = pd.DataFrame(movies["MovieID"].value_counts())
        return initial_rating(n, popular_movies, "p")
    else:
        popular_movies = pd.DataFrame(movies["MovieID"].value_counts())
        return initial_rating(n,popular_movies, "r")



import pandas as pd
import numpy as np


#function that allows the user to rate movies found on our generated recommendations
#returns a list of MovieID, Rating that can then be appended to the users previously created ratings
def run_file(movies):
    print("Please rate on a scale of 1 to 5 (1,2,3,4,5), you haven't seen it just hit Enter.")
    print("We can then update your recommendations based on your newly added ratings!")
    cont = True
```

```
    out = []
    rated = []
    while cont:
        a = int(input("Enter index of movie you want to rate from the recommendations above: "))
        if a not in rated:
            print("You are rating:", movies[a-1][1])
            b = int(input("Rate from 1 to 5: "))
            print("You rated:", movies[a-1][1], "as a", b)
            out.append([movies[a-1][0], b])
            rated.append(a)
        else:
            print("Already rated, ", movies[a][1])
        b = str(input("Continue rating? y/n"))
        if b == "y" or b =="Y":
            continue
        else:
            cont = False
    return out



import pickle
from tqdm import tqdm
import numpy as np
import pandas as pd
import os
import pathlib
from surprise import Reader, Dataset
from surprise import SVD
from surprise import SVDpp
from surprise.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

movie_title = pd.read_csv("data/movie_titles.csv", encoding='unicode_escape', usecols=[2], header=None)
movie_title.columns = ['title']
movie_title.head(15)



class SVDPredictor:
    error_table = pd.DataFrame(columns = ["Model", "Train_RMSE", "Test_RMSE"])

    #Class takes in final.csv as a whole as a DataFrame
    def __init__(self, data, titles):
        self.movie = data
        self.titles = titles
        self._createAlgorithmFromData()

    def _createAlgorithmFromData(self):
        #check if algo and trainset/train_data files are already created
        self._splitMovie()
        self._createTrainSet()
        self._run_surprise()
```

```
    def recommendFor(self, customerID, count, user_ratings):
        preds = []
        ids = []
        watched = []
        for movie in user_ratings:
            watched.append([movie][0][0])
        for mov in self.movie.MovieID.unique().tolist():
            preds.append(self.predict(customerID, mov).est)
            ids.append(mov)

        movieAndRating = {}
        copyPreds = preds[:]
        while count > 0:
            index = copyPreds.index(max(copyPreds))
            maxPred = max(copyPreds)
            mov = ids[index]
            if mov not in watched:
                title = movie_title.iloc[mov-1:mov]['title'][mov-1]
                movieAndRating[mov] = maxPred
            copyPreds.pop(index)
            count -= 1
        return movieAndRating

    def predict(self, userID, movieID):
        #use algo to predict rating. Return predicted rating
        return self.algo.predict(userID, movieID)

    def _splitMovie(self):
        self.movie = self.movie.iloc[:1500000]

    def _createTrainSet(self):
        reader = Reader(rating_scale=(1,5))
        movieInput = pd.DataFrame()
        movieInput['CustomerID'] = self.movie['CustomerID']
        movieInput['MovieID'] = self.movie['MovieID']
        movieInput['Rating'] = self.movie['Rating']

        self.train_data = Dataset.load_from_df(movieInput, reader)
        self.trainset = self.train_data.build_full_trainset()
        #write to a file

    def _reduceDataSize(self):
        #self.movie['Date'] = self.movie['Date'].astype('category')
        self.movie['MovieID'] = self.movie['MovieID'].astype('int16')
        self.movie['CustomerID'] = self.movie['CustomerID'].astype('int32')
        self.movie['Rating'] = self.movie['Rating'].astype('int8')

    def _run_surprise(self):
        self.algo = SVD(n_factors = 5, biased=True, verbose=False)
        self.algo = self.algo.fit(self.trainset)


movieID = pd.read_pickle("data/cleanedMovie.pkl")
```

```
def recommendFor(customerID, model):
    predictions = []
    ids = []
    for mov in movieID.MovieID.unique().tolist():
        predictions.append(model.predict(customerID, mov).est)
        ids.append(mov)
    return predictions

def recommendedMovies(count, preds, movs):
    movieAndRating = {}
    copyPreds = preds[:]
    for i in range(count):
        index = copyPreds.index(max(copyPreds))
        maxPred = max(copyPreds)
        mov = movs[index]
        title = movie_title.iloc[mov-1:mov]['title'][mov-1]
        movieAndRating[title] = maxPred
        copyPreds.pop(index)
    return movieAndRating

