Building recommendation system to predict user preferences for entertainment product

“Ayushi Tripathi” “Anurag” “Sugandha Tanwar” “Satyam Sharma” “Varun Parmar”

ABSTRACT

Recommender engines (REs) also known as recommender systems are software tools and techniques providing suggestions to a user. The suggestions provided are aimed at supporting their users in various decision making processes such as what items to buy, what music to listen, what profiles to browse, or what news to read. The vast amount of data available on the Internet has led to the development of recommendation systems. This project proposes the use of soft computing techniques to develop recommendation systems. It addresses the limitations of current algorithms used to implement recommendation systems, evaluation of experimental results, and conclusion. This report provides a detailed summary of the project “Buildings recommendation system to predict user preferences for entertainment product”. The report includes a description of the topic, system architecture, and provides a detailed description of the work done till point. Included in the report are the detailed descriptions of the work done: snapshots of the implementations, various approaches, and tools used so far. The report also includes the project schedule and deliverable.

INTRODUCTION

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggestions based on these preferences. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google. Often, these systems are able to collect information about a users choices, and can use this information to improve their suggestions in the future. Two most ubiquitous types of personalized recommendation systems that we used in our project are Content-Based and Collaborative Filtering. The Content-Based Recommender relies on the similarity of the items being recommended. The basic idea is that if you like an item, then you will also like a “similar” item. It generally works well when it's easy to determine the context/properties of each item. A content based recommender works with data that the user provides, either explicitly movie ratings for the MovieLens dataset. Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate. The Collaborative Filtering Recommender is entirely based on the past behavior and not on the context. More specifically, it is based on the similarity in preferences, tastes and choices of two users. It analyses how similar the tastes of one user is to another and makes recommendations on the basis of that. For instance, if user A likes movies 1, 2, 3 and user B likes movies 2,3,4, then they have similar interests and A should like movie 4 and B should like movie 1. This makes it one of the most commonly used algorithm as it is not dependent on any additional information.

In general, collaborative filtering is the workhorse of recommender engines. The algorithm has a very interesting property of being able to do feature learning on its own, which means that it can start to learn for itself what features to use. It can be divided into Memory-Based Collaborative Filtering and Model-Based Collaborative filtering

TECHNOLOGY USED

- **Python:**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

- **Jupyter Notebook (Google Colab)**

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Also include data cleaning and transformation, Numerical simulation, statistical modelling data visualization, machine learning and much more.

LITERATURE SURVEY

As we know that with an increasing no of online companies which are utilizing recommendation system to increase user interaction and enrich shopping potential. Use cases of recommendation systems have been expanding rapidly across any aspects of e-Commerce and online media over the last 4-5 years and we expect this trend to continue.

Recommendation sytems(often called “recommendation engines”) have the potential to change the way websites communicate with users and to allow companies to maximize their ROI based on the information the can gather on each customer’s preferences.

DATASET AND FEATURE DESCRIPTION

Dataset Description

Our movielens dataset consists of these files contain 1,000,209 anonymous ratings of approximately 3,900 movies. made by 6,040 MovieLens users who joined MovieLens in 2000.

Feature description

The movielens dataset that we have got contains 3 files i.e ratings file, users file and movies file.

Movies Dataset: It contains following

- MovieID: Each movies has a Unique ID
- Title: Titles (Name) of Movies
- Genres: Category of Movies(like: Drama, Comedy, Action etc..)
- Titles are identical to titles provided by the IMDB (including year of release)
- Shape : (3883,3)

User Datasets: It Contains following

- UserID: Each user who rate the movies has an unique ID
- Age: age is chosen from the following ranges:
    - 1: "Under 18"
    - 18: "18-24"
    - 25: "25-34"
    - 35: "35-44"
    - 45: "45-49"
    - 50: "50-55"
    - 56: "56+"
- Gender: User's Gender (M or F)
- Occupation: It ranges (0-20) each number has represents Occupation of a particular of a User
- 0: "Other" Or Not Specified
- 1: "Academic/Educator"
- 2: "Artist"
- 3: "Clerical/Admin"4: "Co
- 5: "Customer Service
- 6: "Doctor/Health Care"
- 7: "Executive/Managerial"
- 8: "Farmer”
- 9: "Homemaker"
- 10: "K-12 Student"
- 11: "Lawyer"
- 12: "Programmer"
- 13: "Retired"
- 14: "Sales/Marketing"
- 15: "Scientist"
- 16: "Self-Employed"
- 17: "Technician/Engineer"Lege/Grad Student"
- 18: "Tradesman/Craftsman"
- 19: "Unemployed"
- 20: "Writer"
- Zip-code: Area or Zip code of a user
- Shape: (6040,5)

Rating Datasets:

- UserIDs: ranges(1 to 6040)
- MovieIDs: range between 1 and 3952
- Ratings: 1-5-star rating by user
- Timestamp: Represented the time mentioned in seconds when the rating was provided
- Shape: (1000209,4)

LIBRARIES USED

Pandas - It is a fast , powerful , flexible and easy to use open source data analysis and manipulation tool , built on top of the python programming language.

