**Recommendation System**

**Outline:**

**1. Data pre-processing**

**2. Feature Importance**

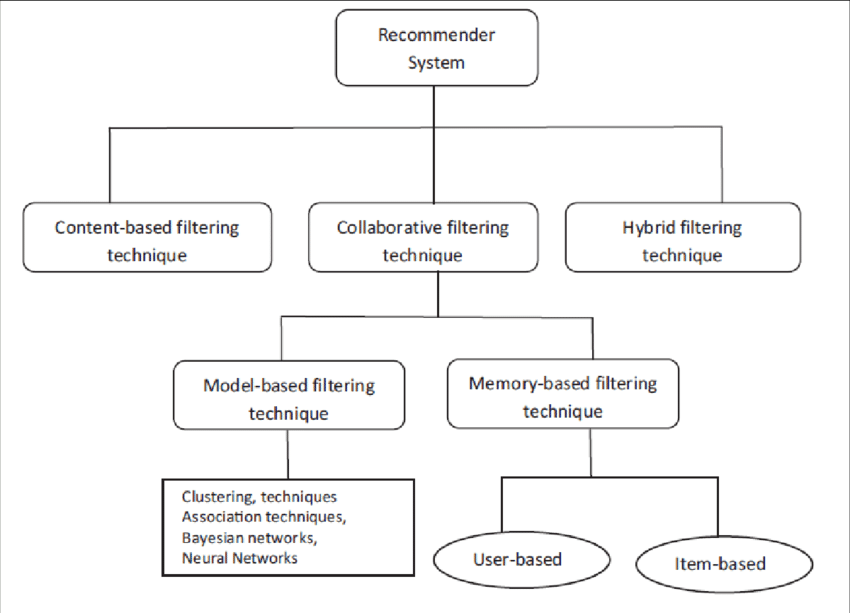
**3. Vectorizer/matrix-factorization**

**4. Similarity**

**5. Recommendations**

**Introduction:**

Recommendation systems are built to predict what users might like, especially when there are lots of choices available. They can explicitly offer those recommendations to users (e.g., Amazon or Netflix, the classic examples), or they might work behind the scenes to choose which content to surface without giving the user a choice.

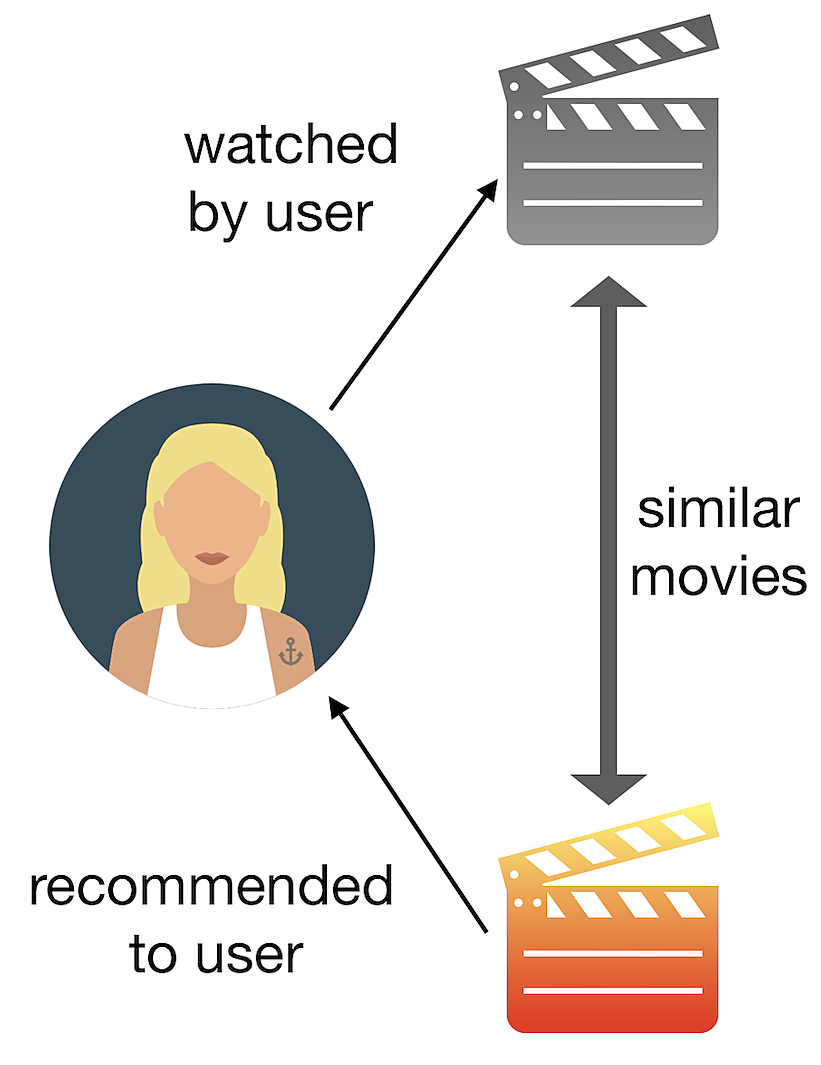


*Fig1. Classification of Recommendation Systems.*

**\*Content – Based Filtering:**

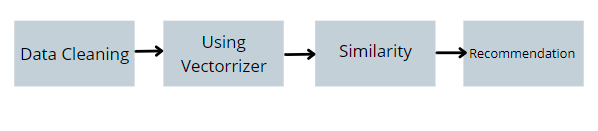
This filtering is based on the description or some data provided for that product. The system finds the similarity between products based on its context or description. The user’s previous history is taken into account to find similar products the user may like.

