COMP1862 (2023/24)	**Final Year Projects**		**Contribution: 100% of course**
Project Supervisor: Pushparajah Rajaguru	**Emirhan Ülgen - 001150599**		**Deadline Date: Tuesday 12/07/2024**

This coursework should take an average student who is up-to-date with tutorial work approximately 50 hours

Feedback and grades are normally made available within 15 working days of the coursework deadline

Learning Outcomes:

1 Deploy theory, design principles, tools and methodologies to implement and evaluate human- computer interactions;

2 Carry out design research to inform development of systems and applications;

3 Construct and create prototypes of human-computer interactions;

4 Demonstrate the origins of ideas by correctly citing and referencing sources used in the work.

Plagiarism is presenting somebody else's work as your own. It includes: copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing coursework from another student and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see the assessment misconduct procedure for further details of what is / isn't plagiarism.

Note that submitting writing generated by AI tools as your own work might an Academic Offence (see also the Guidance on the use of artificial intelligence (AI)), and penalties apply as detailed in above procedures.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using.

Your work will be submitted for plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Table of Contents

List of Figures

List of Tables

Generating a Movie Recommendation System Using Machine Learning

Abstract

Nowadays, the recommendation system makes finding what people need easy. Movie recommendation systems focus on helping movie watchers by recommending movies to watch avoiding the process of deciding from a huge set of movies that could go up to thousands and even millions, so it is confusing. In this world, where technology plays a big role in every industry, the amount of information and data has increased. A recommendation system can therefore filter out the useful information that is quick and relevant to the user's choice from such a large range of data. This report demonstrates how a movie is chosen, processed, and recommended to users. There is also a detailed description of the algorithms implemented. Besides that, it provides information about the recommendation models, but focuses on one way. The report contains the strengths and weaknesses of the system based on the findings and usage.

Preface

This project was initiated with the objective of how machine learning is effective in the cinema sector. People used to go to shops that sell various genres of movies such as drama, horror, comedy, etc. Workers in such shops could help people make a decision on what movie to watch. Over the course of this work, how artificial intelligence does this in platforms like Netflix, Disney Plus, and other steaming websites. The main idea behind

this is to lay emphasis on the capabilities of machine learning, therefore artificial intelligence, to do some actions usually done by humankind. This preface focuses on an overview of the project's motivation and algorithms that facilitated to go through with the project, setting the stage for a detailed discussion in the following chapters.

Keywords

Recommendation systems, machine learning, movie recommendation, KNN algorithm, collaborative filtering, cosine similarity

Introduction

The Internet has become common quickly and its rising continues day by day since it was invented. Access to the correct information in quick and easy way has become an exhausting task thanks to lots of information found online (Nassar, et al., 2020). The development of technology provides several advanced platforms, namely Machine Learning, Deep Learning, the Internet of Things (IoT), etc., which are all necessary to provide for society (Furtado & Singh, 2020). It also provides Artificial Intelligence as a real-life application (Haenlein & Kaplan, 2019). Artificial intelligence facilitates individuals in many ways. It is simply there for people. When people have struggles deciding what to do, they get help from others. It is possible to get help with what to do from artificial intelligence nowadays. Individuals challenging with making decisions

between things, keeping in dilemmas, this may be because they don’t have time to be in this situation when they go to market, has opened a gate for the idea of recommendation systems (Hande, et al., 2016). It is extra challenging for people to decide what book to read, what music listen to, and what movie to watch in particular where there are almost 15 million films available in libraries, this makes the need of a recommendation system inevitable in order to ensure the entertainment of both movie service workers and customers (Kumar, et al., 2015). This aims to bring a huge comfort on individuals struggling with choices and organizations to take advantage of this system to attract more clients by growing up the satisfaction of the users (Virk, et al., 2015).

In the modern era, recommendation systems are widely used in a variety of applications such as e-commerce, retail, banking, entertainment, etc. These systems take and observe the user data automatically to create personalized recommendations for users (Beheshti, et al., 2020). The most common methods to implement recommendation system are content based filtering (CBF), collaborative filtering (CF) and hybrid filtering (HF) (Sharma, et al., 2022). Many recommendation systems prefer the hybrid filtering method which involves the properties of both content based filtering (CBF) and collaborative filtering (CF) (Reddy, et al., 2019). The popularity of a movie is based on the reviews from the audience. Those reviews play an important role in the decisions of other users as well. Users are more prone to choose a movie that was watched by most people instead of a movie largely disliked (Yasen & Tedmori, 2019). This project aims

at generating a movie recommendation system in which users can see what they look for as well as the other movies related to that particular one.

Literature Review

Furtado and Singh (2020) even discuss that there is an algorithm implemented for this task called K Nearest Neighbor (KNN) algorithm, as defined by (Cui, 2017).

Furtado and Singh explain an algorithm where if the majority of neighbors tested within the component space belong to a class, then it can be concluded that the given example also belongs to that category. The theory is illustrated in Figure 1 below, where we can see that the nearest neighbors of W are found in the X class indicating that W should be classified as belonging to X. (Oyelade, et al., 2010).

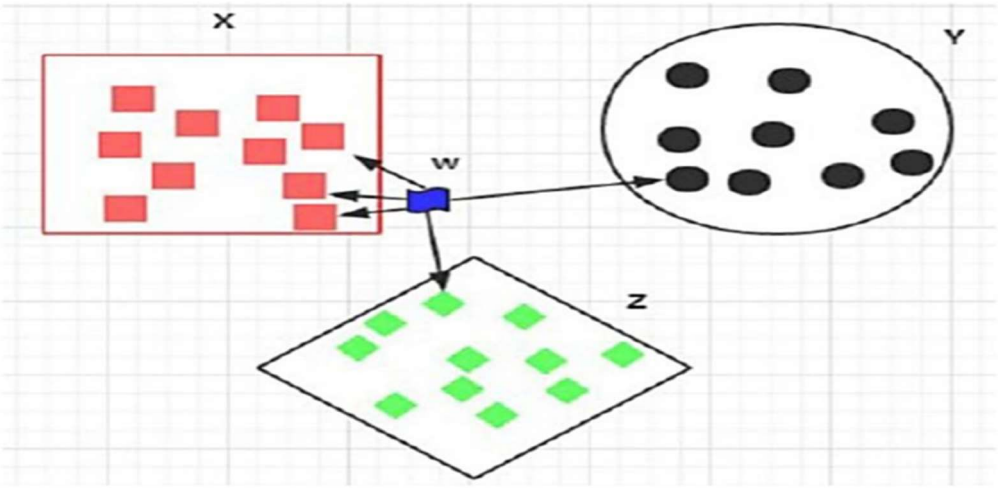

Figure 2: K nearest algorithm

Using this method, the neighbors play an important role on the recommendations to be generated since they are used for prediction. As a result, this should be done with caution to keep the structure of nature of the suggestions made. Therefore, the

neighbors are compared and those who have the highest relation are selected. Hence, this value has to be picked up carefully.

The users who are asked to predict the movies that are not rated, should use the similar weights calculated in the previous steps to make a prediction.

To understand this algorithm better, it is important to know what KNN is and how it works exactly.

What is KNN?

The K-Nearest Neighbors (KNN) is a classification algorithm used in machine learning. It is nonparametric which means it does not make any assumptions on the dataset (Taunk, et al., 2019). Its conciseness and effectiveness make it a good choice. It is a supervised learning algorithm. A labeled dataset is given with data points classified into different categories. It is the algorithm's job to predict which class the unlabeled data belongs to.

Various specifications detect the class of the unlabeled data in classification. KNN is mostly used for this method, so it categorizes data regarding the closest or neighbour examples in a region. Due to its simplicity of execution and low calculation time, this method is commonly used.

For a new input, the K nearest neighbours are computed and the majority amongst the neighbours is meant to make decision for the new input. The value of 'K' is crucial in categorizing the unlabeled data despite the simplicity of the algorithm.

There are lots of ways to choose ‘K’, however, the classifier could be tried several times with various values to see which one brings out the best result. The calculation cost is moderately high, and the reason is all the computations are made during the classification of the data, not its interaction in the dataset.

Taunk, et al. (2019) puts down the classification method into two steps, since KNN is a classification algorithm:

1. Learning step in which the data is used where a classifier is built.
2. Analysis of the classifier.

KNN does two operations by the time an unlabeled data is included in the dataset. It first assesses the K points that are closest to the new data point and then decides which class the new data should belong to. Figure 2 demonstrates the structure of the algorithm.

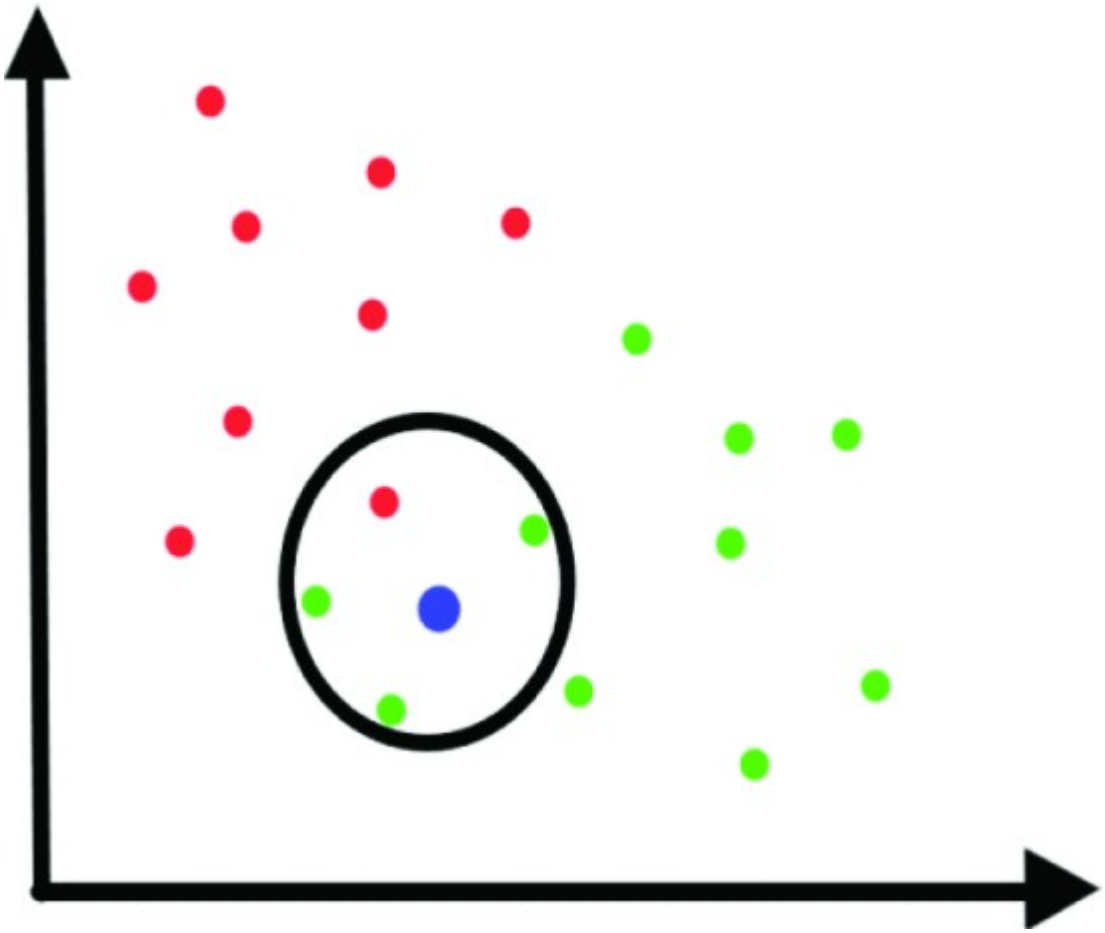

Figure 3: A simple KNN

Some new data is included, and the data is categorized in accordance with them. This method is more effective when a dataset is randomly pieced into clusters and has a specific region.