NumPy – It is a library for the python programming language, adding support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Scikit – Scikit-learn is probably the most useful library for machine learning in python . the sklearn library contains a lot of efficient tools for machine learning and statistical modelling, regression, clustering and dimensionality reduction.

Scipy – It is a free and open source python library used for scientific computing. It conatins modules for optimisation ,linear algebra, integration, interpolation etc.

Surprise library - It is a easy-to-use python scikit for recommender system

LIBRARIES FOR DATA VISUALISATION

Matplotlib – It is a plotting library which we have used in our project for data visualisation. It is a library for python programming language and its numerical extension NumPy.

Seaborn – It is a python data visualisation library based on matplotlib. It provides a high level interface for drawing attractive and informative statistical graphics.

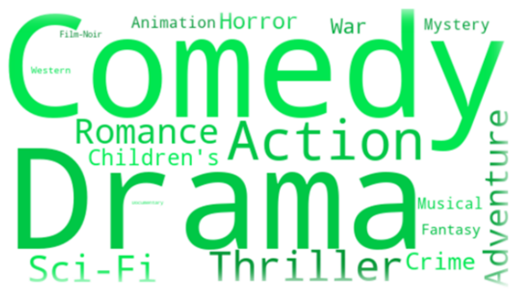
Word Cloud – It is a data visualisation technique used for representing text data in which the size of each word indicates its frequency or important significant textual data points can be highlighted using a word cloud. Word Cloud is mostly used for analysing data from social network websites.

EXPLORATORY DATA ANALYSIS

Exploratory data analysis is an approach to analysing the datasets to summarise their main characteristics, often with visual methods . A statistical can be used or not , but primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task.

Word Cloud:

First we created a word cloud for 'Genres' from this we all had visualise that the word 'Comedy' is the most commonly used word in the movies Genre column. I think this encapsulates the idea of presence of comedy in large amount in movies Genre.



Univariate Analysis

- Numerical Features – Ratings , UserID, MovieID , Age, Occupation,Timestamp
- Categorial Features – Title, Gender , Genre , Gender

Plot Histogram:

- Frequency is highest for rating 4 and least for rating 1.
- Frequency is highest for age group (22-24) and least for age group below 10 i.e (0-10).
- Frequency for 0 occupation is highest i,e for ('others' or 'not-specified') and least for occupation(7.5 to 10)

Occupation Distribution – By using bar plot we visualize that users belong to which occupation are more and for which are least

Inferences:

- Highest frequency is for Occupation 4 i.e ('col').
- Lowest frequency is for Occupation 8 i,e ('farmer').
- Gender Distribution – We visualise with the help of histogram that Frequency of number of users is high for male as compare to female

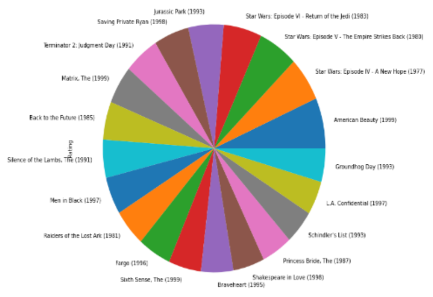
Bivariate Analysis:

Seaborn Facet Grid - We have used it because it takes the given data onto multiple axes arrayed in a grid of rows and columns that corresponds to levels of variables in the dataset.

