Decision Support Systems

Shivaram Rammohan
202202968@post.au.dk
Department of Computer Engineering
Aarhus University
Aarhus, Denmark

Instructors:

Christian Fische Pedersen & Christian Marius Lillelund

Group 11

Project -1 Recommender systems

Contents

Introduction:

Recommender systems have become an integral part of our digital experiences, helping us discover new movies, music, products, and more. One popular approach to building recommender systems is collaborative filtering. In this context, collaborative filtering refers to the process of predicting user preferences or item ratings by leveraging the collective knowledge of a community of users.

In this project, we are focusing on building a movie recommender system based on collaborative filtering and using different models to recommend movies whereas also train the models and put them against each other to see the results.

Project description:

The project is done for the recommender systems to recommend movies to the end user based on his existing data and or as a new user. We will develop a basic system and then train different models too check for their accuracy, they are compared against the state-of-the-art models in the market and then evaluated on their accuracy.

Report structure:

The report will contain the first theory on how the data is obtained and what kind of models have been used and how its accuracy stands against the existing system and the other models that we have implemented against the data collected. The results and observations are then discussed to find the best model for the recommender system and display their results.

Repository:

Material related to this project is stored in a GitHub account where the report and source code can be found. And, in the ZIP file that has been submitted for this project.

Project Background:

The project discusses recommending movies to the users based on their history or based on the rating if they are a new user. The Recommender system is a complex system that most modern-day entertainment and consumer websites use them to recommend items to users that are relevant and what the system expects the user to like.

Approach:

The Movie recommender system is first loaded with the movies data from the dataset of 100k as we are limited to hardware to train the system. The System is then built to be used by various model to train the basic model and then it is tested across multiple models from the previous steps and finally we can see the prediction of the models and then see which models are the best in recommending movies to the user.

Basic architecture used:

The basic flow in creating the recommender system is that we as always load the dataset using pandas' data frames and then choose a model to train and then make the model recommendations to be listed to the end user to be validated along with accuracy metrics to determine its accuracy.

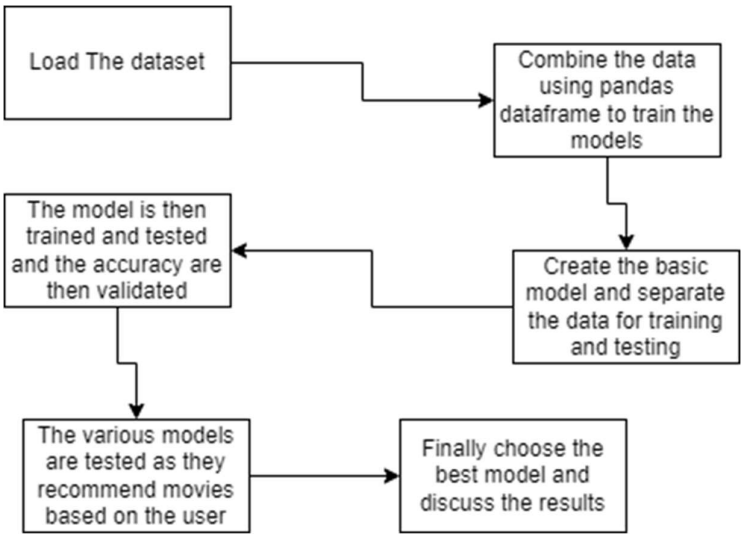

Figure 1: Flowchart of the basic strategy used for training a recommender system.

Dataset:

The Dataset used here is from Movie lens which has 100836 ratings and 3683 tag applications across 9742 movies. The data is a small subset of the main data set from movielens as hardware is limited to only google colab this dataset is used and has proven to be sufficient for training models to a decent level of accuracy.

```
column_names = ["user_id", "movie_id", "ratings", "timestamp"]
df_ratings= pandas.read_csv("/content/drive/MyDrive/ml-latest-
small/ratings.csv", names=column_names, skiprows=1)
df_ratings.head()
column_names_movies = ["movie_id", "title", "geners"]
df_movies = pandas.read_csv("/content/drive/MyDrive/ml-latest-
small/movies.csv", names=column_names_movies, skiprows=1)
```