By doing so, the algorithm assures a better result in separating the data inputs into various classes in a better way. KNN finds out the class which has the maximum number of points with the lowest distance from the data point to be categorized.

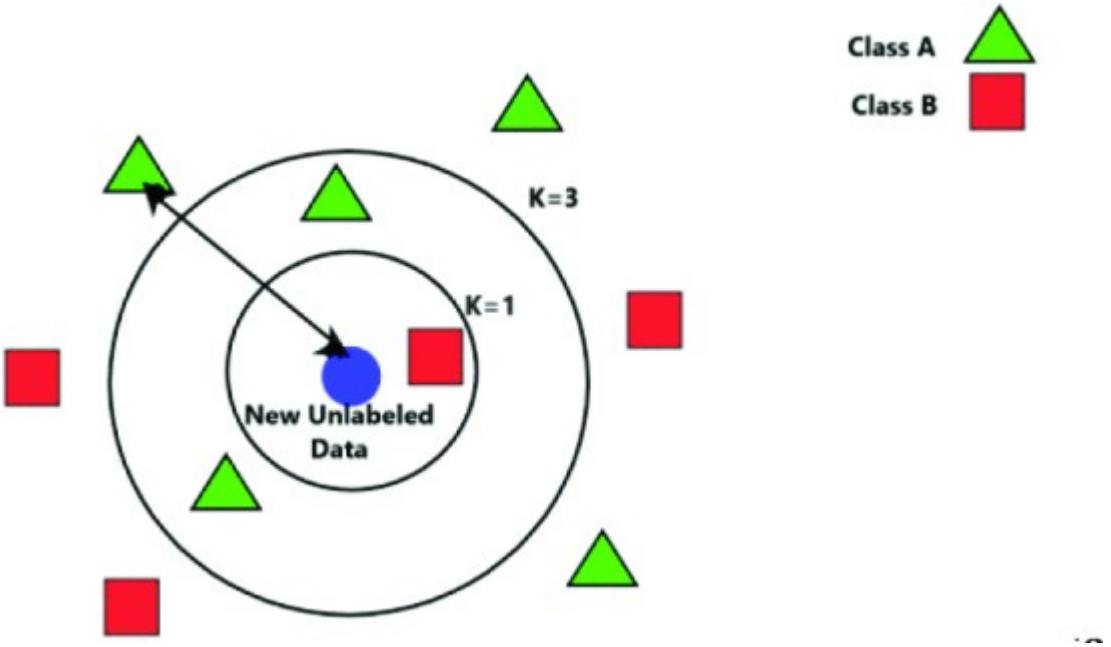

Figure 4: KNN with two classes A and B.

Figure 3 proves that the unlabeled data measures its distance from each neighbor regarding the value of K. It then decides which class it is included in, considering the maximum number of closest neighbors.

Ahuja, et al. (2019) demonstrate their own architecture of the system they proposed for this KNN algorithm in Figure 2 below. This clues in the conceptual idea of the recommendation system proposed. Every single module done is illustrated with details and the architecture of the system is described. In this way, the structure of the system is more easily comprehended.

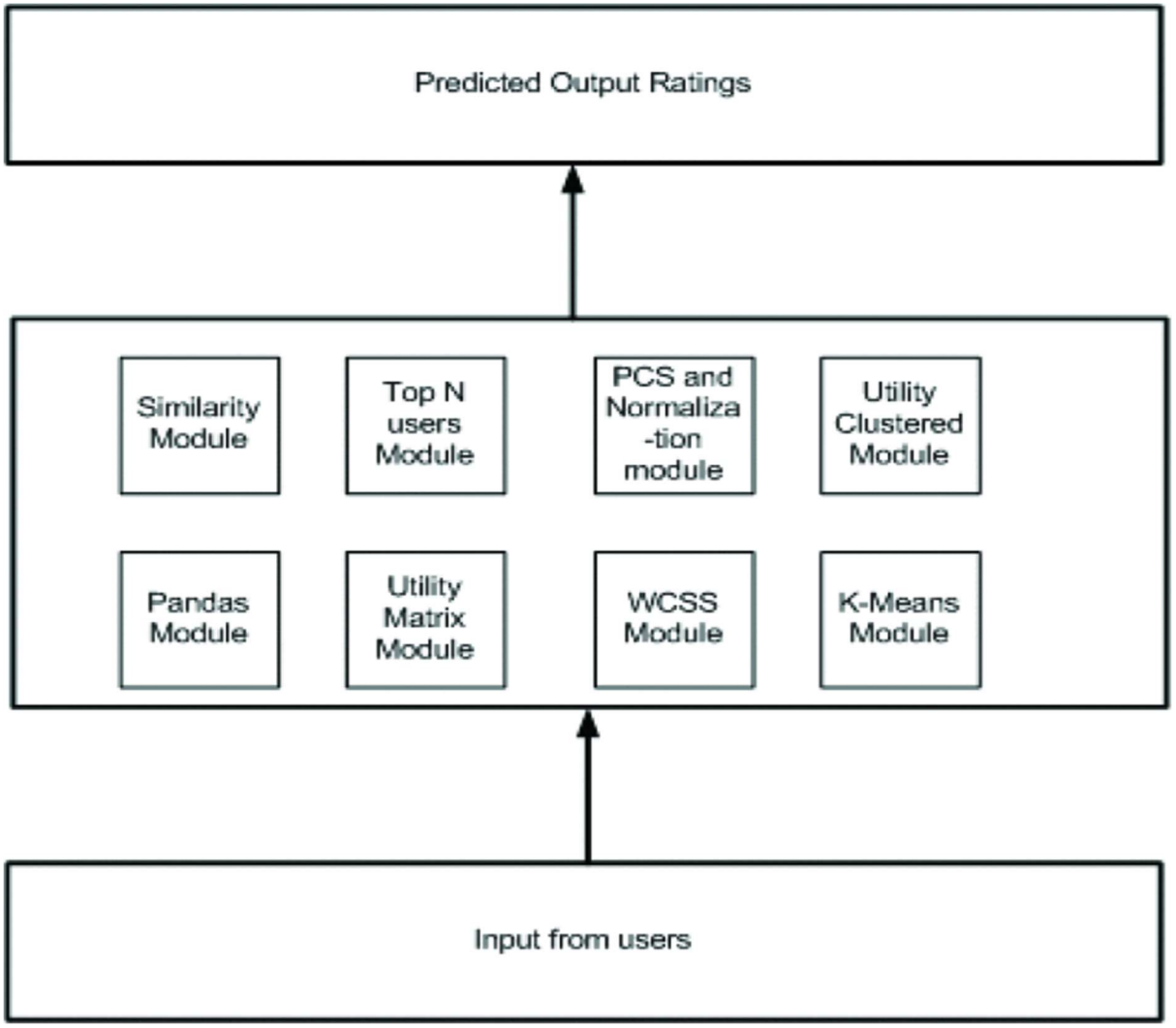

Figure 5: Architecture of the proposed system

As it can be observed in the Figure 3 above, there are three main modules in the architecture:

Input Module: Users enter the details as input about themselves like userId.

Process Module: This is really a long process since it is the processing module, as the name implies. A library called Pandas is used in this step to distinguish the data within the raw files,

puts the user and movie information into different data frames with athe help of the panda's library. It then builds a utility matrix in a module used to illustrate which movie is rated by which user. This facilitates finding out the number of ratings a movie has. After that, different data frames for the train and testing set are created with the use of previous data processing in order to observe the system performance in detail. Once the utility matrix is received, a new data frame is built to provide information of the relationship a movie has with the genres with the use of K-means clustering. A cluster possessing a small total value of squares is usually more complex than a cluster with a huge total value of squares. The correct number of clusters is selected with the use of the Within Cluster Sum of Squares (WCSS) which measures the variability of the analysis in each cluster. The PCS and the module of normalization computes the connection with the use of the utility-clustered matrix. Eventually, predictions for movie rating in KNN and similarity modules are computed with the use of the similarity matrix and utility- clustered matrix, which is for finding the similarity matrix. This is done with the help of K Nearest Neighbors.

- Output Module: In this module, the predicted movies that might be interesting to users are defined. Then, the interpreted ratings are defined as that could be useful for the movies since users would use them to rate the movies.

Ahuja and his colleagues also provide a diagram which illustrates how the proposed system is processed. Figure 2 below demonstrates the function of the system, how the raw data is being used and how the system interprets the rating for the input “userId”.

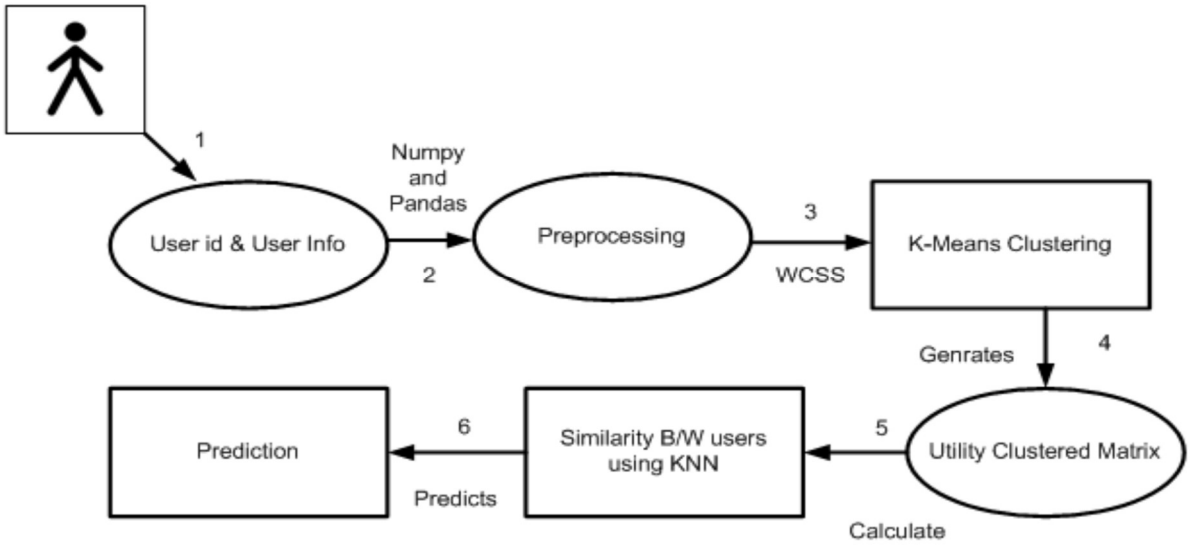

Figure 6: Process Flow Diagram

There are six steps in the process flow as shown above:

The user enters the userId and gender, pin code, etc.

The raw data is processed into various data frames with the use of Pandas and Numpy libraries.