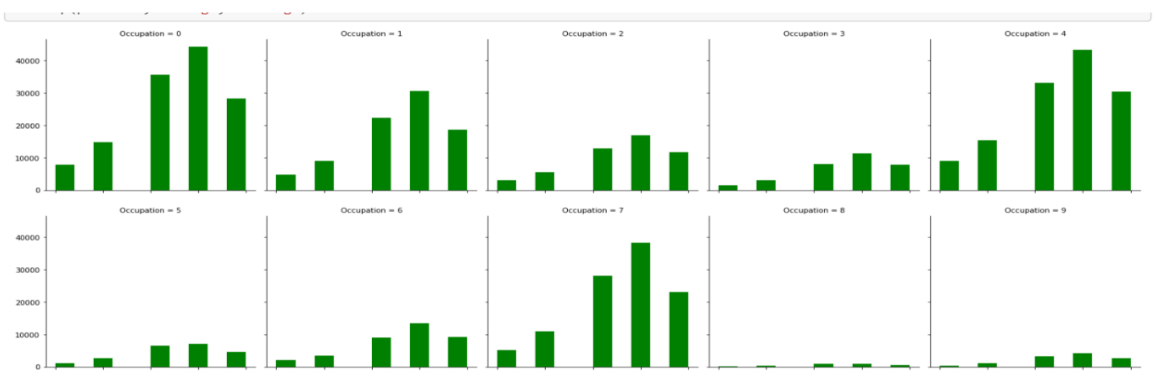
Seaborn Facet Grid between gender and Rating

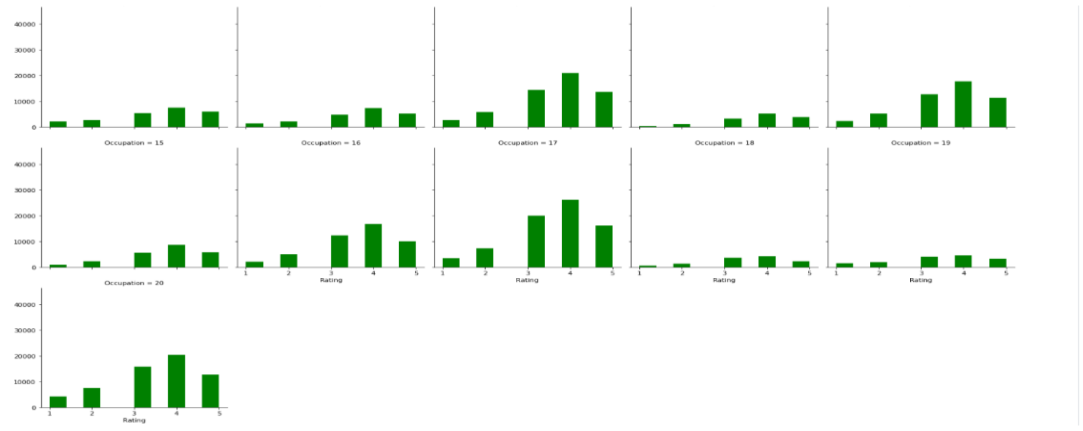
Inferences:

- Most of the users are male who watched or rate the movies at maximum.
- The users of age group 25-34 are most interested about giving rating. It means youngsters are more interested in giving ratings.
- The users of age group under 18 and 56+ are least concerned about giving rating.
    - Top 10 most rated movies