#function required to run svd by main.py
def run_svd(user_ratings):
    movie = pd.read_pickle("data/cleanedMovie.pkl")
    #print(movie.shape)
    for rating in user_ratings:
        movie = movie.append(pd.DataFrame([[rating[0],2649430,rating[1]]], columns=movie.columns))
    #print(movie.shape)
    #print(movie.loc[movie['CustomerID'] == 2649430])
    svd = SVDPredictor(movie, movie_title)
    out= svd.recommendFor(2649430, 10, user_ratings)
    print("")
    print("Recommended movies:")
    movies_returned = 0
    output = []
    for key, value in out.items():
        print(movies_returned+1, movie_title.iloc[key-1]['title'])
        movies_returned += 1
        output.append([key,movie_title.iloc[key-1]['title']])
    return output
```

```
# In[1]
import numpy as np
import pandas as pd
import pickle
from sklearn.neighbors import NearestNeighbors
from math import sqrt
from sklearn.metrics import mean_squared_error
import math
```

```
df = pd.read_pickle('data/cleanedMovie.pkl')
user_movie_df = df.pivot(index='CustomerID',columns ='MovieID' ,values='Rating').fillna(0)


movie_names = list(user_movie_df.columns)
customer_names = list(user_movie_df.index.values)

knn = NearestNeighbors(metric='cosine',algorithm = 'brute', n_neighbors=10)

movie_title = pd.read_csv("data/movie_titles.csv", encoding='unicode_escape', usecols=[2], header=None)
movie_title.columns = ['title']
movie_title

recommendation = {}
#function to make recommendation, returns a list of recommendations
def make_recommendation(user_ratings, input_user,data,model,n_recommendation):
    model.fit(data)
    input_user_array = input_user.to_numpy()
    similar_users_list =
(model.kneighbors(input_user_array,n_neighbors=n_recommendation+1,return_distance=False)).tolist()
    for i in similar_users_list[0][1:]:
        for j in data.columns:
            ratingLst = []
            if int(input_user[j]) == 0 and data.iloc[i][j] > 3:
                if j not in recommendation:
                    ratingLst.append(data.iloc[i][j])
                    recommendation[j] = ratingLst
                else:
                    recommendation.get(j).append(data.iloc[i][j])
    print("Recommended Movies:")
    number = 0
    recs = []
    watched = []
    for movie in user_ratings:
        watched.append([movie][0][0])
    for k in sorted(recommendation, key=lambda k: len(recommendation[k]), reverse=True):
        if number < 10:
            if k-1 not in watched:
                res = movie_title.loc[k-1]['title']
                recs.append([k-1, res])
                print(number+1, res)
                number += 1 #recommend top 10 movies
    return recs

#convert user_ratings from list to dataframe
# create a empty dataframe



#run function from main.py

def classify(user_ratings):
```

```
    column_names = movie_names
    df_newUser = pd.DataFrame(columns = column_names)
    for i in user_ratings:
        df_newUser.at[0 , i[0]] = i[1]
    newUser_df = df_newUser.fillna(0)
    recs = make_recommendation(user_ratings, newUser_df, user_movie_df, knn, 10)
    return recs
```

```
# In[1]
import numpy as np
import pandas as pd
import pickle
from sklearn.neighbors import NearestNeighbors
from math import sqrt
from sklearn.metrics import mean_squared_error
import math

df = pd.read_pickle('data/cleanedMovie.pkl')
user_movie_df = df.pivot(index='CustomerID',columns ='MovieID' ,values='Rating')
movie_names = list(user_movie_df.columns)
customer_names = list(user_movie_df.index.values)

movie_users_df = df.pivot(index='MovieID',columns = 'CustomerID',values='Rating').fillna(0)

movie_title = pd.read_csv("data/movie_titles.csv", encoding='unicode_escape', usecols=[2], header=None)
movie_title.columns = ['title']
movie_title
```