WCSS measures the accurate number of clusters, so K-means clustering is ready to be applied to the movie.

- Next step is in which a utility-clustered matrix is created to store the average rating given by users to each cluster.

- The similarity value between the users is found using clustered matrix and Pearson correlation.
- The utility-clustered matrix and similarity are used by KNN to interpret the movies for input user.

When applying this algorithm to the movie recommender system, data is either meant to be the description, genre, or the ratings of the movies while the distance is the similarity between those terms. Similarity between the ratings means the values closest to the target value. Singh et al. (2020) provides three different methods to calculate the similarity:

- Cosine similarity
- Euclidean distance
- Pearson's correlation

Cosine similarity is the popular method for this particular system. It calculates cosine angle between two objects by finding the scalar product which is the closeness of the two objects.

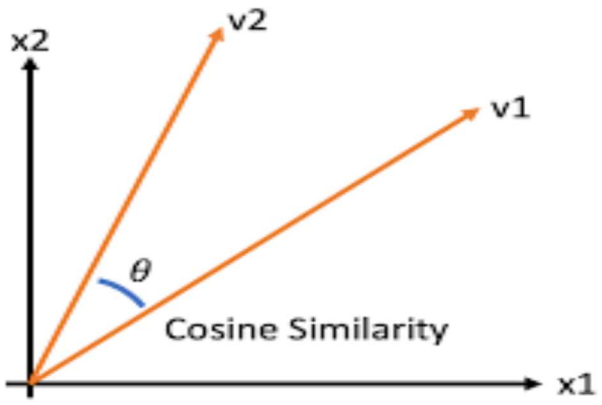

Figure 7: The visual representation of the cosine angle. (Sambhwani, 2020)

According to Figure 7, the arrows v1 and v2 represent the vectors and the angle between them is the angle of cosine. The size of the angle is inversely proportional similarity of the vectors. If the angle between the vectors is small, they are more likely to be similar; however, they are less similar when the angle is large. In other words, the closer the vectors are, more likely to look like each other the vectors are.

Airen and Agrawal (2022) propose another architecture as observed below:

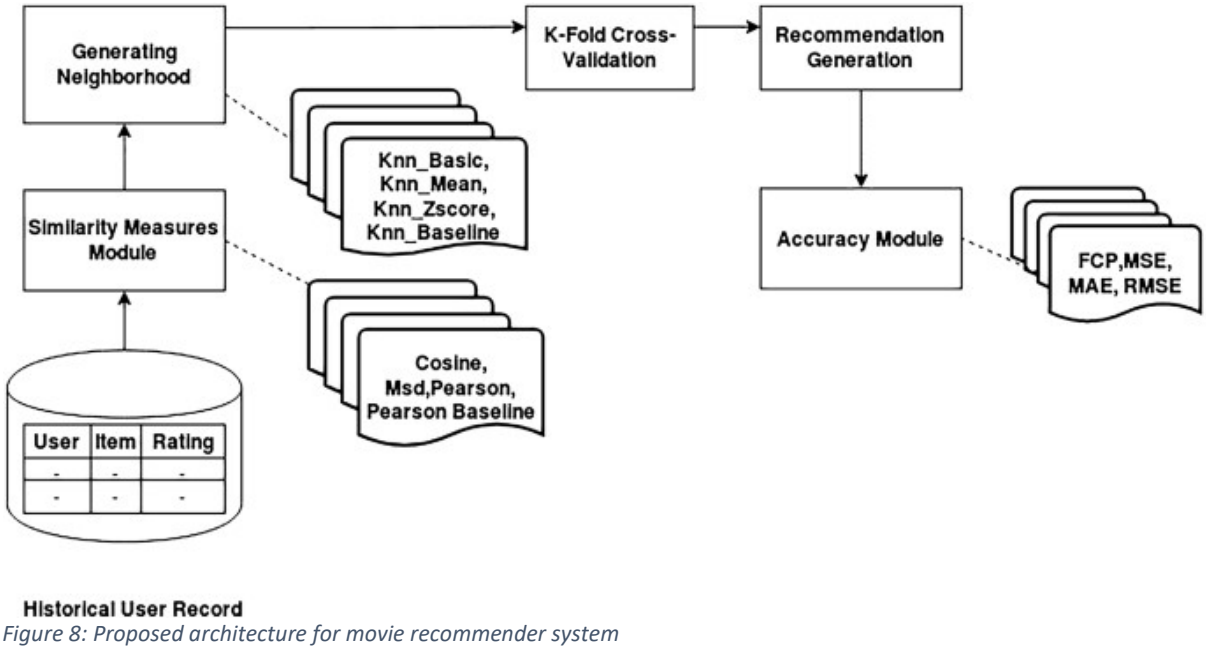

Figure 8: Proposed architecture for movie recommender system

They define their architecture as their movie recommendation system using various KNN algorithms and various similarity measures as observed in Figure 7. The historical data was used to aim at recommendation. Similarity between users is measured with the use of user rating matrix. They used four different types of similarities, cosine, msd, pearson, and pearson baseline, are measured with the help of user rating matrix. After the similarities are measured, the Collaborative Filtering

recommendation algorithm based on KNNs is employed with five fold cross validation before movie recommendation is generated. The comparison of various metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and fraction of concordant pairs (FCP) on different values of the number of nearest neighbors is made. No research has ever been carried out on movie recommendation system using K Nearest Neighbours variants such as KNN-Basic, KNN-WithMeans, KNN-WithZScore, KNN-Baseline on the four similarity calculation methods, i.e. cosine, msd, pearson and pearson baseline similarities, as Airen and Agrawal know.

Beside these architectures, there have been different works and research that are existing. A number of authors discussed their work that are with respect to recommendation systems based on K Nearest Neighbors (KNN) algorithms. These works are what other people have done related to this project as not all of the methods and terms they used are included in the intended project to be proposed.

Related Work

There have been several existing works related to Recommendation System based on K Nearest Neighbor (KNN). Calculation of similarity is an essential step in KNN based recommendation system. Ayub, et al. (2019) suggested a new technique named Improved PCC (Pearson Correlation Coefficient) Weighted with RPB (Rating Preference Behavior)

(IPWR) as a similarity measurement. It uses Rating Preference Behavior (RPB), which is dependent on the average ratings of users and standard deviation, of users. Resnick, et al. (1994) used K Nearest Neighbor based collaborative filtering algorithm to form neighborhood. Schafer, et al. (2007) discussed the design of rating system, acquiring ratings, and privacy issues of collaborative filtering. According to Adomavicius and Tuzhilin (2005), lots of authors suggest collaborative filtering with clustering. In ClustKnn (Al Mamunur, et al., 2006), clustering model is constructed initially and during recommendations, KNN approach is used to predict ratings. Ahuja and the colleagues debated about the generation of movie recommendations by grouping KNN with k means clustering. They reached a conclusion that the value of root mean squared error (RMSE) reduces with respect to the reducing number of clusters. Lund and Ng (2018) mention auto encoders based movie recommendation system. Movie swarms is created in order to fix the initial problem of new users and items in movie recommender system (Halder , et al., 2012).

Similarity matrix is built in neighbour based recommender system that uses collaborative filtering in order to calculate similarities between items. Neighbourhood model is factored to eliminate the scalability problem of similarity matrix (Koren, 2010). Bahadorpour, et al. (2017) found ideal number of nearest neighbors for movie recommender service with the help of item based KNN collaborative filtering. Shani and Gunawardana (2011) argue different metrics to see the performance of user

prediction such as accuracy metrics, error metrics, and statistical metrics.

Nguyen and his colleagues proposed a web based movie recommendation system that uses hybrid filtering method in 2007. A movie recommender system that uses genre correlations is presented by Ko, et al. in 2011. In 2013 a movie recommendation system based on Bayesian network, imported for modeling user preference, and trust model, this filters the recommending history data and allow the system to tolerate the unnecessary data, is released (Wei & Junliang, 2013). Recommender systems to estimate the rating for users and items, mainly from large data to recommend the favorites, is suggested in 2016. Movie recommendation systems enable a system to help users categorize users with similar interests. This system, K-Mean Cuckoo, has a mean absolute error of 0.68 (Adeniyi, et al., 2014) (Kataria & Verma, 2016). In 2017 a new technique that is able to fix sparsity problem to a good extent is proposed (Mishra, et al., 2017). Das and his colleagues developed a recommendation engine by assessing rating data sets taken from Twitter to suggest movies in order to specific user with the use of R.

Main Chapters

Analysis

The biggest problem when deciding to watch a movie could be the challenge to pick one among the millions of movies available. People would want to see the options of specific features, like they would like to see comedy movies if they want to have a fun time. Surrendran, et al. (2020) propose two main approaches used for this: content-based filtering and collaborative filtering. Collaborative filtering is used to gather similar users and use the relevant information to make recommendations to the user, while content-based filtering focuses on user's interests with the use of information collected and recommend items based on the user profile.

Collaborative Filtering: Surrendran, et al. (2020) define collaborative filtering as a technique for the prediction of preferences of users with the use of the preferences that are already known. The similarity is calculated on two bases: the user and the item. The uses of cosine and Pearson correlation similarity approach are mentioned in this method as in the KNN algorithm. Collaborative filtering mostly deals with the density, scalability of data and cold start problem. It provides three algorithms, namely, memory-based, model-based, and hybrid collaborative filtering (CF). These link CF with other recommendation techniques and their impact on coping with the difficulties.

As an example, two users, James and Creed have similar tastes. If the ratings of these two are very similar, their similarity can be

expressed by the fundamental algorithm. In this case, it is highly possible that the ratings are likely to be alike. This resemblance can be useful for predictions about partly stated values.

- Content Based Filtering: This method focuses on a description of the item and a profile of the user's preferences. These are the best to use where known data on an item such as name, description, etc. exists, however, the user does not have this information. Content-based recommenders see recommendation as a classification problem which focuses on a user and learns a classifier for the user's interests as well as dislikes about the features of an item.

Keywords play a significant role on items to build a user profile to show the type of item the user might be interested in. These algorithms deal with recommending items that are like those the user was familiar with in the past. It does not depend on a user sign-in mechanism to create this sequent temporary profile. Especially, different candidate items are compared with rated items and those which have the most rate of match are recommended.

Hybrid Filtering: Hybrid filtering method contains both collaborative and content based filtering methods. Singh et al. (2020) states that this is the most widely used technique nowadays. It tries to be thorough, so it avoids the vulnerabilities

of the recommender techniques. There are several issues about the recommendation system:

- Cold-start problem: A new register is someone who has not watched a movie yet. The system therefore has troubles finding similar movies since it does not have any movie (Muozorganero, et al., 2010). This is named as the cold-start problem (Sarwar, et al., 2001). The system goes through this issue.
- Data sparsity problem: This happens when the user has given ratings to very few items, so it becomes a problem for the recommendation system to bring out correct results. In this problem, the results are not quite similar to the estimated result. The system might even fail to give successful results and creates bad recommendations (Wang, 2012). Data sparsity brings about coverage issues as well.
- Scalability: This is another issue in which the relationship between the encoding and items is linear. The system is efficient with the dataset that has a limited size (Albadvi & Shahbazi, 2009). The recommendation system struggles to give good results considering the genres of the movies when the dataset increases.
- Synonym: When the words have a similar meaning, the system fails to interpret the difference between the two likely words and thus becomes unable to produce the expected output. For instance, words film and movie carry the same meanings, but the system sees

> them as different words. Singular Value Decomposition (SVD), Latent Semantic Indexing in particular, can solve this synonymy problem (Hernandez & Garcia, 2016).