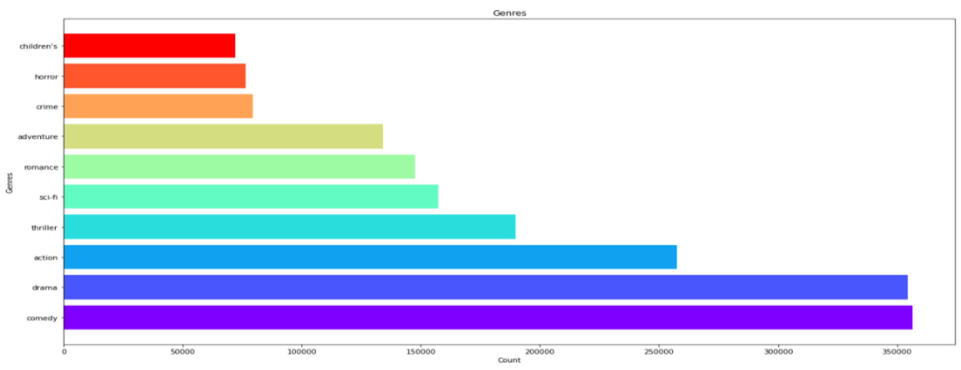


- Show the rating distribution across all the Occupation

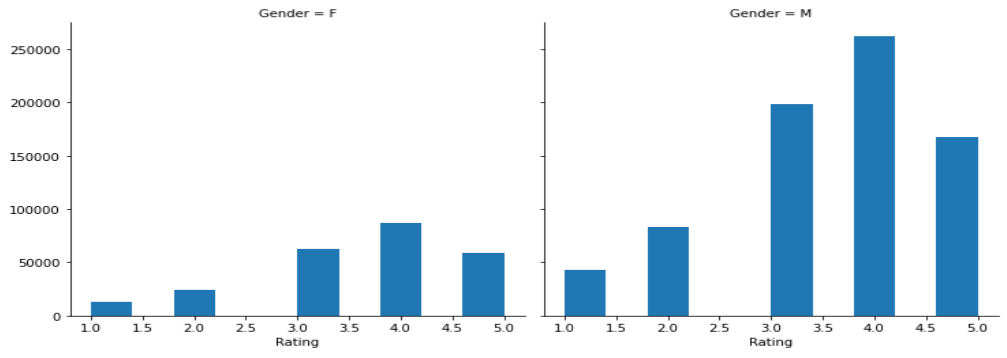




- Show the Rating distribution across all Genres



- Show the Rating Rating distribution across Gender



FEATURE ENGINEERING

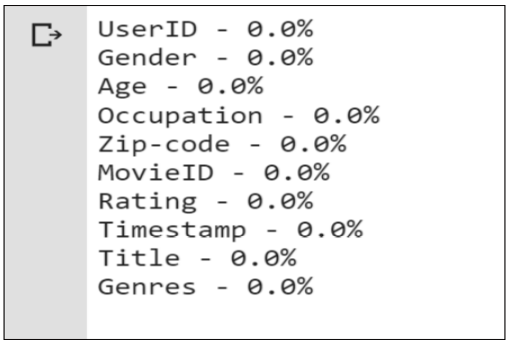
Feature engineering is the process by which knowledge of data is used to construct explanatory variables, features, that can be used to train a predictive model. Engineering and selecting the correct features for a model will not only significantly improve its predictive power, but will also offer the flexibility to use less complex models that are faster to run and more easily understood.

The movielens Dataset that we got is already cleaned by grouplens research but for our concern and satisfaction we checked for NAN and NULL values and it is right that there are no NULL,NAN and missing values in our dataset. And we also perform some technique to check weather outliers in dataset present or not.

Detecting and handling missing values:

We checked for the missing values in our dataset using Missing data percentage list which returns the percentage of missing values in each and every column present in our dataset. In our case we got 0.0% missing values in every columns.

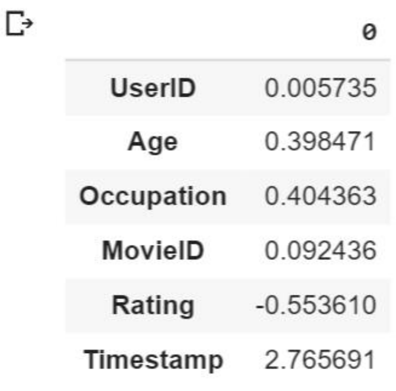
Since we don’t get any missing values in dataset, so there is no need to perform handling missing values techniques.



```
UserID - 0.0%
Gender - 0.0%
Age - 0.0%
Occupation - 0.0%
Zip-code - 0.0%
MovieID - 0.0%
Rating - 0.0%
Timestamp - 0.0%
Title - 0.0%
Genres - 0.0%
```

Outliers handling and detection

For detection outliers in data we check skew value for continuous features. If skewness value lies between -1 and 1 then the data is normally distributed and any major deviation from this range indicates the presence of extreme values in dataset. So we see only Timestamp skew value is 2.765691 which is greater then 1 so there maybe chance of outliers in Timestamp. and other features skewness is bearable. but timestamp column which stores the last time the rating was updated. it is the time mentioned in seconds when the rating was provided. because of it is a time. it is be like primary key or it has more then 45000 unique values. It is different for different users. we can't perform outlier treatment on this column.

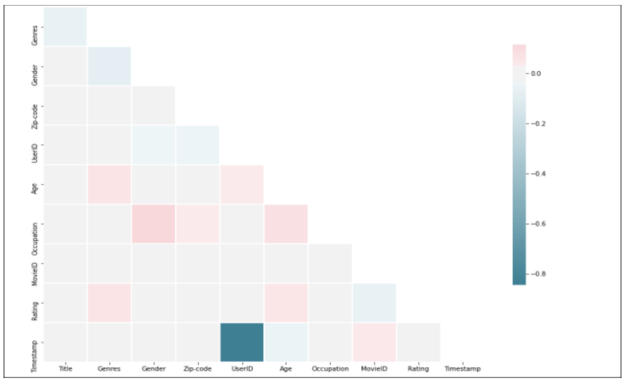


	0
UserID	0.005735
Age	0.398471
Occupation	0.404363
MovieID	0.092436
Rating	-0.553610
Timestamp	2.765691

Feature Selection

Pearson Correlation: Pearson's Correlation Coefficient. Correlation is a technique for investigating the relationship between two quantitative, continuous variables, for example, age and blood pressure. Pearson's correlation coefficient (r) is a measure of the strength of the association between the two variables.