For example, if a user likes movies such as ‘Mission Impossible’ then we can recommend him the movies of ‘Tom Cruise’ or movies with the genre ‘Action’.



**Required Columns:** Only item details like Item-ID, Item-Description, etc.,

**Architecture:**



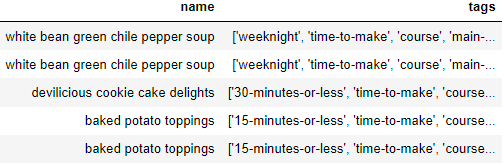
**Work Flow:**

1. **Data Cleaning:**

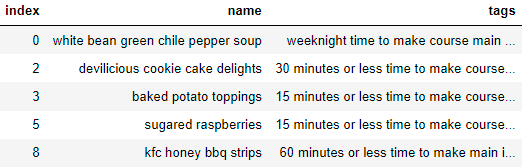
**a. Text Preprocessing**: Raw data contains numerical value, punctuation, special characters etc. as shown in the following Figure. This value can hamper the performance of model so before applying any text featurization first we need to convert raw data into meaningful data which is also called as text preprocessing.

Stop words are removed for more accuracy.

Before cleaning



After cleaning



1. **Vectorizer:**

The text format is converted into matrix format to find the similarity between items.

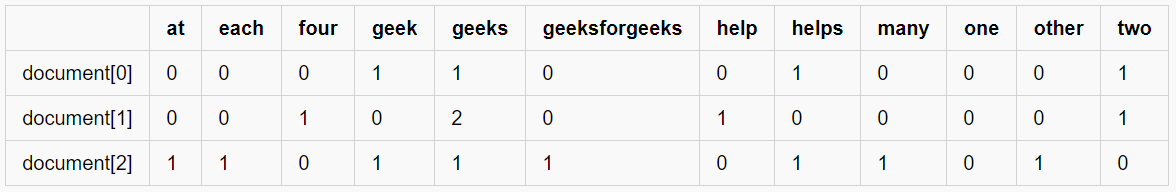
1. **Count Vectorizer:** It is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

Let us consider a few sample texts from a document (each as a list element):

***document = [ “One Geek helps Two Geeks”, “Two Geeks help Four Geeks”, “Each Geek helps many other Geeks at GeeksforGeeks.”]***

Count Vectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

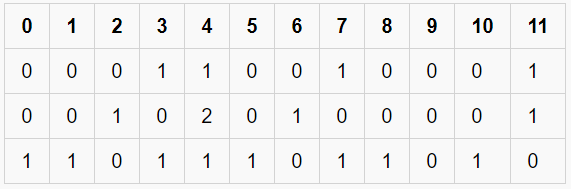
This can be visualized as follows –



**Key Observations:**

1. There are 12 unique words in the document, represented as columns of the table.
2. There are 3 text samples in the document, each represented as rows of the table.
3. Every cell contains a number, that represents the count of the word in that particular text.
4. All words have been converted to lowercase.
5. The words in columns have been arranged alphabetically.

Inside CountVectorizer, these words are not stored as strings. Rather, they are given a particular index value. In this case, ‘at’ would have index 0, ‘each’ would have index 1, ‘four’ would have index 2 and so on. The actual representation has been shown in the table below –



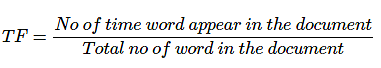
*A Sparse Matrix*

f. CountVectorizer will produce sparse matrix.

**2. TD-IDF:** TF-IDF stands for Term Frequency-Inverse Document Frequency which basically tells importance of the word in the corpus or dataset. TF-IDF contain two concept Term Frequency (TF) and Inverse Document Frequency (IDF).

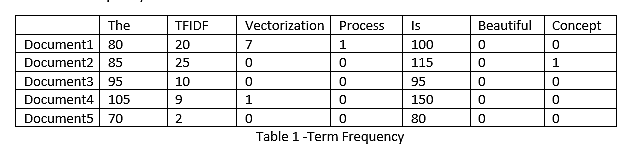
**Term Frequency**

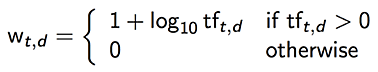
Term Frequency is defined as how frequently the word appear in the document or corpus. As each sentence is not the same length so it may be possible a word appears in long sentence occur more time as compared to word appear in sorter sentence. Term frequency can be defined as:



Let’s understand with this example

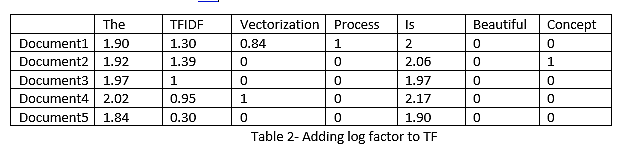
Suppose we have sentence “*The TFIDF Vectorization Process is Beautiful Concept*” and we have to find the find frequency count of these words in five different documents

As shown in Table 1 frequency of ‘***The***’ is maximum in every Document. Suppose frequency of ‘***The***’ in Document6 is 2 million while frequency of ‘***The***’ in Document7 in 3 million. Frequency of ‘***The***’ is very large in Document6 and Document7 so we can add log term to reduce the value of frequency count (log(2 million) =21). Adding log not only dampen the performance of idf but also reduce the frequency count of TF. Hence formula of TF can be defined as:



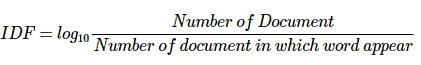
When tf = 1 log term will become zero and value will become 1 . Adding 1 is just to differentiate between tf=0 and tf =1

Hence Table 1 can be modified to :

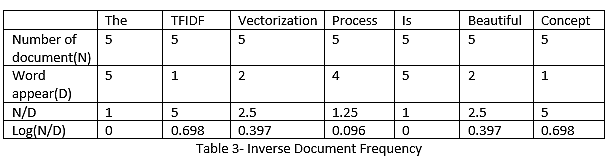


**Inverse Document Frequency**

Inverse Document frequency is another concept which is used for finding out importance of the word. It is based on the fact that less frequent words are more informative and important. IDF is represented by formula:



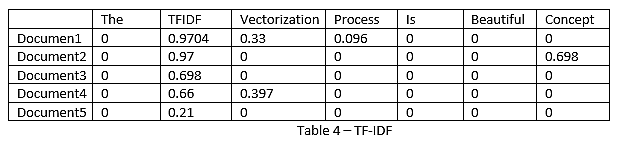
Let us consider the above example again



In Table 3 most frequent word is ‘***The***’ and ‘***is’*** but it is least important according to IDF and the word which appear very less such as ‘***TFIDF***’, ‘***Concept***’ are important words. Hence, we can say that IDF of rare term is high and IDF of frequent term is low.

**TF-IDF**

TF-IDF is basically a multiplication between Table 2 (TF table) and Table 3(IDF table) . It basically reduces values of common word that are used in different document. As we can see that in Table 4 most important word after multiplication of TF and IDF is ‘***TFIDF***’ while most frequent word such as ‘***The***’ and ‘***is***’ are not that important



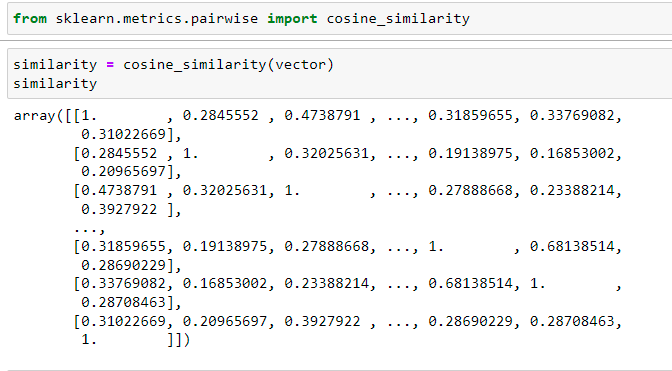
\*\*\*Based on the use case we can use the vectorizer.

If the features in the data are more, then we can combine all the relevant features and form a new column called tags and use count vectorizer .

If the data has only description of the item, then we can use TD-IDF vectorizer.

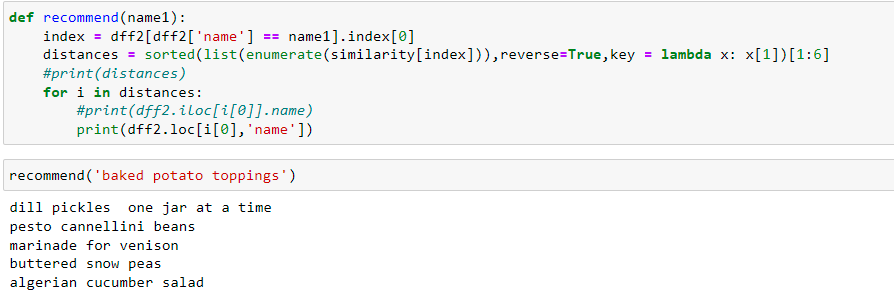
**3. Similarity:**

Cosine similarity is a metric used to measure how similar two items are. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The output value ranges from 0–1. 0 means no similarity, whereas 1 means that both the items are 100% similar.



The python Cosine Similarity or cosine kernel, computes similarity as the normalized dot product of input samples X and Y. We will use the sklearn cosine\_similarity to find the cos θ for the two vectors in the count matrix.

