MOVIE RECOMMENDATION SYSTEM USING COLLABERATIVE FILTERING

➢ Overview: -

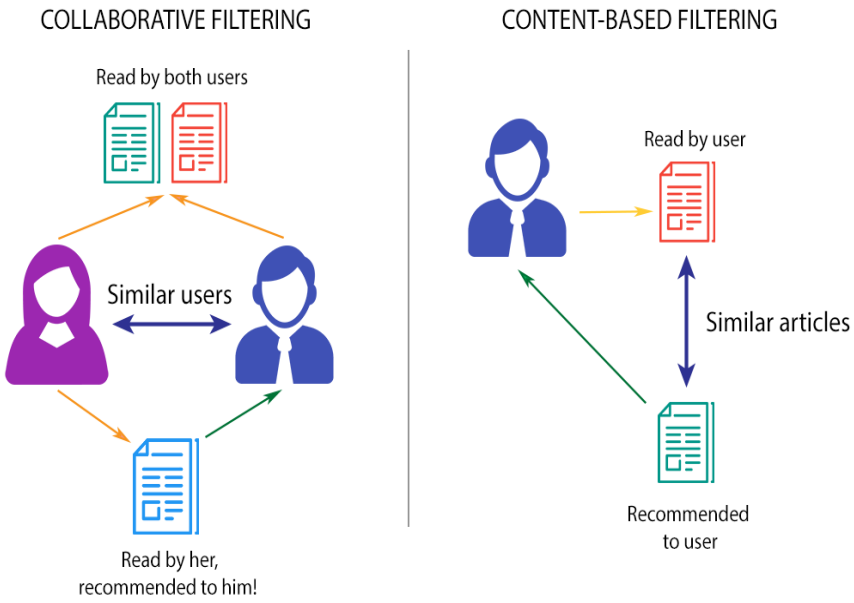
- In this project, We will build an item-based collaborative filtering system using MovieLens Datasets. Specially, we will train a KNN models to cluster similar movies based on user's ratings and make movie recommendation based on similarity score of previous rated movies.

➢ Recommender System:-

- Most internet products we use today are powered by recommender systems. YouTube, Netflix, Amazon, Pinterest, and long list of other internet products all rely on recommender systems to filter millions of contents and make personalized recommendations to their users.
- Recommender systems are well-studied and proven to provide tremendous values to interest business and their consumers.
- Recommender systems are the secret source for those multi-billion business, prototyping a recommender system can be very low cost and doesn't require a team of scientists.
- You can actually develop your own personalized recommender for yourself. It only takes some basic machine learning techniques and implementations in Python.

➢ Approaches:-

- Recommender systems can be loosely broken down into three categories: -
    1) Content based systems.
    2) Collaborative filtering systems.
    3) Hybrid systems (it uses combination of the other two).



- Content base approach: - It utilizes a series of discrete characteristics of an item in order to recommend additional items with similar properties.
- Collaborative filtering approach: - It builds a model from a user's past behaviors (items previously purchased or selected and numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items that the user may have an interest in.
- Hybrid approach: - It combines the previous two approaches. Most business probably use hybrid approaches in their production recommender systems.

- Here we use most common approach collaborative filtering. In Collaborative filtering based systems use the actions of users to recommend other items. In general, they can either be user based or item based. User based collaborating filtering uses the patterns of users similar to me to recommend a product (users like me also looked at these other items). Item based collaborative filtering uses the patterns of users who browsed the same item as me to