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}}$$



- A score closer to 1 or -1 is a positive or negative relationship. A perfect score of 1 is a direct correlation.
- in this heat map very light pink means positive, dark voilet means negative. The stronger the color, the larger the correlation magnitude.

- So by analyzing the above heatmap and correlation matrix Timestamp and Zip-code has very negative value (-0.843524) ,(-0.044845) and it goes on -1 side it negative correlation with other so we can think about to Drop it from dataset because it not contribute much information as comapare to others.
- And also if think about categories of recommendatiomns systems both are not contribute much as we need Because in Content Based Filtering we needs user related information which relate it to recommend movies and Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future and that they will like similar kind of objects as they liked in the past So according to this scenario we don"t need these features.
- Features needed Collaborative filtering*: in this we recommend movies on the basis of ratings provided by user or the similarty of user and we need contains only those features that provide information about the users(i.e.UserID, Age, User_Rating, Occupation, Gender, Zip-code)
- Content Based filtering: In this we recommend movies to user on the basis of past history of user and content that it has.In this the algorithm will simply pick an item which similar content to recommender.So in this we need only those features who related to a particular user(i.e. UserID, MovieID, Title, Genres, Ratings)

DIFFERENT APPROACHES TO RECOMMENDATION SYSTEM

There are several taxonomies for recommender systems but we used two most popular approaches: collaborative and content-based.

COLLABORATIVE FILTERING

It is the most popular and most implemented approach. In its original and simplest implementation this approach recommends to the active user, items that other users with similar tastes liked in the past. The similarity in taste of two users is calculated based on the similarity in the rating history of the users.

User-Based Collaborative Filtering: a target user’s choices are compared with other users in the database to identify a group of “similar minded” people. Once this group is identified, highly rated content from the group are then recommended to the target user. This approach computes the correlation with all other users for each item and aggregate the rating of highly correlated users.User-based methods rely on the opinion of like-minded users to predict a rating, and generate recommendations.

IMPLEMENTATION

SINGULAR VALUE DECOMPOSITION

The Singular Value Decomposition (SVD), a method from linear algebra that has been generally used as a dimensionality reduction technique in machine learning. SVD is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). In the context of the recommender system, the SVD is used as a collaborative filtering technique. It uses a matrix structure where each row represents a user, and each column represents an item. The elements of this matrix are the ratings that are given to items by users.

The factorisation of this matrix is done by the singular value decomposition. It finds factors of matrices from the factorisation of a high-level (user-item-rating) matrix. The singular value decomposition is a method of decomposing a matrix into three other matrices as given below:

$$A = USV^T$$

Where

- *A* is a *m x n* utility matrix,
- *U* is a *m x r* orthogonal left singular matrix, which represents the relationship between users and latent factors.
- *S* is a *r x r* diagonal matrix, which describes the strength of each latent factor and
- *V* is a *r x n* diagonal right singular matrix, which indicates the similarity between items and latent factors.

The latent factors here are the characteristics of the items, for example, the genre of the music. The SVD decreases the dimension of the utility matrix *A* by extracting its latent factors. It maps each user and each item into a *r*-dimensional latent space. This mapping facilitates a clear representation of relationships between users and items.

Item-Based Collaborative Filtering: examines each item on the target user’s list of chosen/rated items and finds other items in the choice set that seems similar to the item. In this case, similarity can be determined on predefined attributes (e.g. movie genre, lead actors, director, etc.) and/or by calculating correlations between items. The main advantage of Item-Based CF over User-Based CF is increased scalability due to the fact that items can be classified or pre-scored based on explicit attributes, making recommendation computation much faster. This approach computes for each user item the correlation with all other item and aggregates for each user the ratings for item that are already highly correlated. Item-based approaches look at ratings given to similar items and generate recommendation.

IMPLEMENTATION:

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

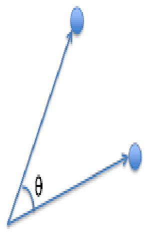
KNN is a non-parametric, lazy learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.