**4.Recommendation:**



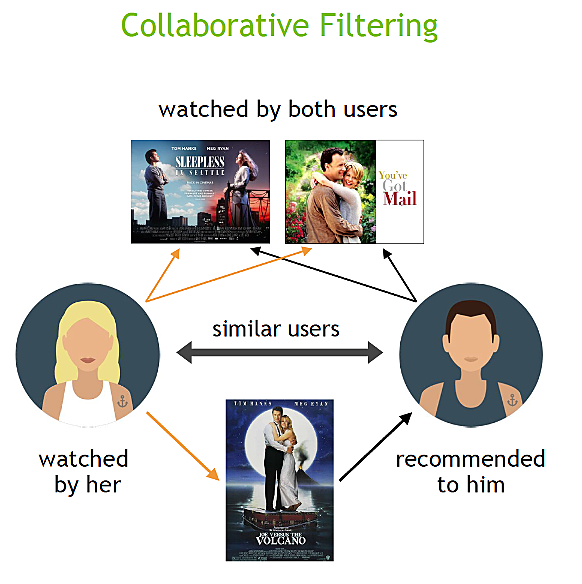
**Disadvantages**

a. The user will never be recommended for different items.

b. Business cannot be expanded as the user does not try a different type of product.

**\*Collaborative Filtering:**

The recommendations are done based on the user’s behaviour. History of the user plays an important role. For example, if the user ‘A’ likes ‘Coldplay’, ‘The Linkin Park’ and ‘Britney Spears’ while the user ‘B’ likes ‘Coldplay’, ‘The Linkin Park’ and ‘Taylor Swift’ then they have similar interests. So, there is a huge probability that the user ‘A’ would like ‘Taylor Swift’ and the user ‘B’ would like ‘Britney Spears’. This is the way collaborative filtering is done.



Classification based on technique:

A. Memory- Based

B. Model- Based

1. **Memory Based Collaborative Filtering:**

The main characteristics of user-user and item-item approaches it that they use only information from the user-item interaction matrix and they assume no model to produce new recommendations. Two types of collaborative filtering techniques are used:

I. User-User collaborative filtering

II. Item-Item collaborative filtering

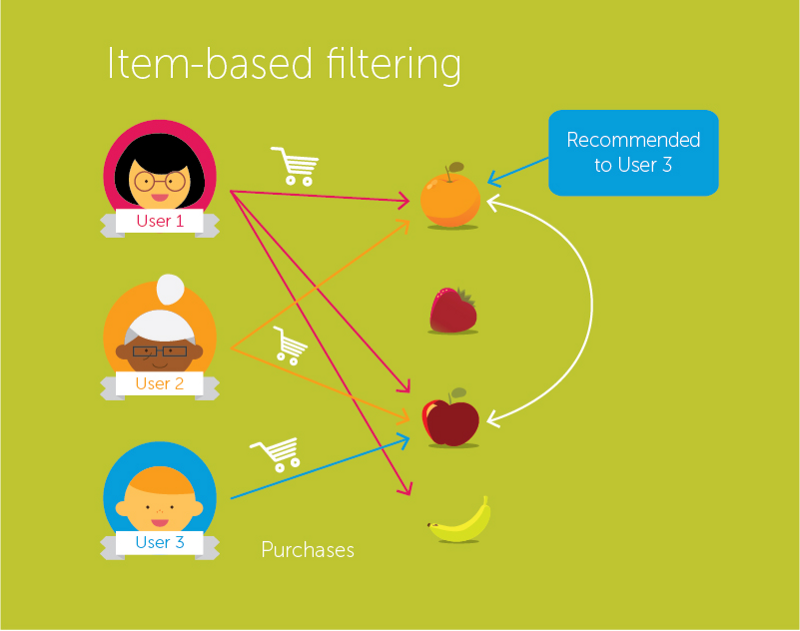
**User-User collaborative filtering**

In this, the user vector includes all the items purchased by the user and rating given for each particular product. Collaborative filtering

If a new user comes or old user changes his or her rating or provides new ratings then the recommendations may change.

**Item-Item collaborative filtering**

In this, rather than considering similar users, similar items are considered. If the user ‘A’ loves ‘Inception’ he may like ‘The Martian’ as the lead actor is similar. Here, the recommendation matrix is m\*m matrix where m is the number of items present.