Correlation Matrix:

A correlation matrix is a mathematical matrix that provides information about the correlation between multiple variables. In the context of recommender systems, a correlation matrix is often used to analyze the relationships between items or users based on their interactions or preferences. In a correlation matrix, each row and column represent a variable, which can be items or users in the case of recommender systems. The cells of the matrix contain correlation coefficients that indicate the strength and direction of the relationship between the corresponding variables. The correlation coefficient is a statistical measure that ranges from -1 to 1. A positive correlation coefficient indicates a positive relationship, meaning that when one variable increases, the other variable also tends to increase. Conversely, a negative correlation coefficient indicates a negative relationship, where an increase in one variable is associated with a decrease in the other variable. A correlation coefficient of 0 indicates no linear relationship between the variables.

In the context of recommender systems, the correlation matrix can be used to identify similar items or users based on their interactions or preferences. For item-item collaborative filtering, the correlation matrix helps determine the similarity between items by measuring the correlation between their ratings or interactions by users. Similarly, for user-user collaborative filtering, the correlation matrix measures the similarity between users based on their ratings or interactions with items. Once the correlation matrix is calculated, it can be used to make recommendations. For example, in item-item collaborative filtering, when a user expresses interest in a particular item, the system can recommend similar items based on the correlation values in the matrix. Similarly, in user-user collaborative filtering, the system can recommend items that are highly rated by users who are like the target user based on the correlation matrix.

Figure 2: example for Correlation Matrix

The correlation matrix can be used to determine the closely related item for a user hence in the movie recommender system we load the dataset and then create a correlation among the user and the items used.

```
rating = pandas.DataFrame(df.groupby("title")["ratings"].mean())
rating.head()
rating["number_of_ratings"] =
pandas.DataFrame(df.groupby("title")["ratings"].count())
rating.head()
movie_matrix= df.pivot_table(index="user_id", columns="title",
values="ratings")
movie_matrix.head()
similar_to_forest_grump=movie_matrix.corrwith(forest_grump_ratings)
corr_forest_grump= pandas.DataFrame(similar_to_forest_grump,
columns=["Correlation"])
corr_forest_grump.dropna(inplace=True)
corr_forest_grump.head()
```

Discussion of the implementation:

The correlation matrix is first build from the users' movies and their ratings by which the ratings are sorted in descending such that the higher the rating the more it gets recommended to the users. Thus , we obtain the below output for the matrix we have created in Fig 3.

title	Correlation
Lost & Found (1999)	1.0
Century of the Self, The (2002)	1.0
The 5th Wave (2016)	1.0
Play Time (a.k.a. Playtime) (1967)	1.0
Memories (Memorîzu) (1995)	1.0
Playing God (1997)	1.0
Killers (2010)	1.0
Girl Walks Home Alone at Night, A (2014)	1.0
Tampopo (1985)	1.0
Cercle Rouge, Le (Red Circle, The) (1970)	1.0

Figure 3: Correlation Matrix Output

The Correlation matrix has some limitations such as that the matrix is large and holding that huge amount data is very ineffective when we want to actively recommend users and update the newer movies and their ratings. The major factor here is that the User recommendations are based on the user ratings and their corelation to each other. Thus, newer movies do not get higher up in the recommendation list as you can see in the above Fig 3, we can only get movies that are highly popular and newer movies are not up there in the outputs.

Singular Value Decomposition (SVD) Model:

Singular Value Decomposition (SVD) is a matrix factorization technique commonly used in recommender systems like movie recommendation. SVD helps identify hidden features or factors that contribute to user preferences and item similarities. It decomposes a matrix into three separate matrices, capturing the underlying structure and latent factors of the data.

Mathematically, SVD decomposes a matrix A into three matrices as follows:

$$A = U * \Sigma * V^T$$

U: The left singular matrix represents the user-feature matrix. It captures the relationships between users and latent features.

Σ: The diagonal singular value matrix represents the strength or importance of each latent feature. The diagonal elements represent the singular values, which indicate the significance of the corresponding features.

V^T: The right singular matrix represents the item-feature matrix. It captures the relationships between items and latent features.