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.

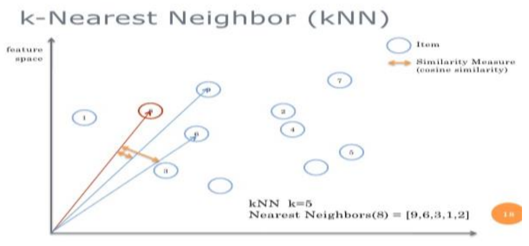
WORKING:

We use Nearest Neighbors algorithms with sklearn.neighbors. The algorithm we use to compute the nearest neighbors is "brute", and we specify "metric=cosine" so that the algorithm will calculate the cosine similarity between rating vectors.

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



Similarly, we will calculate distance of all the training cases with new case and calculates the rank in terms of distance. The smallest distance value will be ranked 1 and considered as nearest neighbor.


k-Nearest Neighbor (kNN)
feature space
Item
Similarity Measure (cosine similarity)
1
2
3
4
5
6
7
8
9
kNN k=5
Nearest Neighbors(8) = [9,6,3,1,2]
18

Train model:

we fit the model by trainset. And we use it for recommend the rating to user for each movies that he didn't watch or rated.

TYPES OF KNN ALGORITHM(SURPRISE)

- KNNBasic is a basic collaborative filtering algorithm.
- KNNWithMeans is basic collaborative filtering algorithm, taking into account the mean ratings of each user.
- KNNBaseline is a basic collaborative filtering algorithm taking into account a baseline rating.

Recommedation:

1. Now we extract the data in the testset using data.build_anti_testset(). The build_anti_testset return a list of ratings that can be used as a testset in the test() method the ratings are all ratings that are not in the testset.

2. Now we are predicted the rating for the movies that are not rated by the user. There is function model.test which returns the dataset that contains the predicted rating for all the movies. After this we created a function which returns predicted for all the movies in descending order to user.

CONTENT-BASED FILTERING

The system learns to recommend items that are similar to the ones that the user liked in the past. It exploits the content of data items to predict its relevance based on the user's profile. For instance if the active user has rated positively a news article in the sports section, the system can learn to recommend other news articles in the same section.

Content-based filtering is based on the profile of the user's preference and the item's description. In CBF to describe items we use keywords apart from user's profile to indicate user's preferred liked or dislikes. In other words CBF algorithms recommend those items or similar to those items that were liked in the past. It examines previously rated items and recommends best matching item.

Approaches

- TF-IDF
- CountVactorizer

TF-IDF

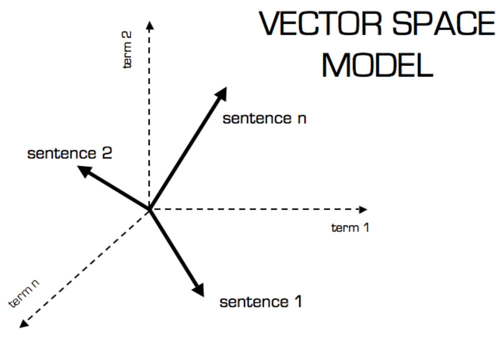
The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systems and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

equation to calculate the TF-IDF score:

$$\mathbf{tfidf}_{i,j} = \mathbf{tf}_{i,j} \times \log\left(\frac{\mathbf{N}}{\mathbf{df}_i}\right)$$

$\mathrm{tf}_{i,j}$ = total number of occurences of i in j
df_i = total number of documents (speeches) containing i
N = total number of documents (speeches)

After calculating TF-IDF scores, how do we determine which items are closer to each other. This is accomplished using the Vector Space Model which computes the proximity based on the angle between the vectors. In this model, each item is stored as a vector of its attributes in an n-dimensional space and the angles between the vectors are calculated to determine the similarity between the vectors. Next, the user profile vectors are also created based on his actions on previous attributes of items and the similarity betwee n an item and a user is also determined in a similar way.



Sentence 2 is more likely to be using Term 2 than using Term 1. Vice-versa for Sentence 1. The method of calculating this relative measure is calculated by taking the cosine of the angle between the sentences and the terms. The ultimate reason behind using cosine is that the value of cosine will increase with decreasing value of the angle between which signifies more similarity. The vectors are length normalized after which they become vectors of length 1 and then the cosine calculation is simply the sum-product of vectors.

You will compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each document. This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document), and each column represents a movie, as before.