The popular and preferred method for a recommendation system is the collaborative filtering in which the ratings are used for the system. It filters out the content based on the users' similar interest with others, it technically suggests the items to users with similar taste (Sarwar, et al., 2001). Two common filtering algorithms exist in the memory based techniques (Molina & Bhulai, 2018). Another method is called model based, but it is not that reliable when compared to other memory based methods (Breese, et al., 2013). Gupta, et al (2020) provides

two diagrams, Figures 7 and 8, on how collaborative filtering works based on users and items.

In the user based, the user is presumed to like the items which are liked by users he is similar with (Phorasim & Yu, 2017). Table

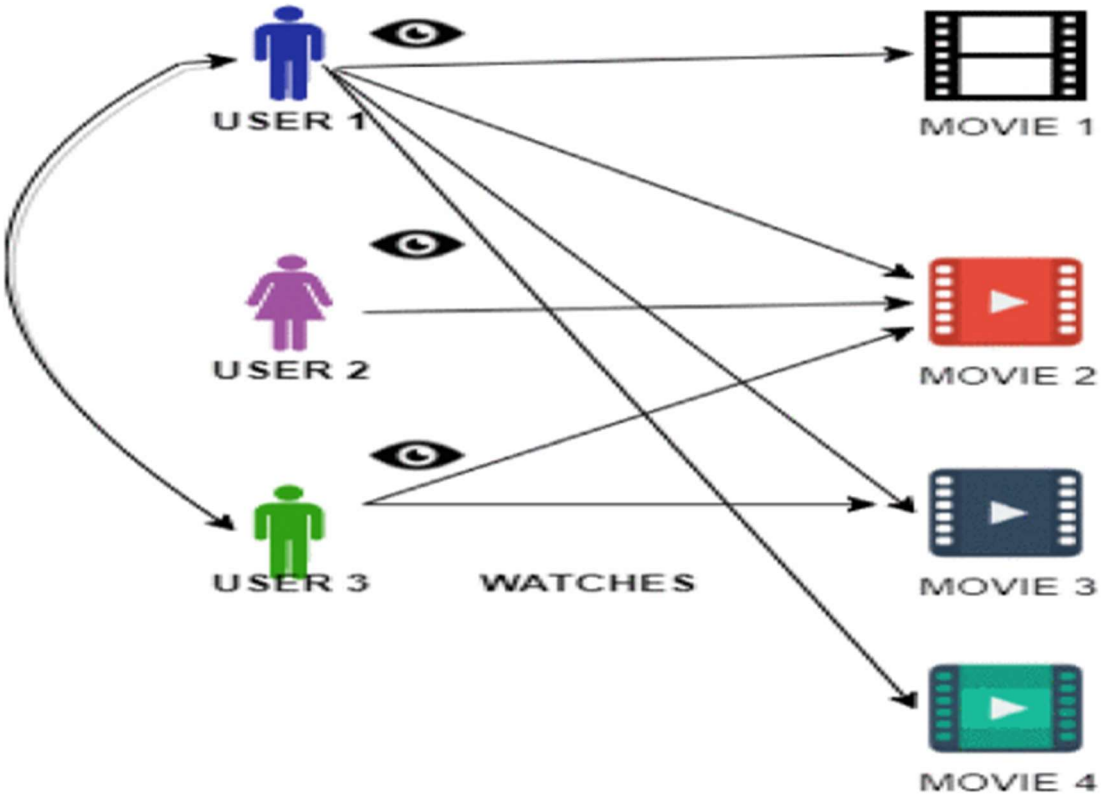

Figure 7: Demonstration of user-based CF

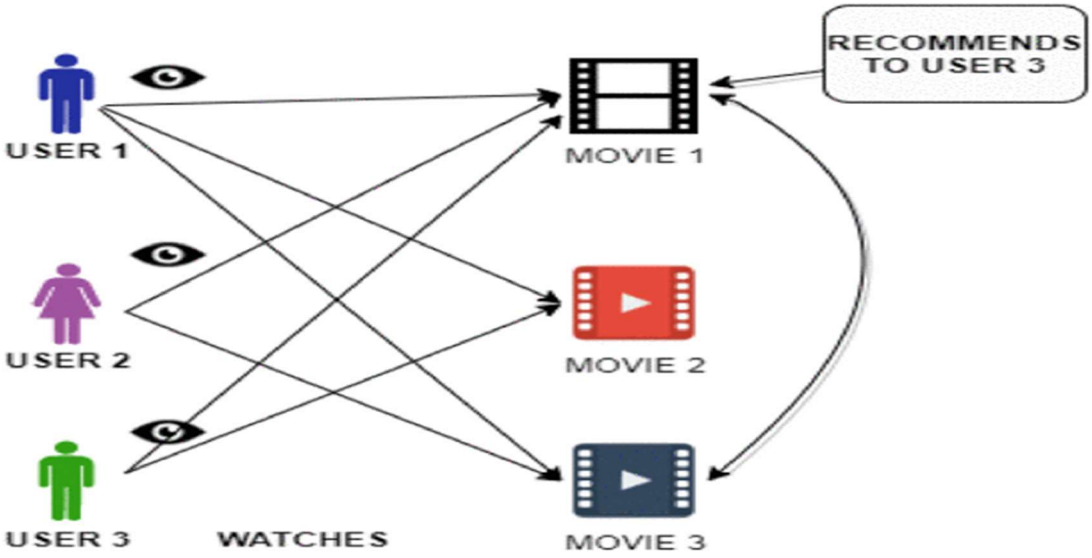

Figure 8: Demonstration of item based CF

1 is another example in which all the users like item A and people that like A like item C as well.

USER/ITEM	ITEM A	ITEM B	ITEM C
USER A	✓		✓
USER B	✓	✓	✓
USER C	✓		Recommended

Table 1: Item-based collaborative filtering

The user is expected to like the items similar to the other previously liked items (Sarwar, et al., 2001). This hybrid approach gives very accurate results with the help of both collaborative and content based filtering methods reducing the weaknesses of the algorithms at the same time. This integrated system is getting more familiar since it is better than both the methods (Pathak, et al., 2013).

Collaborative Filtering makes use of a database of item preferences of users to guess products a new user might be interested in (Su & Khoshgoftaar, 2009). Collaborative Filtering involves a list of 'm' users, and 'n' items, $\{i_1, i_2, \ldots, i_m\}$,. Every user, '$u_i$', has a list of items, Iu_i, rated by the user. The ratings can be explicit indications that range from 1 to 5 (Miller, et al., 2004). For instance, the list including users and the movies liked

or disliked by them, can be transformed to a user-item ratings matrix as shown below.

Table 2: List of the users and movies they liked or disliked

(a)
Alice: (like) Shrek, Snow White, (dislike) Superman
Bob: (like) Snow White, Superman, (dislike) spiderman
Chris: (like) spiderman, (dislike) Snow white
Tony: (like) Shrek, (dislike) Spiderman

Table 3: Conversion of the list to a user-item matrix

	Shrek	Snow White	Spider-man	Super-man
Alice	Like	Like		Dislike
Bob		Like	Dislike	Like
Chris		Dislike	Like	
Tony	Like		Dislike	?

As it can be observed above, Tony is considered to be an active user to be recommended for. The matrix lacks some values where users did not give feedback for items, the reason may be that they did not use them.

Collaborative Filtering should be able to deal with highly sparse data, scale with many users and items, and recommend in a short-time period. Some collaborative filtering systems like GroupLens, (Resnick, et al., 1994), make use of the user rating data to compute the similarity and make recommendations using those calculated similarity values. Memory-based methods are deployed into commercial organizations like Amazon and Barnes and Noble, the reason is the implementation is simpler and more effective (Linden, et al., 2003). Customization of Collaborative Filtering saves time for users to search. It also provides a higher customer loyalty, an increase in sales and advertising, and the advantage of targeted promotions (Ansari, et al., 2000). The system aimed to create also uses collaborative filtering.

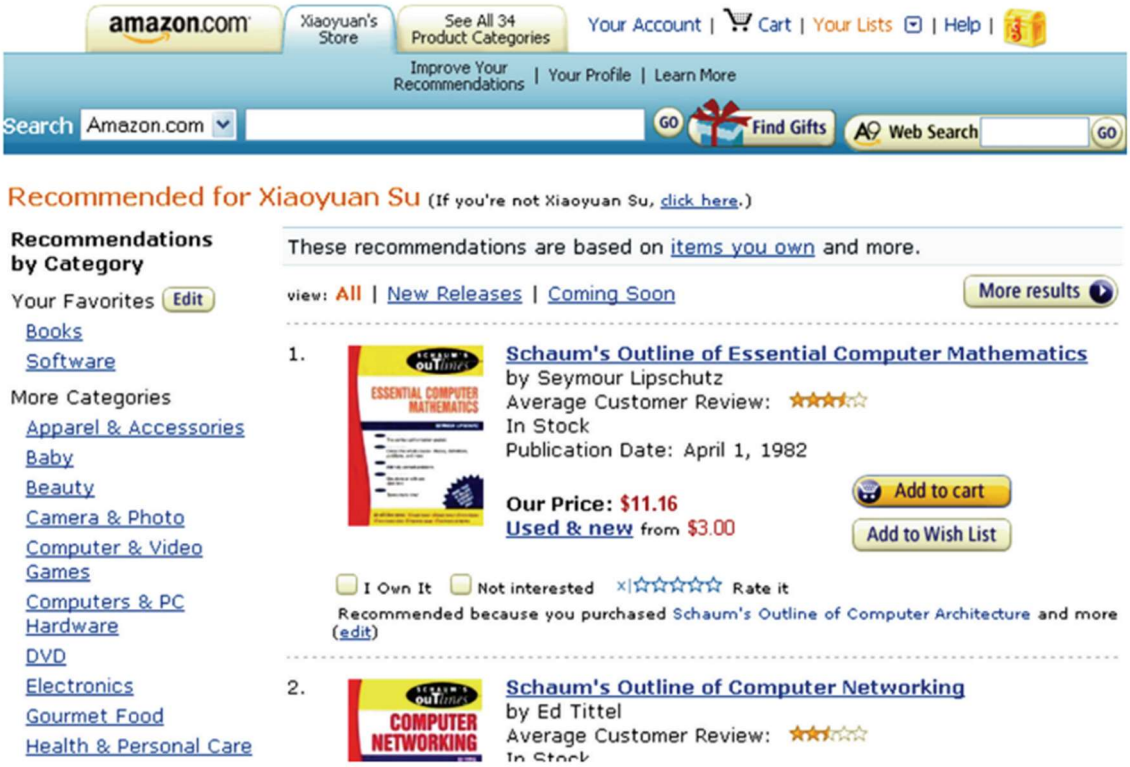

Figure 9: Amazon recommending products to customers by customizing CF systems.

