Problem formulation

To build a Movie Recommendations with Movielens Dataset

What is a Recommendation System?

Simply put a Recommendation System is a filtration program whose prime goal is to predict the “rating” or “preference” of a user towards a domain-specific item or item. In our case, this domain-specific item is a movie, therefore the main focus of our recommendation system is to filter and predict only those movies which a user would prefer given some data about the user him or herself.

Collaborative Filtering

This filtration strategy is based on the combination of the user’s behavior and comparing and contrasting that with other users’ behavior in the database. The history of all users plays an important role in this algorithm. The main difference between content-based filtering and collaborative filtering that in the latter, the interaction of all users with the items influences the recommendation algorithm while for content-based filtering only the concerned user’s data is taken into account. There are multiple ways to implement collaborative filtering but the main concept to be grasped is that in collaborative filtering multiple user’s data influences the outcome of the recommendation. and doesn’t depend on only one user’s data for modeling.

There are 2 types of collaborative filtering algorithms:

- ***User-based Collaborative filtering***: The basic idea here is to find users that have similar past preference patterns as the user ‘A’ has had and then recommending him or her items liked by those similar users which ‘A’ has not encountered yet. This is achieved by making a matrix of items each user has rated/viewed/liked/clicked depending upon the task at hand, and then computing the similarity score between the users and finally recommending items that the concerned user isn’t aware of but users similar to him/her are and liked it.

  For example, if the user ‘A’ likes ‘Batman Begins’, ‘Justice League’ and ‘The Avengers’ while the user ‘B’ likes ‘Batman Begins’, ‘Justice League’ and ‘Thor’ then they have similar interests because we know that these movies belong to the super-hero genre. So, there is a high probability that the user ‘A’ would like ‘Thor’ and the user ‘B’ would like The Avengers’.

  Disadvantages

  - People are fickle-minded i.e their taste change from time to time and as this algorithm is based on user similarity it may pick up initial similarity patterns between 2 users who after a while may have completely different preferences.
  - There are many more users than items therefore it becomes very difficult to maintain such large matrices and therefore needs to be recomputed very regularly.
  - This algorithm is very susceptible to shilling attacks where fake users profiles consisting of biased preference patterns are used to manipulate key decisions.

- ***Item-based Collaborative Filtering***: The concept in this case is to find similar movies instead of similar users and then recommending similar movies to that ‘A’ has had in his/her past preferences. This is executed by finding every pair of items that were rated/viewed/liked/clicked by the same user, then measuring the similarity of those rated/viewed/liked/clicked across all user who rated/viewed/liked/clicked both, and finally recommending them based on similarity scores.

  Here, for example, we take 2 movies ‘A’ and ‘B’ and check their ratings by all users who have rated both the movies and based on the similarity of these ratings, and based on this rating similarity by users who have rated both we find similar movies. So if most common users have rated ‘A’ and ‘B’ both similarly and it is highly probable that ‘A’ and ‘B’ are similar, therefore if someone has watched and liked ‘A’ they should be recommended ‘B’ and vice versa.

  Advantages over User-based Collaborative Filtering:

  - Unlike people’s taste, movies don’t change.
  - There are usually a lot fewer items than people, therefore easier to maintain and compute the matrices.
  - Shilling attacks are much harder because items cannot be faked.

Algorithms followed

Collaborative Filtering

This Movie Recommendation Sysytem employs an algorithm known as *Collaborative Filtering*.We will be using ***Item-based Collaborative Filtering*** which filters the items based on the similarity between different items.

The concept in the case of ***Item-based Collaborative Filtering*** is to find similar movies and then recommend them to a user based on his/her past preferences.This is executed by finding every pair of items that were rated/viewed/liked/clicked by the by the same user, then measuring the similarity of those rated/viewed/liked/clicked across all users who rated/viewed/liked/clicked both, and finally recommending them based on similarity scores.

To acheive that we use a "distance matrix" which finds the cosine distance between two items and if the distance is less the items are similar and more likely to be recommended. To generate the distances or similarity between all the items we first create a *ratings matrix* which contains all the movies in columns and users in rows and the ratings of the users as the values. Then scaling is applied to the matrix to scale the values appropriately.

We will be using the ***KNN algorithm*** to compute similarity with metric **cosine distance** which is very fast and more preferable than *pearson coefficient*.

What is K-NN algorithm?