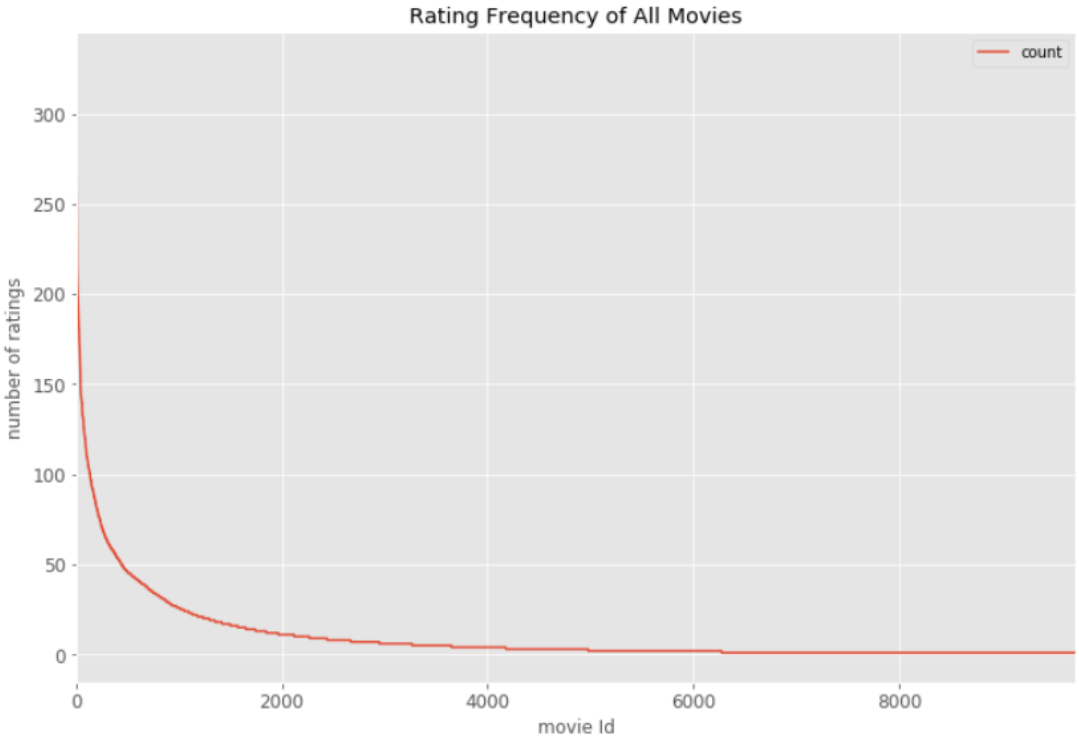
recommend me a product (users who looked at my item also looked at these other items). Item-based approach is usually preferred than user-based approach. User-based approach is often harder to scale because of the dynamic nature of users, whereas items usually don't change much, so item-based approach often can be computed offline.

➢ Data:-

- To build a movie recommender, we choose MovieLens Datasets.
- It contains 100,000 ratings and 3,600 tag application across 9000 movies.
- These data were created by 600 users between January 09,1995 and September 26,2018. The ratings are on a scale from 1 to 5.
- We have use two files ratings.csv and movies.csv.
- Ratings data provides the ratings of movies given by users.
- There are three fields in each row userId, movieId, rating.
- Each row can be seen as a record of interaction between a user and a movie.
- Movies data provides a movie title and genres for each movieId in Ratings data.

➢ Data Filtering:-

- In a real world setting, data collected from explicit feedbacks like movie ratings can be very sparse and data points are mostly collected from very popular items (movies) and highly engaged users. Large amount of less known items (movies) don’t have ratings at all. Let’s see plot the distribution of movie rating frequency.



- We can see that roughly 10,000 out of 53,889 movies are rated more than 100 times. More interestingly, roughly 20,000 out of 53,889 movies are rated less than only 10 times.