One way to apply this collaborative filtering and bring out some movies that the user might like is to use the cosine similarity. Singh, et al (2020) provides a formula which is used to calculate the similarity of the movies that have different features. It demonstrates the cosine, cos, of the two vectors, or sparse matrix's angle using a multidimensional space in mathematical terms. This cosine similarity is very useful because it plays a key role in finding similar movies.

$$CosSim(x, y) = \frac{\sum_i x_i \, y_i}{\sqrt{\sum_i x_i^2} \, \sqrt{\sum_i y_i^2}}$$

Equation 1: The Cosine similarity formula which is used for the recommendation of movies.

x and y represent the vectors in the formula above. Their dot product is calculated and divided by the magnitudes of each vector.

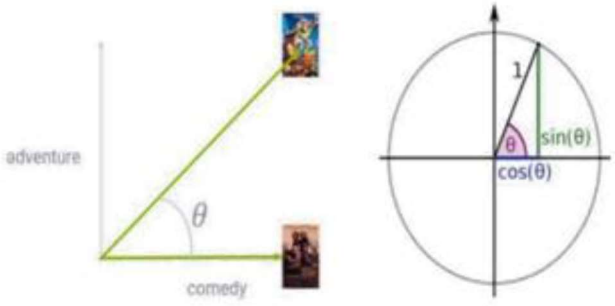

Figure 10:An example to implement Cosine similarity.

The theta, the value of cosine, differs from 0 to 1. If the value is close to 1, it means that it is most similar and least similar if it is close to 0. The movie will be recommended based on the

distance of the value to 1 since there would not be any similarity if it is far from 1. The relevant movies are recommended using cosine similarity.

Requirements Specification

The proposed recommendation system uses the collaborative filtering method as it is very popular and accurate as well as so useful. The system uses the K Nearest Neighbor (KNN) algorithm to find the distance, which represents the similarity, between the target movie with each movie included in the dataset and it lists the similar movies with the help of cosine similarity. The main requirements for the recommendation system to work are:

- K Nearest Neighbor (KNN) algorithm: This algorithm is used because it has the skill of faster prediction and low computation time. According to Liang, et al. (2014), it categorizes any class that is new to where they belong to with the use of the “k” value, shown in Figure 9.

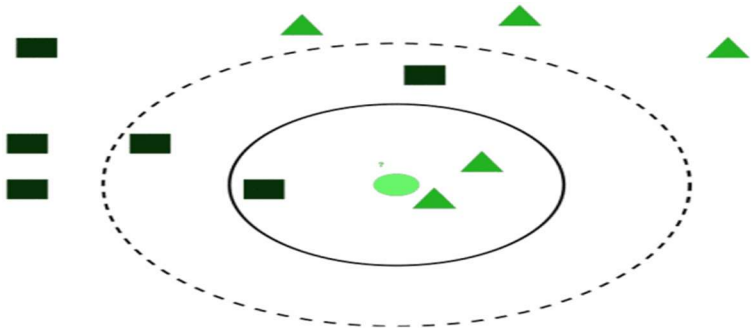

Figure 11: Demonstration of the K Nearest Neighbor algorithm (value of k=3) (Bahadorpour, et al., 2017).

- Cosine similarity: This is important to find the similarity between the target movie and the rest. It is used to calculate the similarity of two documents inversely related to their difference in size and finds the cosine angle found between the two vectors (Lahitani, et al., 2016).

- Item based collaborative filtering: This is used because the items, movies, will play a key role on the prediction of the

user so the technique is based on items. The idea is that the user is more prone to like the items that are similar to the items which are previously liked by the same user.

Before using the techniques above, a dataset that contains the movies as well as all its specifications, such as description, genre, ratings, etc., is needed. It can be collected from the Internet Movie Database (IMDB) website. Python is the essential language to implement those techniques. Some libraries are required to import the dataset and process it as well as implementing the algorithm. Jupyter notebook is needed to work everything out. It has the most useful and efficient interface so everything required can be done quickly and easily.

Evaluation Metrics

The experimental work was done using Python. The performances of similarity measures are calculated based on mean squared error (MSE), mean absolute error (MAE) precision and recall for top “n” recommendations.

Precision and Recall

For recommendations, movie ratings used usually range from 1 to 5. The precision and recall metrics are binary metrics that are calculated based on the binary output generated by a model. Airen and Agrawal (2022) define precision and recall as follows:

Equation 2: The Formula of Precision (Airen & Agrawal, 2022)

$$Precision@k = \frac{|\{Recom - and - Relev - Items\}|}{|\{Recom - Items\}|}$$

Equation 3: The Formula of Recall (Airen & Agrawal, 2022)

$$Recall@k = \frac{|\{Recom - and - Relev - Items\}|}{|\{Relev - Items\}|}$$

Airen and Agrawal define k as user defined parameter that refers to relevancy threshold for the rating of an item. An item is considered as relevant if its real rating is greater than k, or else the item is irrelevant. The relevancy of the item is known before the recommendation is generated. According to Airen and Agrawal:

- "Relev-Items" refers to the items that are relevant
- "Recom-Items" is the items which are recommended
- "Recom-and-Relev-Items" denotes the items which are recommended and relevant. "|| ||" indicates the cardinality of set.

Mean Squared Error (MSE)

Mean Squared Error (MSE) measures the average squared difference between the actual and predicted values. The calculation is parameter-free and inexpensive, requiring only one multiply and two additions per sample (Wang & Bovik, 2009). As well as being memoryless, the squared error can be evaluated for each sample independently of the others. It is a metric that

is desirable in the context of statistics and estimation. Its formation was introduced by C. F. Gauss, by whom the convenience of the MSE was noted aside from its arbitrary nature in relation to actual loss in applications (Casella & Lehmann, 1999).

Equation 4: Formula of Mean Squared Error (MSE) (Airen & Agrawal, 2022)

$$MSE = \frac{\sum_{\widehat{r_{ui}} \in \hat{R}} (r_{ui} - \widehat{r_{ui}})^2}{|\hat{R}|}$$

where:

- r_{ui} is the actual rating given by the user u for item i
- $\widehat{r_{ui}}$ is the predicted rating for item i by user u
- $\hat{R}$ is the set of all predicted ratings $\widehat{r_{ui}}$

Mean Absolute Error (MAE)

Mean absolute error (MAE) is quite similar to mean squared error (MSE), but it takes the absolute values into account. It measures the absolute difference between the actual and predicted values. Mean absolute error (MAE) calculates the dataset's total alteration mean based on the alteration between the original values and predictable values (Rajawat, et al., 2022). It is a popular metric because like Root Mean Square Error (RMSE), which is rooted mean squared error (MSE), this metric uses error values that are matched to predicted target values (Schneider & Xhafa, 2022).

Equation 5: Formula of Mean Absolute Error (MAE) (Airen & Agrawal, 2022)

$$MAE = \frac{\sum_{\widehat{r_{ui}} \in \hat{R}} |r_{ui} - \widehat{r_{ui}}|}{|\hat{R}|}$$

The letters in the equation above denote the same meanings in the mean squared error (MSE).

Design

The system first needs a dataset to work on. It is one of the most important requirements. This dataset should contain movie names, descriptions, genre, ratings, etc. The datasets can be downloaded from a website called Kaggle which has a plenty number of The Movie Database (TMDB) and the Internet Movie Database (IMDB) datasets. We have two different databases, one containing the index and names of movies and ratings while the other has the index of movies and users, and the values of ratings given by the users to the movies. The database then needs to be preprocessed so it should be clean and interpretable. The datasets had unnecessary columns, so they were dropped. There was not any other problem besides those. The dataset had genres as another feature, but ratings were preferred to be used. So, the movie names and the index were extracted from the first dataset and the movie index, user index, and ratings were extracted from the other one. The ratings dataset was processed again so it had movie id as rows, user id as columns, and ratings as the values. There were lots of null values because the users might not have rated all the movies,

they were replaced with 0. It was then converted to a sparse matrix. The example can be shown below:

Table 4: An example of a sparse matrix

userId movieId	1	2	3
1	3.5	5.0	3.0
2	4.0	1.5	4.5
3	2.0	2.5	4.0

After that, it was time to implement the KNN algorithm with cosine similarity. It was implemented using python with the cosine metric and the “k” value. A function was created for the main part, the recommendation process. The method was designed in a way that the user calls the name of the function and then enters the name of the movie from the movie dataset, the first one, the matrix created and a desired number of recommended movies to see. In the function, the index and the name of the selected movie were defined. Then, the name of the movie is printed as “Movie selected: ”, index is printed as the same, “Index: ”. The KNN algorithm implemented was applied in the function, the number of the movies was defined as the “k” value so the system searches for the similar movies using the ratings in the sparse matrix and the nearest neighbors defined

initially, and then returns the number of movies entered. If 10 is entered as the number, 10 different movies that has the similar ratings to the selected movie will come out with their indices.

Implementation

Two input datasets are used to implement the system. One has the movies with their IDs while the other one has the IDs of the movies rated and users that rated the movies, and ratings.

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)

Figure 6: Movies.csv

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

Figure 7: Ratings.csv

The system to be implemented is similar to what Gupta, et al. did in 2020. Python is used to implement the system. Some libraries are imported to read the datasets. There are at least 20 ratings available for each user. Pandas, scipy, sklearn, and fuzzywuzzy are used as libraries and they all brought efficient results on Jupyter Notebook where python was implemented. Each rating for a movie differs from 1 to 5 where 5 happens to be the highest value and 1 is the lowest.

```
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import numpy as np
from fuzzywuzzy import process
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Figure 12: Libraries used

Since the database is very big, there might be movies that are not still rated or rated by only one. Hence, a sparse pivot matrix is created and the data of the second dataset, ratings, are used. They are also implemented by KNN and filling null values, or O's, in missing information fields (Sarwar, et al., 2001).

The ratings datasets is pivoted to create a table where movieId is the row, or index, userId is the column and ratings are the values. The pivoted table looks like this below:

```
user_ratings = ratings.pivot(index = 'movieId', columns = 'userId', values = 'rating')
user_ratings
```

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN

```
user_ratings = ratings.pivot(index = 'movieId', columns = 'userId', values = 'rating').fillna(0)
user_ratings
```

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
193583	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
193585	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
193587	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
193609	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

9724 rows × 610 columns

Figure 14: The pivoted matrix with the NaN values replaced by 0.

These NaN values are replaced with 0 by the command ".fillna(0)" at the end of the code above the table. The new look of the table is like the one below:

After this step, a sparse matrix is created using the values of the pivoted table above and it is used for the KNN algorithm model created to fit in. The matrix is used to split the data into train and test data at the percentage of 50. The reason the test size

Figure 15: The creation of the matrix, data splitting and algorithm model and how it is fitted in the matrix

is 50 is more availability of movies to predict. For instance, there are 9724 movies in total. However, only the half of them would be available to be used for prediction. If the test size is to be 0.2, only 20% of the movies were available. 0.5 means many more movies could be used to test the model. The algorithm is created with the “cosine” metric, since cosine similarity is going to be used, the algorithm type is brute, so every single cell is going to be observed in the matrix, and the “k” value is 20 at this step.

The pivot matrix is created by “csr_matrix” imported from “scipy.sparse” library and the model is initiated by “NearestNeighbors” imported from the library “sklearn.neighbors”. The data was split with the help of “train_test_split” property imported from “sklearn.model_selection” library

The KNN observes the pivot matrix created and cosine similarity is used for calculating the similarity with the movie searched based on the ratings. The search process is done with the help of fuzzywuzzy in which a movie name can be typed from the first dataset and the matrix data is used to recommend movies. The library needs to be installed from the terminal, or command prompt by the command “pip install fuzzywuzzy” in order to be able to make use of it. “process” is imported from the library.

A python function is created for the recommendation process. Three different parameters are used, namely, movie name, the dataset, and the number of similar movies. Movie name is the desired name of the movie, the dataset is the pivot matrix