```
#initiate knn algorithm
knn = NearestNeighbors(metric='cosine',algorithm = 'brute', n_neighbors=30)

#function to find similarity between movies
def find_similarity(topMovies,data,model,n_recommendation):
    dict_similarity = {}
    model.fit(data)
    for index in topMovies:
        array_index = data.iloc[index].to_numpy()
        neigh_dist, neigh_ind = model.kneighbors(array_index.reshape(1, 13141),n_neighbors=n_recommendation+1)
        neigh_dist_Lst = neigh_dist.tolist()[0][1:] #ignore itself
        neigh_ind_Lst = neigh_ind.tolist()[0][1:]
        for i in range(len(neigh_dist_Lst)):
            if neigh_ind_Lst[i] not in dict_similarity:
                dict_similarity[neigh_ind_Lst[i]] = (1-neigh_dist_Lst[i]) * topMovies.get(index)
            else:
                ratio = dict_similarity.get(neigh_ind_Lst[i])
                new_ratio = (ratio + (1-neigh_dist_Lst[i]) * topMovies.get(index))/2
                dict_similarity[neigh_ind_Lst[i]] = new_ratio
```

```
    sort_dict = dict(sorted(dict_similarity.items(), key=lambda item: item[1],reverse=True))
    return sort_dict

#function to make a recommendation - returns a list of movies recommended
def make_recommendation(user_ratings, topMovies, newUser_rating, data, model, n_recommendation):
    count = 0
    recs = []
    sort_dict = find_similarity(topMovies,data,model,n_recommendation)
    print("Recommended Movies:")
    for movie_ind in sort_dict:
        if count < 10:
            if int(newUser_rating[movie_ind]) == 0:
                res = movie_title.loc[movie_ind-1]['title']
                recs.append([movie_ind-1, res])
            print(count+1, res)
            count += 1
    return recs




#function called by main.py to run knn - item based
#returns a list of movies that were recommended
def classify(user_ratings):
    newUser_dict = {}
    column_names = movie_names
    df_newUser = pd.DataFrame(columns = column_names)
    for i in user_ratings:
        newUser_dict[i[0]] = i[1]
        df_newUser.at[0 , i[0]] = i[1]
    newUser_df = df_newUser.fillna(0)
    newUser_dict_sort = dict(sorted(newUser_dict.items(), key=lambda item: item[1],reverse=True))
    top3movies = {k: newUser_dict_sort[k] for k in list(newUser_dict_sort)[:3]}
    recs = make_recommendation(user_ratings, top3movies, newUser_df, movie_users_df,knn,10)
    return recs




import numpy as np
import pandas as pd
import pickle

#import kMeans
from sklearn.cluster import KMeans

#open reduced dataset from pickle file
object = pd.read_pickle('data/cleanedMovie.pkl')
movies = pd.DataFrame(object)

#open all movie titles
```

```
terms = pd.read_csv('data/movie_titles.txt', sep='\t', encoding = "ISO-8859-1", header=None, index_col=0)
terms = terms.iloc[:,:2]
terms = terms.iloc[:,1:]

#pivot the data into a customer x movie matrix
movieMatrix = movies.pivot_table(values='Rating', index='CustomerID', columns='MovieID')
movieMatrix = movieMatrix.fillna(0)
movie_arr = np.array(movieMatrix)

#run kmeans and fit the data
kmeans = KMeans(n_clusters=10)
kmeans.fit(movie_arr)

#function to return movie title based on ID
def toTitle(MovieID):
    return terms.loc[ MovieID , : ][2]

#prints recommendations
def top_movies_user(df, n, user_ratings):
    #checks if a viewer has already seen a recommendation
    watched = []
    for movie in user_ratings:
        watched.append([movie][0][0])
    moviesreturned = 0
    movieidx = 0
    recs = []
    while moviesreturned != n:
        #if a movie hasn't been seen, we can print it out as a recommendation
        if df.index[movieidx] not in watched:
            print(moviesreturned+1, toTitle(df.index[movieidx]))
            recs.append([df.index[movieidx],toTitle(df.index[movieidx])])
            movieidx += 1
            moviesreturned +=1

        else:
            movieidx +=1
    #return the movies that were recommended
    return recs