In its essence, the TF-IDF score is the frequency of a word occurring in a document, down-weighted by the number of documents in which it occurs. This is done to reduce the importance of words that frequently occur in plot overviews and, therefore, their significance in computing the final similarity score.

Fortunately, scikit-learn gives you a built-in tfidfvectorizer class that produces the TF-IDF matrix in a couple of lines.

IMPLEMENTATION

We do not have a quantitative metric to judge our machine's performance so this will have to be done qualitatively. In order to do so, We use TfidfVectorizer function from scikit-learn, which transforms text to feature vectors that can be used as input to estimator.
We used the Cosine Similarity to calculate a numeric quantity that denotes the similarity between two movies. Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine Similarity Score. Therefore, we used sklearn's linear_kernel instead of cosine_similarities since it is much faster.

Now We have a pairwise cosine similarity matrix for all the movies in the dataset. The next step is to write a function that returns the 20 most similar movies based on the cosine similarity score.

COUNTVECTORIZER

CountVectorizer 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. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors.

CountVectorizer 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.

After performing count vectorizer we have a matrix ,using it we calculate the cosine similarity between two movies as we did in TF-IDF. And then we recommend the top n movie on the basis of similarity score.

EVALUATION

As for any machine learning algorithm, we need to be able to evaluate the performances of our recommender system in order to decide which algorithm fit the best our situation. So here we have used two main methods in order to compare and measure the performance of different approaches of recommendation systems.

1. MEAN ABSOLUTE ERROR (MAE)

It is the difference between the actual value(rating) and the predicted value. As you might have heard a lot about this metric, I won't be covering it. All you have to know is lower the MAE value is, better will be our model.

$$\text{MAE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}} |\underbrace{y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i}_{\text{actual value}}|$$

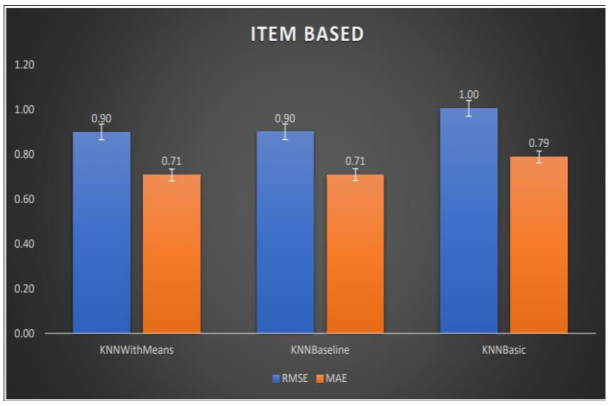
2. ROOT MEAN SQUARED ERROR (RMSE)

RMSE is similar to MAE but the only difference is that the absolute value of the residual(see above image) is squared and the square root of the whole term is taken for comparison.

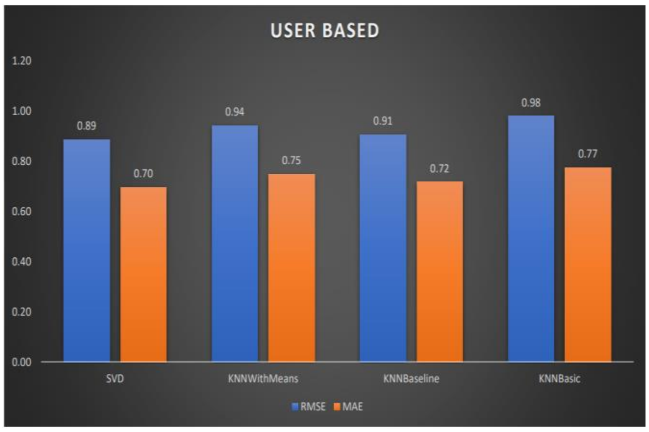
The advantage of using RMSE over MAE is that it penalizes the term more when the error is high. (Note that RMSE is always greater than MAE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

RMSE/MAE COMAPARISON WITH DIFFERENT ALGORITHMS-ITEM BASED



RMSE/MAE COMAPARISON WITH DIFFERENT ALGORITHMS-USER BASED



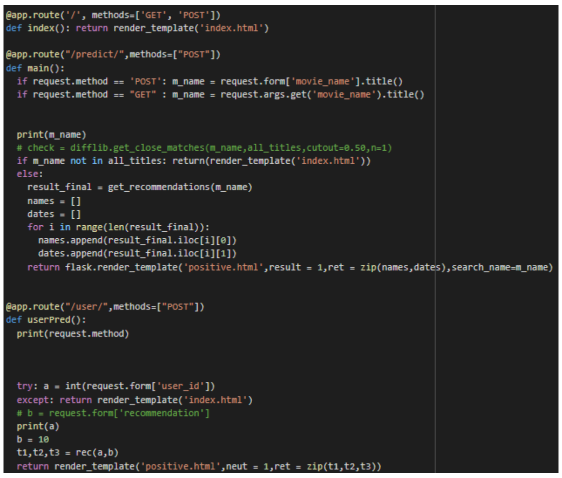
WEB APPLICATION

To recommend movie from the trained model we have used Flask to develop a frontend of the website.

