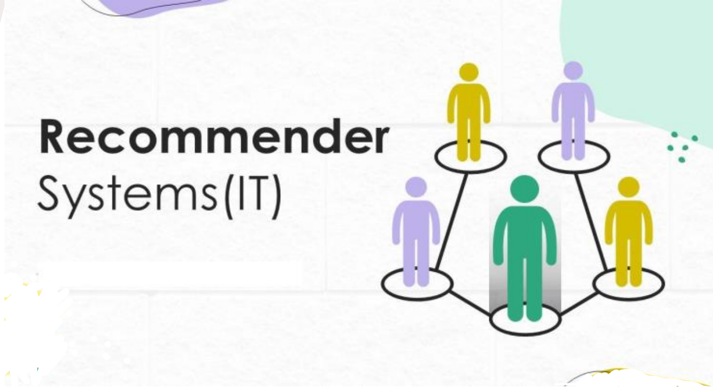
Evaluation and implementation of recommender system

Name: Devanshu Poonith

Student ID: S3970304

Assignment Description: This assignment aims to develop, implement, and evaluate various recommender systems, specifically focusing on movie recommendations. The assignment is divided into three main tasks.

- **Task 1:** User-based Collaborative Filtering
- **Task 2:** Item-based Collaborative Filtering
- **Task 3:** Better Recommender System



Task 1: User-based Collaborative Filtering Using KNN

User-based Collaborative filtering is a type of recommendation system that predicts a user's interest in an item based on other similar users' preferences.

Dataset used: **MOVIELENS DATASET**

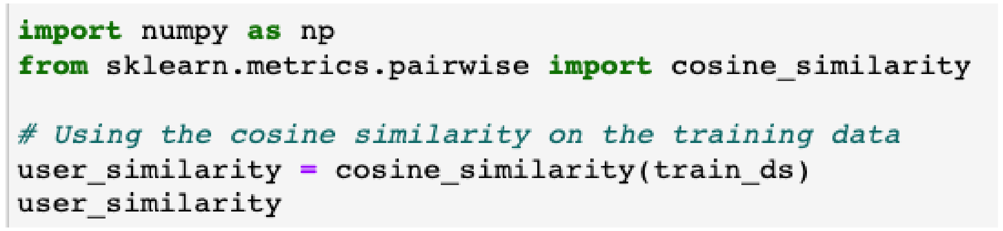
Steps used for the recommender system

- Loading dataset ratings and movies into their dataframe and combining them into merged_df dataframe.
- Filtering the movies to a selective amount with movies having ratings greater than 30 ratings.
- Training and testing the data
- Using cosine similarity metric to calculate the similarity between users.
- Using different values of k ranging from 10,20,30,50,100 for prediction of rating for a randomly chosen user.
- Using root-mean-squared-error as a metric of evaluation.

Task 1: User-based Collaborative Filtering Using KNN

Implementation and experimentation

- The train_test_split function is used to split the data df into training and testing set.
- From sklearn.metrics.pairwise library the cosine_similarity is called.The cosine similarity test is used to find the similarity between the users on the train data.



```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Using the cosine similarity on the training data
user_similarity = cosine_similarity(train_ds)
user_similarity
```

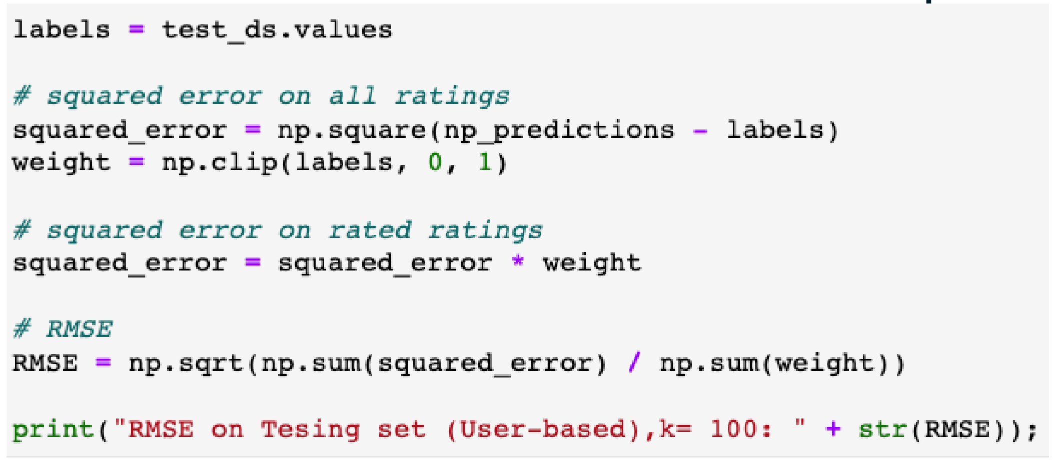
Algorithm used explained

- Use of the prediction algorithm to predict the 'k' most similar active users based on the similarity score
- Then, the predicted rating is used to recommend items not yet rated by the active users