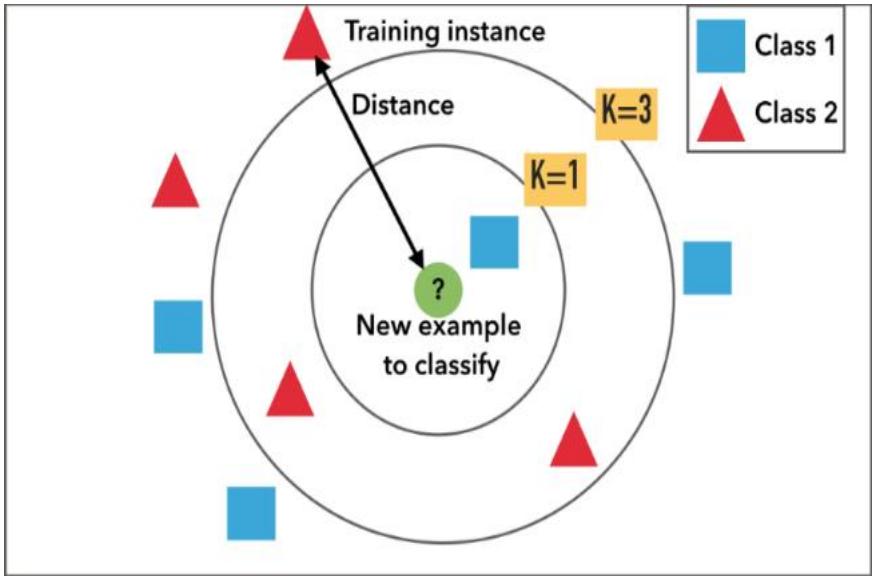
- Most models make recommendations based on user rating patterns. To remove noisy pattern and avoid "memory error" due to large datasets, we will filter our dataframe of ratings to only popular movies. After filtering, we are left with 10,000 movies in the Ratings data, which is enough for a recommendation model.

➢ Modeling:-

- Collaborative filtering systems use the actions of users to recommend other movies. In general they can either be user-based or item-based.
- Item base approach is usually preferred over user-based approach.User-based approach is often harder to scale because of the dynamic nature of users, whereas items usually don't change

much, and item based approach often can be computed offline and served without constantly re-training.

- To implement an item based collaborative filtering, KNN is a perfect go-to model and also a very good baseline for recommender system development.
- KNN is a non-parametric, lazy learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.
- KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity. When KNN makes inference about a movie, KNN will calculate the "distance" between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest movies as the most similar movie recommendations.



- We need to transform the dataframe of ratings into a proper format that can be consumed by a KNN model. We want the data to be in an m x n array, where m is the number of movies and n is the number of users.

- To reshape dataframe of ratings, we'll pivot the dataframe to the wide format with movies as rows and users as columns. Then we'll fill the missing observations with 0s since we're going to be performing linear algebra operations (calculating distances between vectors).
- Our dataframe of movie features is an extremely sparse matrix with a shape of 10,000*113,291. We definitely don't want to feed the entire data with mostly 0s in float32 datatype to KNN. For more efficient calculation and less memory footprint, we need to transform the values of the dataframe into a SciPy sparse matrix.

➢ K-Nearest Neighbors:-

- KNN falls in the supervisedlearning family of algorithms. Informally, this means that we are given a labelled dataset consisting of training observations (x,y)(x,y) and would like to capture the relationship between xx and yy. More formally, our goal is to learn a function h:X→Yh:X→Y so that given an unseen observation xx, h(x)h(x) can confidently predict the corresponding output yy.

- The KNN classifier is also a non parametric and instance-based learning algorithm.