```
def recommender(movie_name, data, n):

    result = process.extractOne(movie_name, movies['title'])
    index = result[2]
    selected_movie_name = movies['title'][index]

    print("Movie selected:", selected_movie_name)
    print("Index:", index)
    print("Searching for recommendation...")

    if index >= data.shape[0]:
        print("Index out of range.")
        return

    distance, indices = model.kneighbors(data[index], n_neighbors = n)
    for i in indices:
            print(movies['title'][i])
```

Figure 16: The python function of the recommendation process

created. When these three parameters are entered, the number of similar movies is printed out with the one written by the user.

Index is a variable defined to extract the index of the movie name written and the selected movie name is the movie name written. The movie name and the index are printed and the KNN model created is used during the searching process. The data is used to predict the similar movies and the “k” value is defined as the number of to movies the user wants to see. So, the system looks for 20 nearest ratings and picks the “n” number of movies with the similar ratings to the movie name written based on the cosine similarity. The movies with their index numbers are print out at the final step.

Testing and Integration

```
recommender('Iron Man', matrix, 10)
```

```
Movie selected: Iron Man (2008)
Index: 6743
Searching for recommendation...
6743                                  Iron Man (2008)
7197                                    Garage (2007)
7195                          Merry Madagascar (2009)
7354                               A-Team, The (2010)
6726                           Superhero Movie (2008)
7137                           Thirst (Bakjwi) (2009)
7026                                    Scorpio (1973)
7571                                    Win Win (2011)
3880                    Look Who's Talking Now (1993)
6388    After the Wedding (Efter brylluppet) (2006)
```

Figure 17: An example of how the system works.

Figure 15 above shows that the movie name is “Iron Man”, the dataset is the matrix and “n” is 10. The system actually brings out 9 different movies and the movie written with the total number of 10. The years are included in movie names, but they do not have to be written, the system recognizes the name already.

The system does not return any error when entering a wrong movie name. Instead, even if the selected movie name is left blank, it picks a random movie and keeps processing.

```
recommender(' ', matrix, 10)
```

```
Movie selected: Toy Story (1995)
Index: 0
Searching for recommendation...
0                                        Toy Story (1995)
2353                                   'night Mother (1986)
418                                    Jurassic Park (1993)
615                  Independence Day (a.k.a. ID4) (1996)
224             Star Wars: Episode IV - A New Hope (1977)
314                                     Forrest Gump (1994)
322                                   Lion King, The (1994)
910     Once Upon a Time in the West (C'era una volta ...
546                             Mission: Impossible (1996)
963                                             Diva (1981)
```

Figure 18: The system still recommends movies even if the desired movie is left empty.

As it can be observed from Figure 16 above, no movie was entered in the function; however, the system took “Toy Story (1995)” as the desired movie and recommended 10 movies that has the similar ratings to that of “Toy Story (1995)”.

When some random name is entered, the system picks the movie that has the similar letters to the name entered.

```
recommender('ttttttt', matrix, 10)
```

```
Movie selected: Itty Bitty Titty Committee (2007)
Index: 6615
Searching for recommendation...
6615          Itty Bitty Titty Committee (2007)
6862                        City of Ember (2008)
6953                    Battle in Seattle (2007)
6761                               Recount (2008)
6658                            27 Dresses (2008)
7046                          Sweeney Todd (2006)
7820    Journey 2: The Mysterious Island (2012)
6460     How the Grinch Stole Christmas! (1966)
3211                    Sand Pebbles, The (1966)
6620                               Beowulf (2007)
```

Figure 19: A similar movie is picked when random letters are entered.

Figure 17 indicates that when “ttttttt” is entered, the system picks the movie “Itty Bitty Titty Committee (2007)” thinking that this movie is the most similar to what is wanted because this movie name has multiple number of “t” letters. The system then performs the rest of the operation required.

The movie dataset has 9742 rows and 2 columns while ratings dataset has 100836 rows and 3 columns, due to the fact that there are users rating more than one movie. The ratings dataset is pivoted in a way that the shape of it fits a sparse matrix in which the ratings of each user to each movie are seen. This makes 9724 columns and 610 columns. There are movies that goes beyond 9724. When that kind of movies are written, the system returns error.

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)
...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)
9738	193583	No Game No Life: Zero (2017)
9739	193585	Flint (2017)
9740	193587	Bungo Stray Dogs: Dead Apple (2018)
9741	193609	Andrew Dice Clay: Dice Rules (1991)

Figure 20: The whole "movies.csv" dataset

When looking at Figure 18 above, the last 5 movies are beyond the shape of the matrix. When one of these movies is written,

the system stops the operation and does not recommend any movie, instead, it returns “Index out of range” error. It can be understood that these movies might not have even been rated at all when observing Figure 12.

```
recommender('Flint (2017)', matrix, 10)

Movie selected: Flint (2017)
Index: 9739
Searching for recommendation...
Index out of range.
```

Figure 21: The system throws “Index out of range” error.

For instance, the index of the move “Flint (2017)” 9739 whereas the shape of the matrix was 9724 which means that it goes beyond the matrix shape. The system throws “Index out of range” error when it sees that name since it is out of the matrix dataset.

Product Evaluation

To evaluate the system, mean squared error (MSE), mean absolute error (MAE), the precision and recall scores were used. The recommender function was modified, some functions were added, and some movies were tested.

```
def recommender(movie_name, data, n, relevant_threshold):

    result = process.extractOne(movie_name, movies['title'])
    index = result[2]
    selected_movie_name = movies['title'][index]

    print("Movie selected:", selected_movie_name)
    print("Index:", index)
    print("Searching for recommendation...")

    if index >= data.shape[0]:
        print("Index out of range.")
        return

    distance, indices = model.kneighbors(data[index], n_neighbors = n)
    for i in indices:
            print(movies['title'][i])

    # Calculate relevance of recommendations
    relevant_mask = (ratings['movieId'].isin(indices.flatten())) & (ratings['rating'] >= relevant_threshold)

    # Calculate precision and recall
    relevant_count = relevant_mask.sum()
    precision = relevant_count / n
    total_relevant = len(ratings[ratings['movieId'] == index])
    recall = relevant_count / total_relevant


    print("Precision:", precision)
    print("Recall:", recall)

    # Calculate MSE and RMSE
    mse = mean_squared_error([0]*n, distance[0])
    mae = np.mean(np.abs([0]*n - distance[0]))

    print("Mean Squared Error (MSE):", mse)
    print("Mean Absolute Error (MAE):", mae)
```

Figure 22: The recommender function with the evaluation metrics used.

Mean squared error (MSE) and mean absolute error (MAE) were used to evaluate the performance of the predictions made. Mean squared error (MSE) calculates the squared error between the actual values and predicted results while mean absolute error (MAE) measures the absolute difference between the actual values and predicted results. Mean squared error (MSE) is for the magnitude of the whole error the system made while predicting. On the other hand, mean absolute error (MAE) has a straightforward calculation of the prediction accuracy, representing the average error magnitude avoiding focusing on

larger errors inordinately. However, they are both good for comprehending the prediction characteristics of the model, giving an understandable view of its performance. Both MSE and MAE need to be small to ensure the model performs well. If their values are small, the performance of the models are good whereas bad if they are large. The higher the MSE and MAE are, the worse the model performance is. “mean_squared_error” and “mean_absolute_error” were imported from “sklearn.metrics”.

The relevance of the movies plays a significant role in the calculation of the precision and recall scores. “relevant_mask” variable was created to take the movie ratings that are greater than or equal to the defined relevant threshold with its movieId. Relevant threshold is an important parameter in this system since it is used for definition of a minimum rating value and thus decision of the relevance of the movie name.

“relevant_count” variable was defined to take the sum of the “relevant_mask” and it is used in the calculation of both precision and recall scores. The division of the variable by “n” results in precision. For recall, the “relevant_count” variable is divided by the finding of the total number of ratings for the movie name. In other words, precision calculates the proportion of recommended movies relevant to the user whereas recall computes the proportion of relative items which are successfully recommended.

The relevant threshold and number of movies desired to be printed out, “n”, are the main variables for precision and recall.

They play a key role in the values of those scores. The default value of the relevant threshold is 4.5, but it can be changed.

The values of precision and recall scores range from 0 to 1. When these values are closer to 1, the value of relevance increases while it decreases when the values are closer to 0.

```
recommender('matilda', X_test_csr, 2, 4.5)
```

```
Movie selected: Matilda (1996)
Index: 649
Searching for recommendation...
462                Scout, The (1994)
2450    White Men Can't Jump (1992)
Name: title, dtype: object
Precision: 0.5
Recall: 1.0
Mean Squared Error (MSE): 0.26823439047038394
Mean Absolute Error (MAE): 0.5176125545382146
```

Figure 23: An example of a movie with precision and recall, mean squared error (MSE) and mean absolute error (MAE).

For example, in Figure 21, the movie “Matilda (2017)” has the precision of 0.5 and the recall of 1 when the relevant threshold is set to 4.5 and “n” is 2. This means that 50% of the movies recommended are relevant to the user and the system had a success rate of 100% in recommending the relevant items. This movie had a fairly good rate for these factors. 2 was used for “n” because the best possible outcomes of precision and recall scores came at this value. The MSE and MAE are 0.27 and 0.52 respectively which indicates that the predictions are reasonably accurate. The test data created in when data got splitted, “X_test_csr”, was used as the dataset to recommend movies from for all the movie names.

However, not all the movies have the same value. Like told earlier, precision and recall scores depend on the relevant threshold and “n”.

```
recommender('matrix', X_test_csr, 5, 4.5)
```

```
Movie selected: Matrix, The (1999)
Index: 1939
Searching for recommendation...
649                    Matilda (1996)
4626              Brother Bear (2003)
1345               Chinese Box (1997)
4347                  Breakin' (1984)
287     Star Trek: Generations (1994)
Name: title, dtype: object
Precision: 1.0
Recall: 0.625
Mean Squared Error (MSE): 0.0
Mean Absolute Error (MAE): 0.0
```

Figure 24: Another example.

As shown in Figure 23, the movie “The Matrix (1999)” has the precision and recall scores of 1 and 0.625 respectively when the relevant threshold is 4.5 and n is 5. This means that 100% of the movies recommended are relevant to the user and the system was approximately 63% successful in recommending the relevant items when the relevant threshold was 4.5 and n was 5. This movie had a high rate for these factors. Both MSE and MAE are 0 which show that the model made no error at all when predicting the movies that are similar to “The Matrix (1999)”.

The system unfortunately has some weaknesses. Even though the movies above had good scores, some other movies may have erroneous score values.