The SVD decomposition allows us to approximate the original matrix A using a lower-rank approximation. By keeping only, the most significant singular values and their corresponding singular vectors, we can reduce the dimensionality of the data and capture the most important information. SVD can be applied to the user-item ratings matrix. Each entry of the matrix represents the rating given by a user to a movie. By performing SVD on this matrix, we can identify latent factors that explain the observed ratings and predict unknown ratings.

The resulting matrices U, Σ, and V can be used to make movie recommendations. For example, given a user, we can represent them in the user-feature space by multiplying their row in the U matrix by the corresponding singular values in the Σ matrix. Similarly, we can represent movies in the item-feature space by multiplying their column in the V matrix by the singular values. By comparing the representations of users and movies in the latent feature space, we can make recommendations based on similarity measures such as cosine similarity or Euclidean distance. SVD-based recommendation systems offer several advantages. They can handle sparse and incomplete data, provide personalized recommendations, and capture complex relationships between users and items.

```
# Split the data into training and test sets
trainset, testset = train_test_split(data, test_size=0.2,
random_state=42)

# Train an SVD model on the training set
model = SVD(n_factors=50, reg_all=0.1)
model.fit(trainset)
```

Discussion of the implementation:

In the above the implementation is that we first split the data into 80, 20 for train and test size and the random state parameter ensures reproducibility of the split. The SVD model is then imported from surprise library and is initialized with n_factors=50, which sets the number of latent factors to consider during the decomposition of the user-item ratings matrix. The higher number of latent factors allows for a more expressive model but may also lead to overfitting issue hence we kept it to 50. The reg_all parameter is set to 0.1, which controls the regularization term to prevent overfitting.

Discussion of the results:

The results we have obtained from the above implementation is seen in Fig 4 . The movie is recommendations are for user 23 who has similar movies to Heat and Diehard. The Accuracy of the model is represented in RMSE (Root Mean Square Error) which means the lower it is the better the predictive accuracy, suggesting that the model can capture the underlying patterns in the user-item ratings and make more accurate predictions. Hence, I have used it as a metric of measurement for accuracy.

```
RMSE: 0.8769
Model accuracy: 0.88
-Heat (1995)
-Die Hard (1988)
-Following (1998)
-Looking for Richard (1996)
-Blade Runner (1982)
-City of Lost Children, The (Cité des enfants perdus, La) (1995)
-Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
-Fisher King, The (1991)
-Maltese Falcon, The (a.k.a. Dangerous Female) (1931)
-Usual Suspects, The (1995)
-Monty Python's Life of Brian (1979)
-Postman, The (Postino, Il) (1994)
-Reservoir Dogs (1992)
-Once Upon a Time in China (Wong Fei Hung) (1991)
-L.A. Confidential (1997)
-Equus (1977)
-Fargo (1996)
```

Figure 4: Output for SVD recommender model

Other Models In recommender system:

The Goal of the project is to find the best recommendation system models such that the user can be recommended movies that are related to the user and includes some randomness such that the recommendation can recommend newer movies even when there isn't many ratings and users that have watched that such movie. The Other models used are as below.

WARP (Weighted Approximate-Rank Pairwise):

This model is trained using the WARP loss function. WARP optimizes the model by performing pairwise ranking, where it aims to correctly rank positive items higher than negative items. It is a popular choice for recommendation systems as it can handle implicit feedback data effectively.

Given a user u and a positive item i, we aim to find a negative item j that violates the pairwise ranking assumption.

Let s_ui denote the predicted score for user u and item i, and s_uj denote the predicted score for user u and item j.The WARP loss function is defined as:

loss = -log(sigmoid(s_ui - s_uj))

where sigmoid(x) is the sigmoid function that maps any real value to a value between 0 and 1.The loss function encourages the model to learn parameters that maximize the difference between s_ui and s_uj, ensuring that positive items have higher scores than negative items.

Logistic:

This model uses the logistic loss function. It is suitable for binary classification problems, where the goal is to predict whether a user will like or dislike an item. In the context of a movie recommender system, it can be used to predict whether a user will rate a movie positively or negatively.