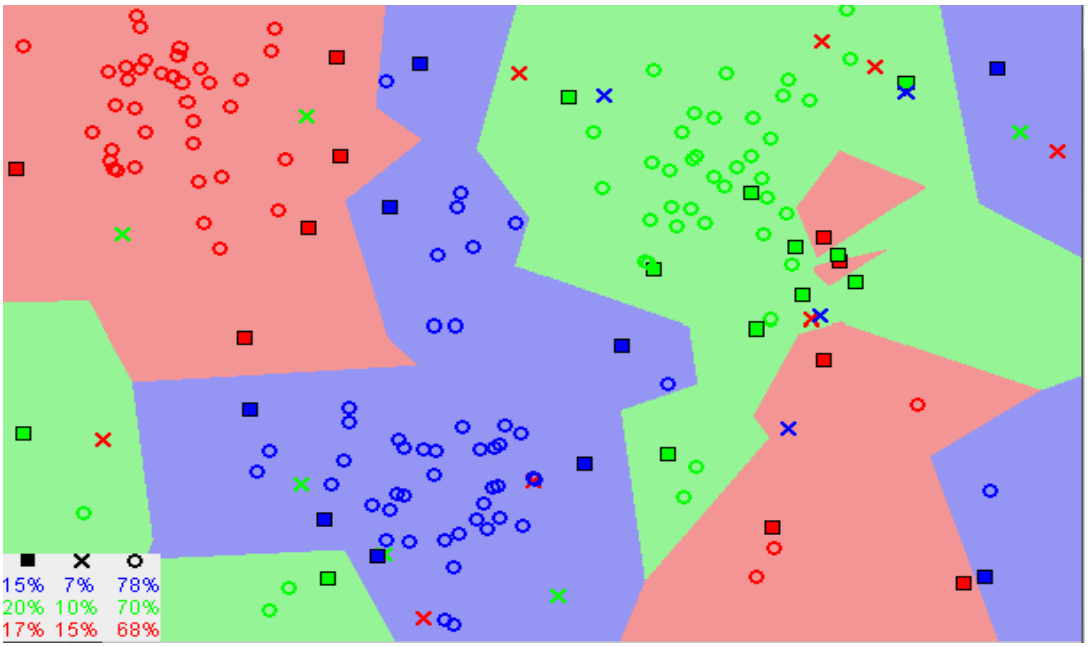
1) Non-parametric means it makes no explicit assumptions about the functional form of h, avoiding the dangers of mismodeling the underlying distribution of the data. For example, suppose our data is highly non-Gaussian but the learning model we choose assumes a Gaussian form. In that case, our algorithm would make extremely poor predictions.

2) Instance-based learning means that our algorithm doesn't explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as "knowledge" for the prediction phase. Concretely, this means that only when a query to our database is made (i.e. when we ask it to predict a label given an

input), will the algorithm use the training instances to spit out an answer.

- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

"Birds of a feather flock together."



- Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

- **Note:** An understanding of how we calculate the distance between points on a graph is necessary before moving on. If you are unfamiliar with or need a

refresher on how this calculation is done, thoroughly read "Distance Between 2 Points" in its entirety, and come right back.

- There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

- K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

1) **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
2) **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

➢ **Working of KNN Algorithm**

- In the classification setting, the K-nearest neighbour algorithm essentially boils down to forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance given by

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + \ldots + (x_n - x'_n)^2}$$

but other measures can be more suitable for a given setting and include the Manhattan, Chebyshev and Hamming distance.

- More formally, given a positive integer K, an unseen observation xx and a similarity metric dd, KNN classifier performs the following two steps:

1) It runs through the whole dataset computing dd between xx and each training observation. We'll call the K points in the training data that are closest to xx the set AA. Note that K is usually odd to prevent tie situations.
2) It then estimates the conditional probability for each class, that is, the fraction of points in AA with that given class label. (Note I(x)I(x) is the indicator function which evaluates to 11 when the argument xx is true and 00 otherwise)

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

Finally, our input xx gets assigned to the class with the largest probability.

- **Important**: KNN searches the memorized training observations for the K instances that most closely resemble the new instance and assigns to it the their most common class.

  - So, in simpler terms K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps

Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** − Now, based on the distance value, sort them in ascending order.
- **3.3** − Next, it will choose the top K rows from the sorted array.

- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

➢ Pros and Cons of KNN:-

Pros:-

- As you can already tell from the previous section, one of the most attractive features of the K-nearest neighbor algorithm is that is simple to understand and easy to implement.
- With zero to little training time, it can be a useful tool for off-the-bat analysis of some data set you are planning to run more complex algorithms on.
- Furthermore, KNN works just as easily with multiclass data sets whereas other algorithms are hardcoded for the binary setting.
- Finally, as we mentioned earlier, the non-parametric nature of KNN gives it an edge in certain settings where the data may be highly "unusual".

Cons:-

- One of the obvious drawbacks of the KNN algorithm is the computationally expensive testing phase which is impractical in industry settings.
- Note the rigid dichotomy between KNN and the more sophisticated Neural Network which has a lengthy training phase albeit a very fast testing phase.
- Furthermore, KNN can suffer from skewed class distributions. For example, if a certain class is very frequent in the training set, it will tend to dominate the majority voting of the new example (large number = more common).
- Finally, the accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.