Item-Item collaborative filtering

**Disadvantages**:

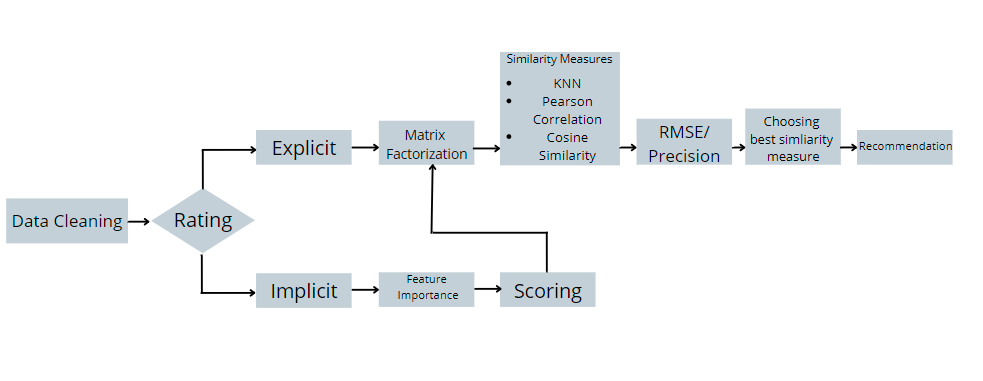
The memory-based method can provide significant recommended accuracy, but the execution time will grow rapidly with the increasing of users and items. Sometimes, it gets difficult to respond in real-time

**B. Model Based Collaborative Filtering:**

Model based collaborative approaches only rely on user-item interactions information and assume a latent model supposed to explain these interactions.

**Required Columns:** UserId, ItemId, Rating or any interaction.

**Architecture:**



**Work Flow:**

**Data Cleaning:**

**Rating:**

**Explicit Feedback:** It is the mechanism that allows a user to unequivocally express her interest in an object or set of objects. Typically, users assign a score to these objects through a survey process, such as the 5- star rating system or like/dislike rating system, to indicate their interest 175 in an object [18]. As discussed in [14], recommender systems usually collect users’ preferences using some of the rating systems cited above. For example, social networks such as Facebook, Twitter, Instagram, LinkedIn or YouTube use the like/dislike rating system as a mechanism for users to be able to rate contents explicitly. On the other hand, online stores such 180 as Amazon, AliExpress and others use the star ratings system, allowing users to indicate which products are of interest to them.

**Implicit feedback**: This process consists of getting the score of the 185 objects or products automatically, through capturing, analysing and processing the information retrieved from users’ behaviour in an application. For example, when a user reads news or accesses an online article, the time she takes for reading, comments on the content or whether the user has shared it on social networks, are automatically processed by the system 190 to infer whether the article or news is of interest to her. The use of this feedback technique helps improve the users’ experience and satisfaction when searching for content on the web, since it does not require explicit ratings to receive recommendations.

To calculate the score of the implicit rating we should give some weightage for each feature. To do that we'll be using feature importance to know how effective each feature.

**a.Feature Importance**: Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relativeimportance of each feature when making a prediction.

Feature importance scores can be calculated for problems that involve predicting a numerical value, called regression, and those problems that involve predicting a class label, called classification.

The scores are useful and can be used in a range of situations in a predictive modelling problem, such as:

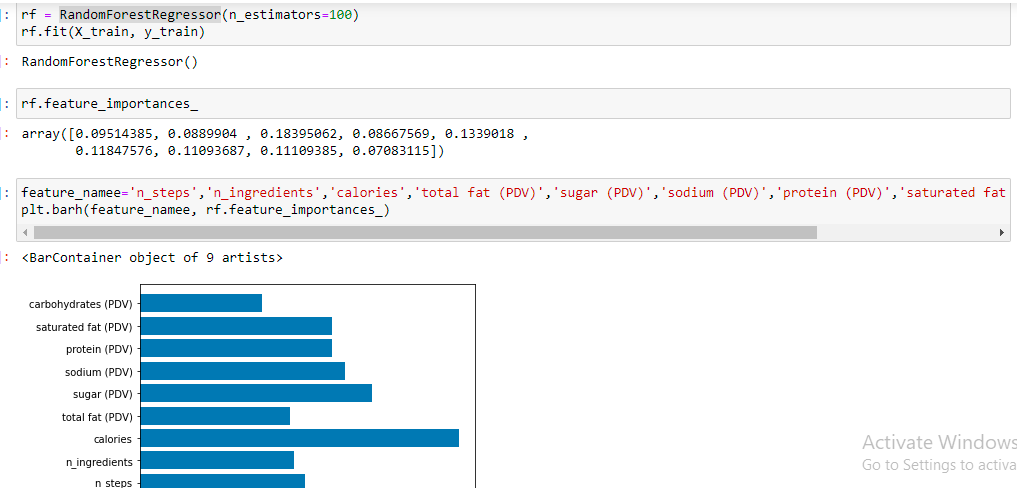
* Better understanding the data.
* Reducing the number of input features.

Feature importance scores can provide insight into the dataset. The relative scores can highlight which features maybe most relevant to the target, and the converse, which features are the least relevant. This maybe interpreted by a domain expert and could be used as the basis for gathering more or different data.

**feature importance reference:https://machinelearningmastery.com/calculate-feature-importance-with-python/**

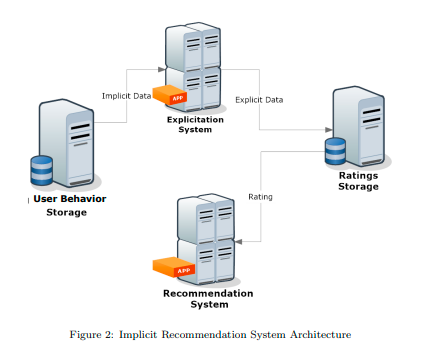
Out of all the above methods, we've tried 2 feature important techniques which are feasible for the dataset that we have tested.