Given a user u and two items i and j, the logistic model aims to estimate the probability that the user prefers item i over item j.Let s_ui denote the predicted score for user u and item i, and s_uj denote the predicted score for user u and item j.The logistic function is applied to the score difference as follows:

probability = sigmoid(s_ui - s_uj)

where sigmoid(x) is the logistic function that maps any real value to a value between 0 and 1.The loss function is derived from the maximum likelihood estimation and aims to maximize the likelihood of the observed preferences (interactions) in the training data.

BPR (Bayesian Personalized Ranking):

The BPR model is trained using the BPR loss function. BPR is designed for implicit feedback data and focuses on pairwise ranking. It aims to learn the item preferences of users based on their interaction history. The model learns to rank positive items higher than negative items, without explicitly predicting the ratings.

Given a user u and two items i and j, BPR aims to learn the probability that user u prefers item i over item j.Let s_ui denote the predicted score for user u and item i, and s_uj denote the predicted score for user u and item j.The probability of user u preferring item i over item j is computed using a sigmoid function as follows:

probability = sigmoid(s_ui - s_uj)

Here, the sigmoid function maps the difference between the scores s_ui and s_uj to a value between 0 and 1, representing the probability of preference. The loss function of BPR is defined based on the logarithm of the sigmoid function:

loss = -log(sigmoid(s_ui - s_uj))

The negative logarithm is used to maximize the likelihood of observing the correct ranking.

From the above models the LightFM package we have trained models to recommend the movies and hence we obtain the results in Fig [5] from the code

```
# Train and evaluate multiple models
models = {
    'warp': LightFM(loss='warp'),
    'logistic': LightFM(loss='logistic'),
    'bpr': LightFM(loss='bpr'),
}

best_model = None
best_precision = 0.0

for model_name, model in models.items():
    model.fit(train, epochs=10)
    train_precision, train_recall, test_precision,
test_recall,train_rmse, test_rmse = evaluate_model(model, train, test)

    print(f"Model: {model_name}")
    print("Train Precision:", train_precision)
    print("Train Recall:", train_recall)
    print("Test Precision:", test_precision)
    print("Test Recall:", test_recall)
    print("Train RMSE:", train_rmse)
    print("Test RMSE:", test_rmse)
```

```
Model: warp
Train Precision: 0.3970492
Train Recall: 0.03263555570359702
Test Precision: 0.09589491
Test Recall: 0.025383253154932558
Train RMSE: 4.0585403
Test RMSE: 4.356398456688366
Model: logistic
Train Precision: 0.3918033
Train Recall: 0.03165997553166568
Test Precision: 0.10344828
Test Recall: 0.027905557566127618
Train RMSE: 1.4442754
Test RMSE: 1.4670441745719074
Model: bpr
Train Precision: 0.50786895
Train Recall: 0.043640815019850274
Test Precision: 0.07553366
Test Recall: 0.025166265541505896
Train RMSE: 8.281701
Test RMSE: 8.839198851256242
The Best Model is logistic Precsion :0.1034482792019844
```

Figure 5: Other model's result from LightFM

Conclusion:

The Models that we have used to recommend movies to a user has been done with SVD, WARP, Logistic and BPR models. The SVD has the highest level of recommending movies to the user hence was able to recommend movies that are relative to the user and also was able to recommend movies that the User liked much easily But the issue with SVD was that the optimization for training new models was not sufficient as it always took longer than Models from LightFM as those models in LightFM were quick in training compared to SVD but the movies recommended from logistic model were mostly similar to users and wasn't able to provide a diverse range of movies to users. Hence, From the above results we can see that SVD is better overall than the other models but as always everything has its drawbacks and SVD can be utilized for recommendations that are not constantly needed to be updated, if not Logistic model is preferred.

References:

1. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.
2. Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 426-434).
3. Maciej Kula. (2015). Metadata Embeddings for User and Item Cold-start Recommendations. In Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with ACM Conference on Recommender Systems (pp. 14-21).
4. Rendle, S., Freudenthaler, C., & Schmidt-Thieme, L. (2012). Factorizing personalized markov chains for next-basket recommendation. In Proceedings of the 19th international conference on World wide web (pp. 811-820).