#prints a specific cluster
def print_clust(kmeans, k, n, user_ratings):
        #accesses cluster data from kmeans
        clust = pd.DataFrame(kmeans.cluster_centers_[k-1])
        clust.index = movieMatrix.columns
        #print(clust)
        #sortDF = pd.DataFrame(clust,terms)
        #print(sortDF)
        sortDF = clust.sort_values(by=[0],ascending=False)
        #print(sortDF.loc[sortDF.index[0]][0])
        #print(sortDF)
        print("Top movies in Cluster", k)
        #calls function above to print movies
        recs = top_movies_user(sortDF, n, user_ratings)
        print("")
```

```
        return recs

#converts user supplied data into a full data row consisting of all movies - 0s subsitutied for movies not yet seen
def user_row(user_data):
    out = pd.DataFrame(np.zeros(len(movieMatrix.columns)), index=movieMatrix.columns).T
    for rating in user_data:
        out[rating[0]] = rating[1]
    out = np.array(out)
    return out

#function that needs to be called to run clustering, passes in the user created ratings list
def cluster(user_ratings):
    user_data = user_row(user_ratings)
    #determines the users cluster based on the model
    user_cluster = kmeans.predict(user_data)
    print("You have been assigned to cluster:", user_cluster[0]+1)
    print("We recommend: ")
    recs = print_clust(kmeans, user_cluster[0]+1, 10, user_data)
    #returns recommended movies back to main.py
    return recs
```

Sample Run:

Enter name and pick which set of movies to rate:

```
(base) PS C:\Users\rober> cd C:\Users\rober\machlearning\DSC478_Final_Project\app
(base) PS C:\Users\rober\machlearning\DSC478_Final_Project\app> python main.py
Welcome to our movie recommender system!
Please enter your name: Bob
We ask you to first rate some movies.
a Rate the 20 most popular movies?
b Rate 20 random movies?
Please choose option a or ba
```

Sample data the user inputs:

```
These are the top 20 most watched movies on our system:
Please rate at least 5 movies on a scale of 1 to 5 (1,2,3,4,5), you haven't seen it just hit Enter.
We can then start providing you recommendations based on your initial input!

You are rating Forrest Gump
Rate from 1 to 5: 4
You rated Forrest Gump as a 4

You are rating The Sixth Sense
Rate from 1 to 5:
You skipped The Sixth Sense

You are rating Pirates of the Caribbean: The Curse of the Black Pearl
Rate from 1 to 5: 4
You rated Pirates of the Caribbean: The Curse of the Black Pearl as a 4

You are rating The Matrix
Rate from 1 to 5: 4
You rated The Matrix as a 4

You are rating Spider-Man
Rate from 1 to 5: 5
You rated Spider-Man as a 5

You are rating Men in Black
Rate from 1 to 5:
You skipped Men in Black

You are rating The Silence of the Lambs
Rate from 1 to 5: 5
You rated The Silence of the Lambs as a 5

You are rating Independence Day
Rate from 1 to 5:
You skipped Independence Day

You are rating Jurassic Park
Rate from 1 to 5: 4
You rated Jurassic Park as a 4

You are rating Gladiator
Rate from 1 to 5:
You skipped Gladiator

You are rating Ferris Bueller's Day Off
Rate from 1 to 5: 4
You rated Ferris Bueller's Day Off as a 4

You are rating Ocean's Eleven
Rate from 1 to 5: 4
You rated Ocean's Eleven as a 4

You are rating Raiders of the Lost Ark
Rate from 1 to 5: 5
You rated Raiders of the Lost Ark as a 5

You are rating Lord of the Rings: The Fellowship of the Ring
Rate from 1 to 5: 4
You rated Lord of the Rings: The Fellowship of the Ring as a 4