1. RandomForestRegressor



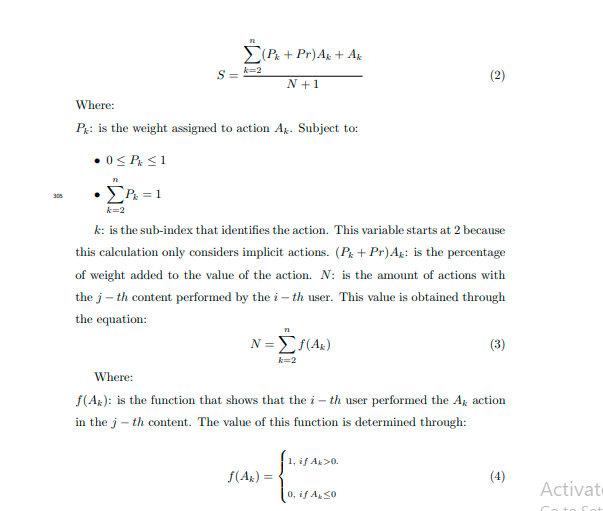
1. permutation\_importance

**b. Scoring [Calculating Rating using implicit feedback]**:



Reference links: file:///C:/Users/ysquare/Downloads/RecommenderSystembasedOnImplicitFeedbackForSelectiveDisseminationOfeBooks-4.pdf

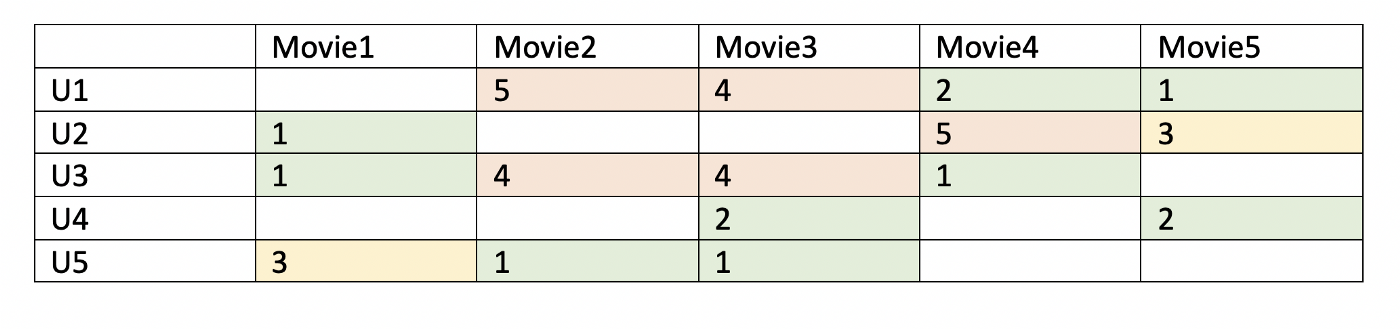
https://livebook.manning.com/book/practical-recommender-systems/chapter-4/



**Matrix Factorization:**

Matrix factorization is a way to generate latent features when multiplying two kinds of entities. Collaborative filtering is the application of matrix factorization to identify the relationship between items’ and users’ entities. With the input of users’ ratings on the shop items, we would like to predict how the users would rate the items so the users can get the recommendation based on the prediction.

Assume we have the customers’ ranking table of 5 users and 5 movies, and the ratings are integers ranging from 1 to 5, the matrix is provided by the table below.



*Table1-Users’ ratings table on movie*

Since not every user gives ratings to all the movies, there are many missing values in the matrix and it results in a sparse matrix. Hence, the null values not given by the users would be filled with 0 such that the filled values are provided for the multiplication. For example, two users give high ratings to a certain move when the movie is played by their favourite actor and actress or the movie genre is an action one, etc. From the table above, we can find that the user1 and user3 both give high ratings to movie and movie3. Hence, from the matrix factorization, we are able to discover these latent features to give a prediction on a rating with respect to the similarity in user’s preferences and interactions.

Given a scenario, user 4 didn’t give a rating to the movie 4. We’d like to know if user 4 would like movie 4. The method is to discover other users with similar preferences of user 4 by taking the ratings given by users of similar preferences to the movie 4 and predict whether the user 4 would like the movie 4 or not.

reference link: https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

**Similarity Measure:**

The performance of a recommender system depends upon the accuracy; how accurately the recommender system makes predictions. The prediction ability depends on similarity measure used to find similar users. The prediction will get as improved as much similarity measure provides better results. There are different similarity measures like pearson correlation, cosine similarity, JACCARD SIMILARITY, etc.,

All kinds of similarity measures are given below.