```
In [21]: recommender('godfather', X_test_csr, 5, 4)

         Movie selected: Godfather, The (1972)
         Index: 659
         Searching for recommendation...
         4858        Barbershop 2: Back in Business (2004)
         848     Eighth Day, The (Huitième jour, Le) (1996)
         3404                      No Holds Barred (1989)
         4399                             Jubilee (1977)
         4527           Once Upon a Time in Mexico (2003)
         Name: title, dtype: object
         Precision: 0.6
         Recall: inf
         Mean Squared Error (MSE): 0.11082838451904206
         Mean Absolute Error (MAE): 0.33289103915962087
```

Figure 25: An example with an erroneous value.

Figure 24 shows that the MSE is 0.11 and MAE is 0.33. These indicate a good prediction performance; however, the precision of “The Godfather (1972)” is 0.6, but the recall is infinite, “inf” when the relevant threshold is 4. The precision is reasonably good, but the recall is suspicious. The reason is it shows that the movie is recommended successfully; however, this might be because there was no relevant movie found in the dataset for “The Godfather (1972)” in terms of the rating values. It seems like the number of relevant ratings is 0 so infinite value was returned. It is still really important to set the relevant threshold and “n” variable.

Another issue is that the test data created was used for testing. Since it is half of the whole dataset, 4862 out of 9724 data, the

indices of the movies beyond 4862 cannot be used for testing. Otherwise, it gives “Index out of range” error.

```
recommender('iron man', X_test_csr, 5, 4.5)
```

```
Movie selected: Iron Man (2008)
Index: 6743
Searching for recommendation...
Index out of range.
```

Figure 26: Index out of range error

As it can be observed from Figure 25, the movie “Iron Man” has the index of 6743 and it is out of the test data of 4862 movies. The system therefore gives “Index out of range” error.

Closing chapters

The system has been designed to meet the requirements of a movie recommendation system that advises movies to users based on ratings of movies. The report involves an explanation of the system design and implementation, including the algorithms and methods used. K Nearest Neighbor (KNN) was chosen as the key algorithm because it was thought to be the most appropriate considering how related it is to machine learning and the filtering methods. The capabilities and weaknesses of the system were discussed based on the results in testing. The system implemented is in the simplest form in terms of the design and the function. More can be done to improve the system with the involvement of front-end development that contains graphical user interface (GUI) and application programming interface (API) so a movie website that uses such recommendation system like this can be created in the future. All in all, this report provides an outline of the performance of the project including the success and lacks of the system.

Appendices

Data Collection and Preprocessing

The whole code was written using Python programming language. Two datasets were used to implement the recommender system. One of them includes the movie names while the other one contains the movie and user IDs as well as the ratings. These are both IMDB datasets. These datasets were imported using Pandas library in Python.

The link of the datasets: https://github.com/Praful2000/YoutubeLectures/tree/master/Movie%20KNN

From those datasets, some unnecessary columns had to be dropped. The first dataset had three columns: “movieId”, “title”, and “genres”. The column “genres” was removed because it was of no use. The other dataset had “userId”, “movieId”, “rating” and “timestamp”. The unnecessary column “timestamp” was dropped and it was pivoted, so “movieId” was the row whereas “userId” was the column and “rating” were the values, in order to create a matrix to the fit the K Nearest Neighbor algorithm in.

Code snippets:

```
import pandas as pd # imports the Pandas library to load the datasets

movies_dataset = pd.read_csv("movies.csv") # imports the movies dataset
```

```
movies = movies_dataset.drop(columns=['genres']) #
drops the column 'genres'

ratings_dataset = pd.read_csv("ratings.csv") #
imports the ratings dataset


ratings = ratings_dataset.drop(columns=['timestamp'])
# drops the column 'timestamp'


user_ratings = ratings.pivot(index = 'movieId',
columns = 'userId', values = 'rating').fillna(0) #
pivots the ratings dataset (fillna(0)) replaces the
NaN values with 0)
```

The movies dataset had 9742 rows, ratings dataset had 100836 rows and one user rated more than one movie. Hence, it can be easy to count the movies; however, it could take time to count the users and ratings since some movies did not get rated.

Model Development

K Nearest Neighbour (KNN) algorithm was used to implement the model. K Nearest Neighbour (KNN) is an algorithm that makes predictions using the neighbours around it. It specifies the number of neighbours to use as “k”. Three methods are used for filtering for recommendation: collaborative filtering, content-based filtering, and hybrid filtering. Collaborative filtering is more likely to use ratings to filter the movies while content-based

filtering, as the name implies, focuses on the content of movies, like genres, and hybrid filtering involves the combination of these two techniques. Collaborative filtering method was chosen to implement the model. The similar movies are recommended using cosine similarity. It is very useful because it plays a key role in finding similar movies.

$$CosSim(x, y) = \frac{\sum_i x_i \, y_i}{\sqrt{\sum_i x_i^2} \, \sqrt{\sum_i y_i^2}}$$

Equation 5: The Cosine similarity formula which is used for the recommendation of movies.

Where:

- x and y are the vectors
- $\sum_i x_i \, y_i$ is the dot product of the vectors x and y.
- $\sqrt{\sum_i x_i^2}$ and $\sqrt{\sum_i y_i^2}$ are the magnitudes of the vectors x and y.

The vectors x and y represent the movies' ratings and the magnitudes represent the summation of the squared ratings. The cosine similarity between two movies' ratings is calculated. Movies with higher cosine similarity values are more similar. For example, there are 3 users and 4 movies, and the ratings are:

Movie 1: [5, 4, 1]

Movie 2: [3, 0, 1]

Movie 3: [0, 0, 0]

Movie 4: [1, 1, 5]

The cosine similarity between Movie 1 and Movie 2 are calculated as follows:

$$CosSim(1,2) = \frac{(5*3)+(4*0)+(1*1)}{\sqrt{5^2+4^2+1^2}*\sqrt{3^2+0^2+1^2}}$$

$$= \frac{16}{\sqrt{42}*\sqrt{10}}$$

$$\approx 0.781$$

Since 0.781 is closer to 1, it can be said that the cosine similarity is high. In this case, Movie 2 is similar to Movie 1 and Movie 4 can be recommended to users. This process is called item-item collaborative filtering.

After the ratings dataset was pivoted. It was used to create a CSR matrix. The matrix created was split into training and test data at 50%. So, the half of the dataset was used for training while the other half was used for testing. Then the KNN model was created and fit to the training data created. After that a function that involves the movies and ratings dataset, the fitted model, and the evaluation metrics was created.

Code snippet:

```
matrix = csr_matrix(user_ratings.values)

X_train, X_test = train_test_split(matrix,
test_size=0.5, random_state=0) # Convert back to CSR
if needed for certain operations

X_train_csr = csr_matrix(X_train)

X_test_csr = csr_matrix(X_test)
```

```
model = NearestNeighbors(metric = 'cosine', algorithm
= 'brute', n_neighbors = 20)
model.fit(X_train_csr)

def recommender(movie_name, data, n,
relevant_threshold):

    result = process.extractOne(movie_name,
movies['title'])
    index = result[2]
    selected_movie_name = movies['title'][index]

    print("Movie selected:", selected_movie_name)
    print("Index:", index)
    print("Searching for recommendation...")

    if index >= data.shape[0]:
        print("Index out of range.")
        return

    distance, indices = model.kneighbors(data[index],
n_neighbors = n)
```

```
    for i in indices:
            print(movies['title'][i])

    # Calculate relevance of recommendations
    relevant_mask =
(ratings['movieId'].isin(indices.flatten())) &
(ratings['rating'] >= relevant_threshold)

    # Calculate precision and recall
    relevant_count = relevant_mask.sum()
    precision = relevant_count / n
    total_relevant = len(ratings[ratings['movieId']
== index])
    recall = relevant_count / total_relevant


    print("Precision:", precision)
    print("Recall:", recall)

    # Calculate MSE and MAE
    mse = mean_squared_error([0]*n, distance[0])
    mae = np.mean(np.abs([0]*n - distance[0]))
```

```
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
```

Precision, recall scores, mean squared error (MSE), and mean absolute error (MAE) were used as evaluation metrics. The formula of each metric is as follows:

$$Precision = \frac{Number\ of\ Relevant\ Movies\ Recommended}{Total\ Number\ of\ Movies\ Recommended}$$

$$Recall = \frac{Number\ of\ Relevant\ Movies\ Recommended}{Total\ Number\ of\ Relvant\ Movies}$$

$$MSE = \frac{\sum(Actual\ Values - Pred\quad Values)^2}{Total\ Number\ of\ Values}$$

$$MAE = \frac{\sum|Actual\ Values - Predict\quad Values|}{Total\ Number\ of\ Values}$$

Testing and Validation

Some tests were made using the test data created. The system implemented brought out different similar movies recommended at a desired number of n. Precision, recall scores, MSE, and MAE results were given. Here is the table including the movie names and the results:

Movie Names	Similar Movies	Relevant Threshold	Precision Score	Recall Score	Mean Squared Error (MSE)	Mean Absolute Error (MAE)
Matilda (1996)	The Scout (1994) White Men Can't Jump (1992) (at n = 2)	4.5	0.5	1.0	0.27	0.52
The Matrix (1999)	Matilda (1996) Brother Bear (2003) Chinese Box (1997) Breakin' (1984) Star Trek: Generations (1994) (at n = 5)	4.5	1.0	0.625	0	0

The Godfather (1972)	Barbershop 2: Back in Business (2004) The Eight Day (Le Huitieme Jour) (1996) No Holds Barred (1989) Jubilee (1977) Once Upon a Time in Mexico (2003) (at n = 5)	4.5	0.6	Infinite (error)	0.11	0.33
Iron Man (2008)	Index out of range.	-	-	-	-	-

Not all the movies can be said to work perfectly. Some movies gave erroneous precision and recall scores, and some movies

gave error because of the fact that their indices not included in the test data created.

Complete Source Code

```
# necessary libraries for the model
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.neighbors import NearestNeighbors
import numpy as np
from fuzzywuzzy import process
from sklearn.metrics import mean_squared_error,
mean_absolute_error

# import the movies dataset
movies_dataset = pd.read_csv("movies.csv")
movies_dataset.head()

# remove unnecessary columns
movies = movies_dataset.drop(columns=['genres'])
movies
```

```
# number of rows and columns in movies dataset
movies.shape

# remove unnecessary columns
ratings_dataset = pd.read_csv("ratings.csv")
ratings_dataset.head()

# number of rows and columns in ratings dataset
ratings = ratings_dataset.drop(columns=['timestamp'])
ratings.head()
ratings.shape

# transform the ratings dataset into matrix
user_ratings = ratings.pivot(index = 'movieId',
columns = 'userId', values = 'rating').fillna(0)
user_ratings

# create a csr matrix from the transformed dataset
matrix = csr_matrix(user_ratings.values)
matrix
```

```
# split the data into train and test at 50%
X_train, X_test = train_test_split(matrix,
test_size=0.5, random_state=0)


# Convert back to CSR if needed for certain
operations
X_train_csr = csr_matrix(X_train)
X_test_csr = csr_matrix(X_test)



print(f"Training data shape: {X_train_csr.shape}") #
number of rows and columns in the train data
print(f"Test data shape: {X_test_csr.shape}") #
number of rows and columns in the test data


# create KNN model for the system
model = NearestNeighbors(metric = 'cosine', algorithm
= 'brute', n_neighbors = 20)
# fit the model to the train data
model.fit(X_train_csr)
```

```
# the recommender system function

def recommender(movie_name, data, n,
relevant_threshold):


    result = process.extractOne(movie_name,
movies['title'])

    index = result[2]

    selected_movie_name = movies['title'][index]


    print("Movie selected:", selected_movie_name)

    print("Index:", index)

    print("Searching for recommendation...")