UI Interface:

- User types in the URL for the system on a Web Browser.
- User logs into the system using his `userid` or 'Item id' or 'Movie Title'. This is because we have used three search bars i.e for content based recommendation and user based and item based for collaborative filtering.
- The user chooses from amongst the different types of recommendation systems available.
- If the user choose 'Collaborative Filtering' option, the system calculates similar users or similar items making use of engineering algorithms, and then recommends items to the users based on the most similar user or most similar items.
- If the user choose 'Content based Filtering' option, the system then makes use of the content information to make recommendations.
- The System provides the user with the functionalities like The System provides its users with auto search box, which automatically pulls the movies matching the keywords typed, by the user. The auto search feature is automatically activated after the user has typed 3 characters. After typing movie id or user id or movie titles in search it depends on him which type of recommendation he prefers i.e content based or collaborative filtering after pressing click here button he'll 10 recommendations on the basis of his interest.

UI BACKEND



```
@app.route('/', methods=['GET', 'POST'])
def index(): return render_template('index.html')

@app.route("/predict/",methods=["POST"])
def main():
  if request.method == 'POST': m_name = request.form['movie_name'].title()
  if request.method == "GET" : m_name = request.args.get('movie_name').title()


  print(m_name)
  # check = difflib.get_close_matches(m_name,all_titles,cutout=0.50,n=1)
  if m_name not in all_titles: return(render_template('index.html'))
  else:
    result_final = get_recommendations(m_name)
    names = []
    dates = []
    for i in range(len(result_final)):
      names.append(result_final.iloc[i][0])
      dates.append(result_final.iloc[i][1])
    return flask.render_template('positive.html',result = 1,ret = zip(names,dates),search_name=m_name)


@app.route("/user/",methods=["POST"])
def userPred():
  print(request.method)



  try: a = int(request.form['user_id'])
  except: return render_template('index.html')
  # b = request.form['recommendation']
  print(a)
  b = 10
  t1,t2,t3 = rec(a,b)
  return render_template('positive.html',neut = 1,ret = zip(t1,t2,t3))
```

CONCLUSION

In the time period that was given to us we tried several algorithms and model in order to implement different recommendation system approaches such as Collaborative Filtering and Content based filtering.

In item based collaborative filtering we implement model based algorithms that is KNNWithMeans, KNNBasic and KNNBaseline .We checked their performance using RMSE and MAE from this we came to the conclusion that RMSE and MAE of KNNWithMenas is much batter than other algorithms because of lowest RMSE value.

In user based collaborative filtering we implemented the above same algorithms along with matrix factorization based algorithm SVD (Singular Value Decomposition). In this we checked for their RMSE and MAE values. From this we came to conclusion that performance of SVD is better than other algorithms.

In content based filtering, to detect similarities between movies, we need to vectorize, using countvectorizer. we decided to use CountVectorizer rather than TfIdfVectorizer for one simple reason: I need a simple frequency counter for each word in Genres column. Tf-Idf tends to give less importance to the words that are more present in the entire data. which is not what we want for this application, because every word is important to detect similarity! Once I have the matrix containing the count for each word, we can apply the cosine_similarity function..

ACKNOWLEDGEMENT

This thesis and the whole internship wouldn't be possible without the efforts of many people. It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my respected guide for their valuable guidance, encouragement and help for completing this work. Their useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ah

REFERENCES

1. Google.com
2. https://medium.com/fnplus/evaluating-recommender-systems-with-python-code-ae0c370c90be
3. https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243
4. https://surprise.readthedocs.io/en/stable/trainset.html
5. https://surprise.readthedocs.io/
6. https://www.kaggle.com/rounakbanik/movie-recommender-systems
7. https://www.edureka.co/blog/background-image-in-html/
8. https://towardsdatascience.com/building-and-testing-recommender-systems-with-surprise-step-by-step-d4ba702ef80b