**Reference Links: http://ijetsr.com/images/short\_pdf/1498555415\_619-626-ieteh326\_ijetsr.pdf**

The models which are tried are:

i. Pearson correlation

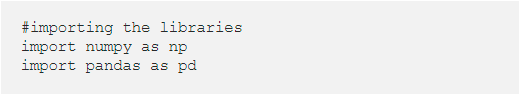
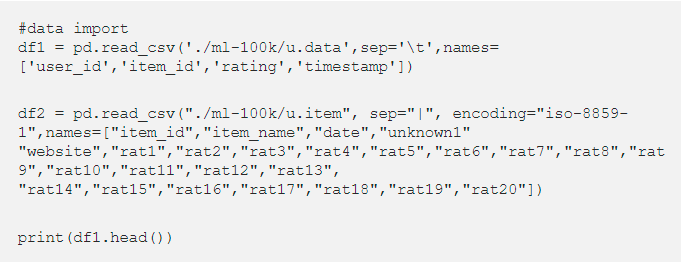
ii. Cosine Similarity

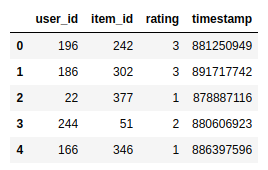
iii. KNN model

**Pearson correlation:**

Pearson’s Correlation Coefficient is a very simple yet effective way to find how 1 variable linearly changes with respect to another

Python Code :

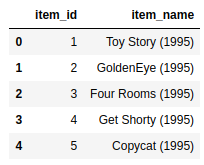
1. **Importing the necessary libraries.**
2. **Dataset (MovieLens Dataset)**  
   for the purpose of implementing recommender systems, I have used the movielens dataset which contains the ratings for 100k movies**output:**



The dataframe1 contains the user id , the movie id and the corresponding ratings

df2 = df2.iloc[:,0:2]  
df2.head()

**output:**



The dataframe2 contains the movie name and it’s corresponding item\_id

**3. Merging the dataframes**

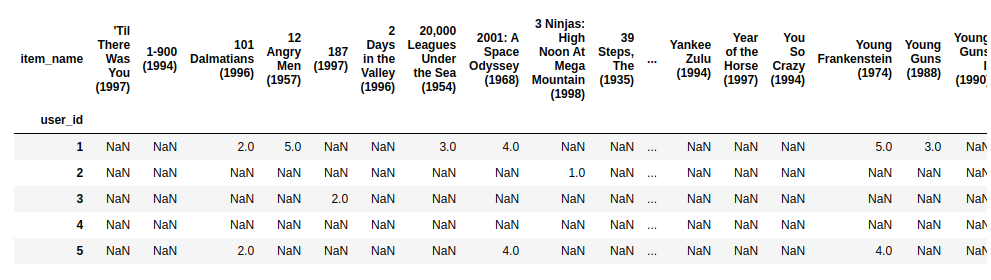
data = df1.merge(df2,on="item\_id")  
data.drop(['timestamp'],inplace=True,axis=1)  
data.head()

Merging the dataframe 1 to dataframe 2 to get the entire dataset

**4. Pivot Table**

data\_table = pd.pivot\_table(data,values='rating',columns='item\_name',index='user\_id')  
data\_table.head()

**output:**

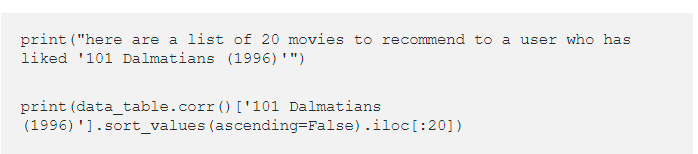


We utilize the Pivot Table from pandas create a table with each movie representing a column and each user representing a row

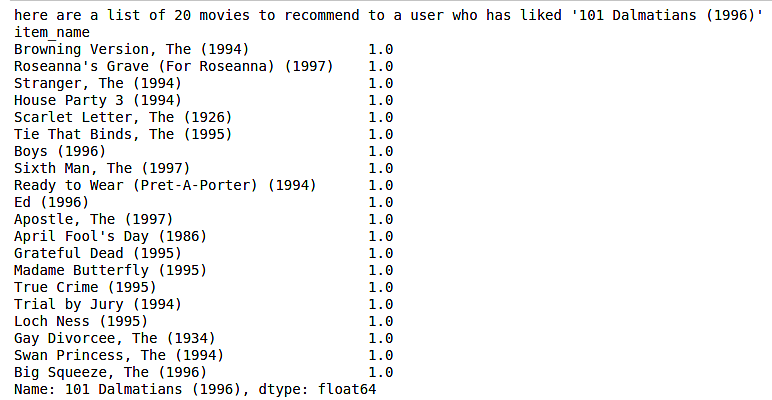
**5. Start Recommending**

That’s it for this basic recommender systems, inorder to make predictions , we are going to get a movie name from the user and give a list of movies that the user might like. This is where the correlation coefficient comes into play

**Let’s assume that the user liked the movie 101 Dalmatians (1996). we have to give a list of movies that we think the user might like.**



**output:**



So, This is how we can use the pearson’s correlation coefficient to recommend movies to users based on the movies they liked

# Conclusion

We use the pivot table and correlation coefficient to recommend movies here. If the user likes a particular movie, we take that movie’s columns and find the correlation of that column with all the other movie columns and get the movies that highly correlate with the chosen movie.  
This works because, the rows represent users and a particular user might like similar movies. Hence, we can use correlation coefficient to recommend movies to the users.