Task 1: User-based Collaborative Filtering Using KNN

Key Results and Findings

Evaluation metric used: **Root-Mean-Squared error**



```
labels = test_ds.values

# squared error on all ratings
squared_error = np.square(np_predictions - labels)
weight = np.clip(labels, 0, 1)

# squared error on rated ratings
squared_error = squared_error * weight

# RMSE
RMSE = np.sqrt(np.sum(squared_error) / np.sum(weight))

print("RMSE on Tesing set (User-based),k= 100: " + str(RMSE));
```

Results for K=10,20,30,50,100 and their respective root mean squared error

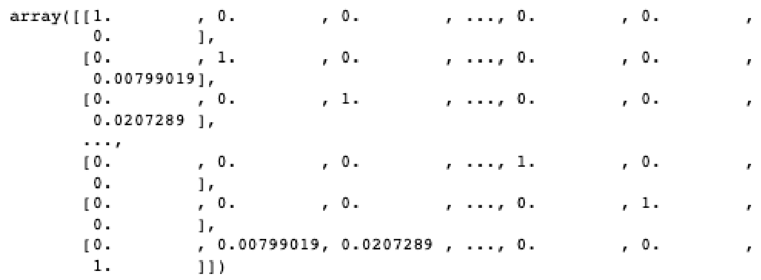
Values of K	Root mean squared error
100	3.7765305365246853
50	3.7761478469737697
30	3.776628083464645
20	3.775956670681962
10	3.776655180721159

Task 2: Item-based Collaborative Filtering Using KNN

Item-based collaborative filtering is a recommendation system approach that identifies relationships between different items in this case movies based on user interactions such as ratings.

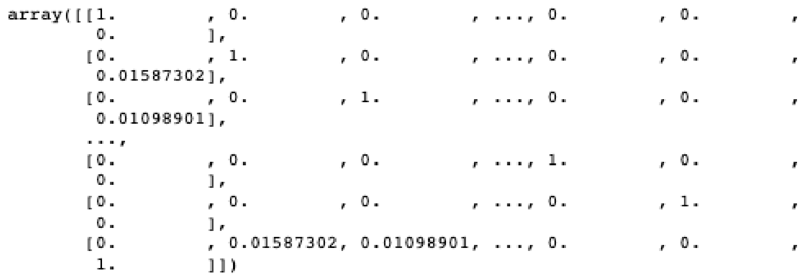
Similarity metric used:

Cosine Similarity



```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 0.        , ..., 0.        , 0.        ,
        0.00799019],
       [0.        , 0.        , 1.        , ..., 0.        , 0.        ,
        0.0207289 ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.00799019, 0.0207289 , ..., 0.        , 0.        ,
        1.        ]])
```

Jaccard Similarity



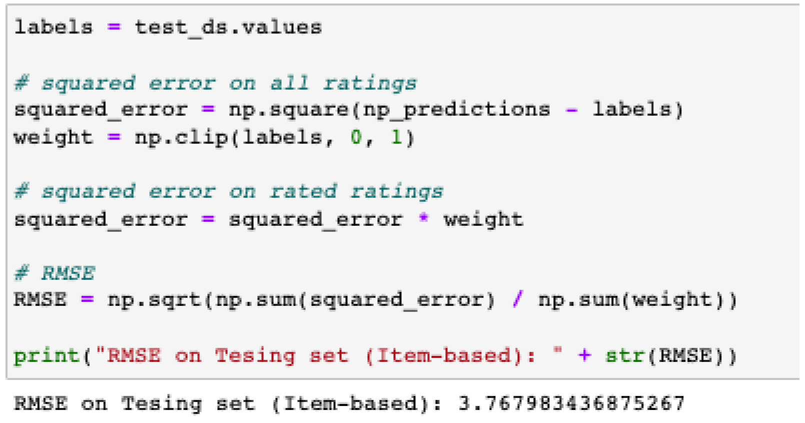
```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 0.        , ..., 0.        , 0.        ,
        0.01587302],
       [0.        , 0.        , 1.        , ..., 0.        , 0.        ,
        0.01098901],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.01587302, 0.01098901, ..., 0.        , 0.        ,
        1.        ]])
```

- Steps used in this recommender system:
- Using the similarity metrics to find the similarity between the items
- Using the prediction algorithm to predict the rating of a user where parameters K is set to 3 and EPSILON is set to $1x10^{-9}$
- Evaluation of the prediction with the actual rating using root mean squared error.