You are rating Minority Report
Rate from 1 to 5:
You skipped Minority Report

You are rating Speed
Rate from 1 to 5:
You skipped Speed

You are rating Meet the Parents
Rate from 1 to 5:
You skipped Meet the Parents

You are rating The Fugitive
Rate from 1 to 5:
You skipped The Fugitive

You are rating Braveheart
Rate from 1 to 5:
You skipped Braveheart

You are rating Pulp Fiction
Rate from 1 to 5: 5
You rated Pulp Fiction as a 5
```

Results of Clustering on user inputs:

```
Our system supports Clustering, Knn User, Knn Item, and SVD, each offering different recommendations!
Which would you like to run?
a Clustering
b Knn User-Based
c Knn Item-Based
d SVD
Please choose option a, b, c, or d a
Clustering...
Please Wait
You have been assigned to cluster: 4
We recommend:
Top movies in Cluster 4
1 Raiders of the Lost Ark
2 Lord of the Rings: The Fellowship of the Ring
3 Lord of the Rings: The Two Towers
4 The Matrix
5 Pirates of the Caribbean: The Curse of the Black Pearl
6 Indiana Jones and the Last Crusade
7 Lord of the Rings: The Return of the King
8 The Terminator
9 The Sixth Sense
10 Star Wars: Episode V: The Empire Strikes Back

Would you like to rate any of the recommendations? y/n
```

Results of Knn user based:

```
Our system supports Clustering, Knn User, Knn Item, and SVD, each offering different recommendations!
Which would you like to run?
a Clustering
b Knn User-Based
c Knn Item-Based
d SVD
Please choose option a, b, c, or d b
KNN user-based...
Please Wait
Recommended Movies:
1 Seven
2 Gladiator
3 Saving Private Ryan
4 Braveheart
5 Memento
6 The Hunt for Red October
7 The Usual Suspects
8 Lord of the Rings: The Two Towers
9 True Lies
10 Lord of the Rings: The Return of the King
Would you like to rate any of the recommendations? y/n
```

Results of Knn item based:

```
Our system supports Clustering, Knn User, Knn Item, and SVD, each offering different recommendations!
Which would you like to run?
a Clustering
b Knn User-Based
c Knn Item-Based
d SVD
Please choose option a, b, c, or d c
KNN item-based...
Please Wait
Recommended Movies:
1 Still
2 The Inspector Lynley Mysteries: A Traitor to Memory
3 Outrage
4 Layer Cake
5 The Fassbinder Collection: Pioneer in Ingolstadt
6 Love Affair
7 Buried in the Sand: The Deception of America
8 The Salton Sea
9 Anaconda
10 First to Die
Would you like to rate any of the recommendations? y/nn
Run a different algorithm? y/ny
```

Results of SVD

```
Our system supports Clustering, Knn User, Knn Item, and SVD, each offering different recommendations!
Which would you like to run?
a Clustering
b Knn User-Based
c Knn Item-Based
d SVD
Please choose option a, b, c, or d d
SVD...
Please Wait

Recommended movies:
1 Lord of the Rings: The Return of the King: Extended Edition: Bonus Material
2 Empire Falls
3 A Piece of the Action
4 Sesame Street: Learning About Letters
5 The Barkleys of Broadway
6 NBA Street Series: Ankle Breakers: Vol. 2
7 American Adobo
8 Halloween: H2O
9 Yes: Symphonic Live
Would you like to rate any of the recommendations? y/n
```

Additionally users can rate any of the movies we've recommended to them, this gets stored with their ratings and is used in subsequent runs of any of the algorithms

```
Recommended Movies:
1 Seven
2 Gladiator
3 Saving Private Ryan
4 Braveheart
5 Memento
6 The Hunt for Red October
7 The Usual Suspects
8 Lord of the Rings: The Two Towers
9 True Lies
10 Lord of the Rings: The Return of the King
Would you like to rate any of the recommendations? y/ny
Please rate on a scale of 1 to 5 (1,2,3,4,5), you haven't seen it just hit Enter.
We can then update your recommendations based on your newly added ratings!
Enter index of movie you want to rate from the recommendations above: 5
You are rating: Memento
Rate from 1 to 5: 5
You rated: Memento as a 5
Continue rating? y/n
```