**KNN:**

**Collaborative filtering** systems use the actions of users to recommend other movies. In general, they can either be user-based or item-based. **Item based 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. But what is the KNN? **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 neighbor movies as the most similar movie recommendations.

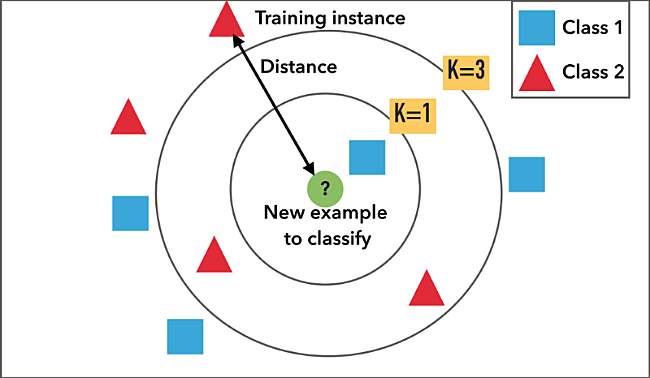


Illustration of how KNN makes classification about new sample

Wait, but how do we feed the dataframe of ratings into a KNN model? First, 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). Let’s call this new dataframe a “dataframe of movie features”.

Our dataframe of movie features is an extremely sparse matrix with a shape of 13,500 x 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**.

**Multi-Criteria Recommender System (MCRS):**

Collaborative\_Recommendation\_with\_Multi-Criteria\_R.pdf------ BASIC INTRO CAN BE FOUND HERE IN THIS LINK.

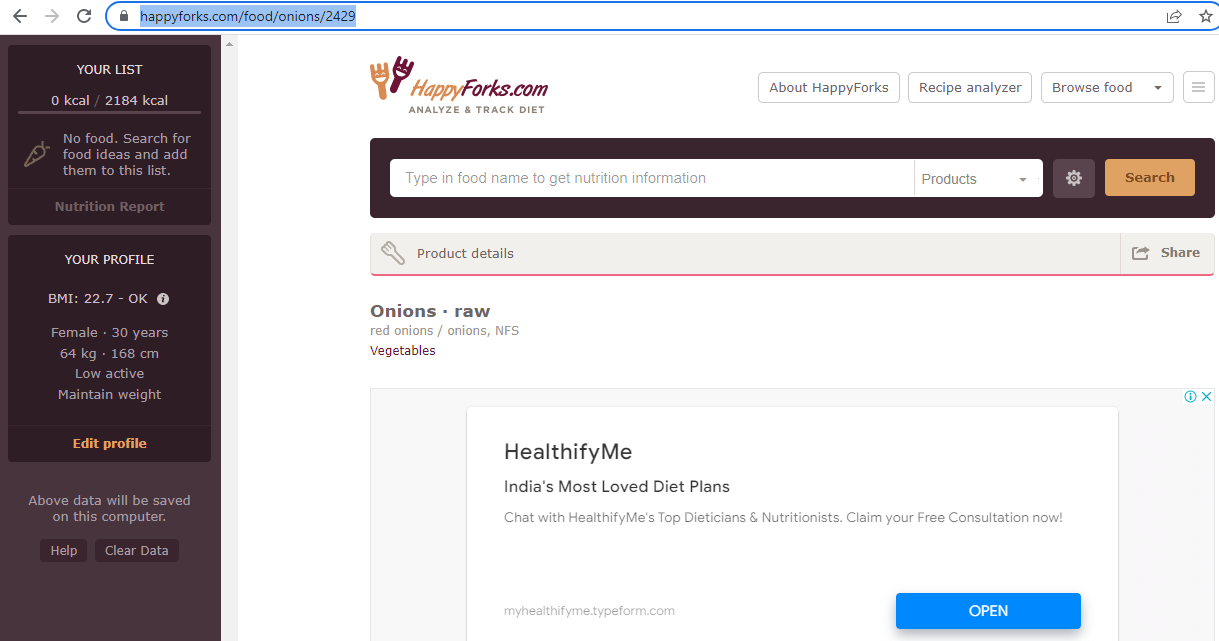
1symeonidis\_panagiotis\_zioupos\_andreas\_matrix\_and\_tensor\_fact.pdf-------101 PAGES book which contains how this multi-criteria problem can be solved using tensor factorization.

**Solve multi criteria recommendation system**

Converted the usecase into supervised keeping rating as the outcome column but faced issues of overfitting

so we are trying out 2 unsupervised algorithms 1 based on ratings and other based on ingredients for a food recommendation usecase and using rules engine combine them both based on the preference of the user.

Nutrition recipe analyser



https://happyforks.com/food/onions/2429

https://www.verywellfit.com/recipe-nutrition-analyzer-4157076