The k-nearest neighbors (KNN) algorithm is a **simple, easy-to-implement, non-parametric, lazy learning, supervised machine learning algorithm** that can be used to solve both classification and regression problems using feature similarity. Learning KNN machine learning algorithm is a great way to introduce yourself to machine learning and classification in general. At its most basic level, it is essentially classification by **finding the most similar data points** in the training data, and making an educated guess based on their classifications.

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily

classified into a well suite category by using K- NN algorithm.

- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example**: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1. The cosine of 0° is 1, and it is less than 1 for any angle in the interval (0, π] radians. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90° relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. The cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. The name derives from the term "direction cosine": in this case, unit vectors are maximally "similar" if they're parallel and maximally "dissimilar" if they're orthogonal (perpendicular). This is analogous to the cosine, which is unity (maximum value) when the segments subtend a zero angle and zero (uncorrelated) when the segments are perpendicular.

These bounds apply for any number of dimensions, and the cosine similarity is most commonly used in high-dimensional positive spaces. For example, in information retrieval and text mining, each term is notionally assigned a different dimension and a document is characterised by a vector where the value in each dimension corresponds to the number of times the term appears in the document. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter.

One advantage of cosine similarity is its low-complexity, especially for sparse vectors: only the non-zero dimensions need to be considered.

Dataset used

MovieLens Dataset is used in this project to train the model for movie recommendations

Results and Discussion

After training the model with the given dataset, the model produces well enough results(recommendations) for some of the movies while for other movies the recommendations are a bit random. This might be happening due to noise in our dataset and one of the shortcommings of KNN model is that when a small amount of noise is introduced to the data, the results deviate a lot.

For Example:

- When ***Iron Man*** is given as the input(previous preference of the user) which is a popular movie in the recent times, the model generates the following results(recommendations):

    - Batman Begins (2005)
    - Dark Knight, The (2008)
    - Dark Knight Rises, The (2012)
    - ***Avengers, The (2012)***
    - Guardians of the Galaxy (2014)
    - Inception (2010)
    - ***Iron Man 2 (2010)***
    - WALL·E (2008)
    - ***Avengers: Age of Ultron (2015)***
    - Avatar (2009)

    And to be honest, the recommendations are really great! It recognizes the movie and recommends movies belonging to the simmilar genre(action, sci-fi thriller) and which are also quite poplular. The model does a great job at recommending the movies and is able to quantify the users preferences.

- But that is not always the case, as we can see when we give ***Titanic***, which is a romantic movie, as the input(previous preference of the user) which is a popular movie but is a bit old. The model generates the following results(recommendations):

    - Truman Show, The (1998)
    - Sixth Sense, The (1999)
    - Saving Private Ryan (1998)
    - Forrest Gump (1994)
    - Good Will Hunting (1997)
    - ***Men in Black (a.k.a. MIB) (1997)***
    - Back to the Future (1985)
    - As Good as It Gets (1997)
    - Gladiator (2000)
    - ***Catch Me If You Can (2002)***

    As we can see, these recommendations are mostly action and thriller movies which are completely different from **Titanic** which is a popular **Romantic** movie. Here we can see the model falls apart and is unable to get good recommendations for the users preferences.

- To give another example we give ***Your Name***, which is a **animated** romantic movie, as the input(previous preference of the user) which is a popular movie. The model generates the following results(recommendations):

    - ***A Silent Voice (2016)***
    - Bruce, King of Kung Fu (1980)
    - Bruce Lee Fights Back from the Grave (1976)
    - Dragon Bruce Lee, Part II (1981)
    - ***Kubo and the Two Strings (2016)***
    - Personal Shopper (2016)

    - Sing (2016)
    - Bloodfisted Brothers (1978)
    - ***Finding Dory (2016)***
    - February (2015)

    For this input, The recommendations are pretty mixed to be honest, for some recommendations it did pretty well and recommended good movies from the same genre or simmilar but for some recommendations it was not quite there and recommended some action thriller which is far from the romantic movie we preffered.

From all these examples, it seems that the model is heavily favouring the **Action** and **Thriller** genre over the rest. This might have occured due to higher number of ratings for those movies and due to the noise in the data. But the model seems quite usable for most recommendations and with some fine tuning might be even better.

Future Improvements

- Adding ***Genre Correlation*** to the feature matrix might help to filter more effectively.
- Using **Movie tags** and quantifying that data can be useful to better recommend similar items to the user based on the preferences.

References

- Concepts
- Hands-on recommendation system
- Dataset
- Movie Recommendation System
- KNN Algorithm
- KNN Algorithm
- Cosine Similarity