    if index >= data.shape[0]:

        print("Index out of range.")

        return


    distance, indices = model.kneighbors(data[index],
n_neighbors = n)

    for i in indices:

            print(movies['title'][i])
```

```
    # Calculate relevance of recommendations
    relevant_mask =
(ratings['movieId'].isin(indices.flatten())) &
(ratings['rating'] >= relevant_threshold)

    # Calculate precision and recall
    relevant_count = relevant_mask.sum()
    precision = relevant_count / n
    total_relevant = len(ratings[ratings['movieId']
== index])
    recall = relevant_count / total_relevant


    print("Precision:", precision)
    print("Recall:", recall)

    # Calculate MSE and MAE
    mse = mean_squared_error([0]*n, distance[0])
    mae = np.mean(np.abs([0]*n - distance[0]))

    print("Mean Squared Error (MSE):", mse)
    print("Mean Absolute Error (MAE):", mae)
```

```
# some tests made

recommender('matilda', X_test_csr, 2, 4.5)

recommender('matrix', X_test_csr, 5, 4.5)

recommender('godfather', X_test_csr, 5, 4.5)

recommender('iron man', X_test_csr, 5, 4.5)
```

References

Adeniyi, D. A., Wei, Z. & Yongquan, Y., 2014. Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method. *Saudi Computer Society,* October, 12(1), pp. 90-108.

Adomavicius , G. & Tuzhilin , A., 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Trans Knowl Data Eng,* 17(6), p. 734–749.

Ahuja, R., Solanki, A. & Nayyar, A., 2019. Movie Recommender System Using K-Means Clustering AND K-Nearest Neighbor. *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence),* pp. 263-268.

Airen, S. & Agrawal, J., 2022. Movie Recommender System Using K-Nearest Neighbors Variants. *Natl. Acad. Sci. Lett.,* Volume 45, p. 75–82.

Al Mamunur, R., Karypis , G., Riedl , J. & Shyong , K. L., 2006. Clustknn: a highly scalable hybrid model- & memory-based cf algorithm. *Proceeding of webKDD.*

Albadvi & Shahbazi, M., 2009. A hybrid recommendation technique based on product category attributes. *Expert Systems with Applications,* 36(9), p. 11 480–11 488.

Ansari, A., Essegaier, S. & Kohli, R., 2000. Internet recommendation systems. *Journal of Marketing Research,* 37(3), p. 363–375.

Ayub , M. et al., 2019. Modeling user rating preference behavior to improve the performance of the collaborative filtering based recommender systems. *PLoS ONE,* 14(8), p. e0220129.

Bahadorpour , M., Neysiani , B. S. & Shahraki , M. N., 2017. Determining optimal number of neighbors in item-based knn collaborative filtering algorithm for learning preferences of new users. *J Telecommun Electron Comput Eng (JTEC),* 9(3), p. 163–167.

Bahadorpour, M., Neysiani , B. S. & Shahraki, M. N., 2017. Determining Optimal Number of Neighbors in Item-based kNN Collaborative Filtering Algorithm for Learning Preferences of New Users. *Journal of Telecommunication Electronic and Computer Engineering (JTEC),* 9(3), pp. 163-167.

Beheshti, A. et al., 2020. Towards cognitive recommender systems. *Algorithms,* 13(8), p. 176.

Breese, J. S., Heckerman , D. & Kadie, C., 2013. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (UAI'98),* pp. 43-52.

Casella , G. & Lehmann, E. L., 1999. *Theory of Point Estimation.* New York: Springer-Verlag.

Cui, B. B., 2017. Design and implementation of movie recommendation system based on Knn collaborative filtering algorithm. *ITM web of conferences,* Volume 12, p. 04008.

Das, D., Chidananda , H. T. & Sahoo, L., 2018. Personalized Movie Recommendation System Using Twitter Data. *Progress in Computing Analytics and Networking,* pp. 339-347.

Furtado, F. & Singh, A., 2020. Movie recommendation system using machine learning. *International Journal of Research in Industrial Engineering,* 1(9), pp. 84-98.

Gupta, M. et al., 2020. Movie Recommender System Using Collaborative Filtering. *International Conference on Electronics and Sustainable Communication Systems (ICESC),* pp. 415-420.

Haenlein, M. & Kaplan, A., 2019. A brief history of artificial intelligence: on the past, present, and future of artificial intelligence. *California management review,* 4(61), pp. 5-14.

Halder , S., Sarkar , A. M. J. & Lee , Y. K., 2012. Movie recommendation system based on movie swarm. *2012 Second international conference on cloud and green computing,* p. 804–809.

Hande, R. et al., 2016. MOVIEMENDER-A movie recommender system. *International journal of engineering sciences & research technology (IJESRT),* 5(11), p. 686.

Hernandez, F. R. & Garcia, N. Y. G., 2016. Distributed processing using cosine similarity for mapping big data in Hadoop. *IEEE Latin America Transactions,* 14(6), p. 2857–2861.

Kataria , R. & Verma, O. P., 2016. An effective collaborative movie recommender system with cuckoo search. *Egyptian Informatics Journal,* July , 18(2), pp. 105-112.

Koren, Y., 2010. Factor in the neighbors: scalable and accurate collaborative filtering. *ACM Trans Knowl Discov Data (TKDD),* 4(1), p. 1–24.

Ko, S.-K.et al., 2011. A Smart Movie Recommendation System. *Lecture Notes in Computer Science,* Volume 6771, pp. 558-566.

Kumar, M., Yadav, D. K., Singh, A. & Gupta, V. K., 2015. A movie recommender system: Movrec. *International Journal of Computer Applications,* 3(124), pp. 7-11.

Lahitani, A. R., Permanasari , A. E. & Setiawan, N. A., 2016 . Cosine similarity to determine similarity measure: Study case in online essay assessment. *2016 4th lnternational Conferenceon Cyber and IT Service Management,* pp. 1-6.

Liang, W. et al., 2014. Difference factor' KNN collaborative filtering recommendation algorithm. *International Conference on Advanced Data Mining and Applications,* pp. 175-184.

Linden, G., Smith, B. & York, J., 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing,* 7(1), p. 76–80.

Lund , J. & Ng , Y. K., 2018. Movie recommendations using the deep learning approach. *2018 IEEE international conference on information reuse and integration (IRI),* p. 47–54.

Miller, B. N., Konstan, J. A. & Riedl, J., 2004. PocketLens: toward a personal recommender system. *ACM Transactions on Information Systems,* 22(3), pp. 437 - 476.

Mishra, N. et al., 2017. Solving sparsity problem in rating-based movie recommendation system. *Computational Intelligence in Data Mining: Proceedings of the International Conference on CIDM,* pp. 111-117.

Molina, L. E. & Bhulai, S., 2018. Recommendation System for Netflix. *Retrieved June,* Volume 6, p. 2021.

Muozorganero, M., Ramezgonzlez, G. A., Muozmerino, P. J. & Kloos, C. D., 2010. A collaborative recommender system based on space-time similarities. 9(3), p. 81–87.

Nassar, N., Jafar, A. & Rahhal, Y., 2020. A novel deep multi-criteria collaborative filtering model for recommendation system. *Knowledge-Based Systems 187,* p. 104811.

Nguyen, N. T. et al., 2007. Hybrid Filtering Methods Applied in Web-Based Movie Recommendation System. *Knowledge-Based Intelligent Information and Engineering Systems. KES 2007,* pp. 206-213.

Oyelade, O. J., Oladipupo, O. O. & Obagbuwa, I. C., 2010. Application of k means clustering algorithm for prediction of students' academic performance. *International Journal of Computer Science and Information Security,* 1(7), pp. 292-295.

Pathak, D., Matharia , S. & Murthy, C. N. S., 2013. ORBIT: Hybrid movie recommendation engine. *2013 IEEE International Conference ON Emerging Trends in Computing Communication and Nanotechnology (ICECCN),* pp. 19-24.

Phorasim , P. & Yu, L., 2017. Movies recommendation system using collaborative filtering and k-means. *International Journal of Advanced Computer Research,* 7(29), p. 52.

Rajawat, A. S., Mohammed, O., Shaw, R. N. & Ghosh, A., 2022. Chapter six - Renewable energy system for industrial internet of things model using fusion-AI,. *Applications of AI and IOT in Renewable Energy,* pp. 107-128.

Reddy, S. R. et al., 2019. Content-based movie recommendation system using genre correlation. *Smart Intelligent Computing and Applications,* pp. 391-397.

Resnick, P. et al., 1994. Grouplens: an open architecture for collaborative filtering of netnews. *Proceedings of the ACM Conference on Computer Supported Cooperative Work,* p. 175–186.

Sambhwani, D., 2020. *Movie Recommender System: Part 1.* [Online] Available at: https://towardsdatascience.com/movie-recommender-system-part-1-7f126d2f90e2

Sarwar, B., Karypis, G., Konstan , J. & Riedl, J., 2001. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web,* pp. 285-295.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J., 2001. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web. ACM,* p. 285–295.

Schafer , J. B., Frankowski , D., Herlocker , J. & Sen , S., 2007. Collaborative filtering recommender systems. *The adaptive web,* p. 291–324.

Schneider, P. & Xhafa, F., 2022. Chapter 3 - Anomaly detection: Concepts and methods. In: *Anomaly Detection and Complex Event Processing over IoT Data Streams.* s.l.:Academic Press, pp. 49-66.

Shani , G. & Gunawardana , A., 2011. Evaluating recommendation systems. *Recommender systems handbook,* p. 257–297.

Sharma, S., Rana, V. & Malhotra, M., 2022. Automatic recommendation system based on hybrid filtering algorithm. *Education and Information Technologies,* 27(2), pp. 1523--1538.

Singh, R. H. et al., 2020. Movie Recommendation System using Cosine Similarity and KNN. *International Journal of Engineering and Advanced Technology (IJEAT),* 9(5), pp. 556-558.

Surendran, A., Yadav, A. K. & Kumar, A., 2020. Movie Recommendation System using Machine Learning Algorithms. *International Research Journal of Engineering and Technology (IRJET),* 7(4), pp. 3694-3696.

Su, X. & Khoshgoftaar, T. M., 2009. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence,* Volume 2009, pp. 1-20.

Taunk, K., De, S., Verma, S. & Swetapadma, A., 2019. A Brief Review of Nearest Neighbor Algorithm for Learning and Classification. *2019 International Conference on Intelligent Computing and Control Systems (ICCS),* pp. 1255-1260.

Virk, H. K., Singh, E. M. & Singh, A., 2015. Analysis and design of hybrid online movie recommender system. *International journal of innovations in engineering and technology (IJIET),* 2(5), pp. 159-163.

Wang, G., 2012. *Survey of personalized recommendation system,* s.l.: s.n.

Wang, Z. & Bovik, A. C., 2009. Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Processing Magazine,* Volume 26, pp. 98-117.

Wei, D. & Junliang, C., 2013. The Bayesian Network and Trust Model Based Movie Recommendation System. *Intelligence Computation and Evolutionary Computation,* Volume 180, pp. 797-803.

Yasen, M. & Tedmori, S., 2019. Movies reviews sentiment analysis and classification. *Proceedings of the IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, JEEIT (2019),* pp. 860-865.