Task 2: Item-based Collaborative Filtering Using KNN

Evaluation metric using root mean squared error

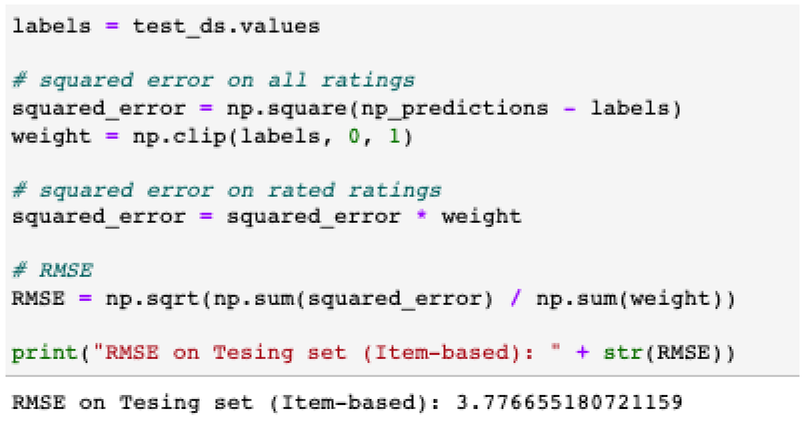
Using Cosine similarity



```
labels = test_ds.values

# squared error on all ratings
squared_error = np.square(np_predictions - labels)
weight = np.clip(labels, 0, 1)

# squared error on rated ratings
squared_error = squared_error * weight

# RMSE
RMSE = np.sqrt(np.sum(squared_error) / np.sum(weight))

print("RMSE on Tesing set (Item-based): " + str(RMSE))
```

```
RMSE on Tesing set (Item-based): 3.767983436875267
```

Using Jaccard Similarity



```
labels = test_ds.values

# squared error on all ratings
squared_error = np.square(np_predictions - labels)
weight = np.clip(labels, 0, 1)

# squared error on rated ratings
squared_error = squared_error * weight

# RMSE
RMSE = np.sqrt(np.sum(squared_error) / np.sum(weight))

print("RMSE on Tesing set (Item-based): " + str(RMSE))
```

```
RMSE on Tesing set (Item-based): 3.776655180721159
```

Conclusion:

Since the root mean squared error from the cosine similarity test is lower than the root mean squared error from the Jaccard Similarity test, we can say that the Cosine similarity test has the better performance matric than Jaccard Similarity

Task 3: Better Recommended System

Option Chosen: **Option 1**

Search and read related publications on recommender systems. Identify/choose one solution that can be used for movie recommendation and will potentially deliver better recommendation quality. The solution must have been published in a peer-reviewed journal or conference. It can be based on KNN or not

Recommender System Used: **Neural Collaborative Filtering**

Definition: Neural Collaborative filtering is a very recent approach toward recommendation system.It uses both the traditional collaborative filtering as well as the concept of neural network to find large scale interaction between users and items.

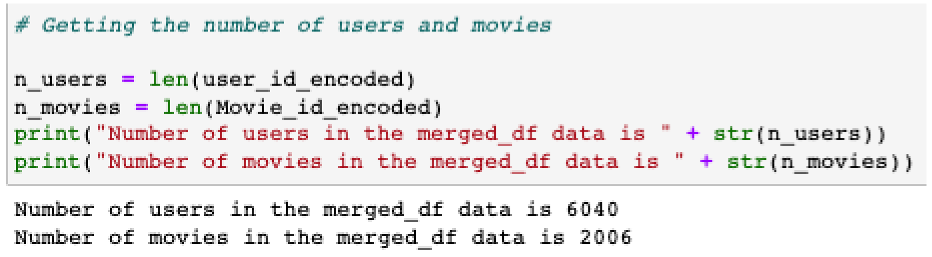
Reference: **He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural Collaborative Filtering.**

Task 3: Better Recommended System

Data Processing and Encoding

Steps used in this process:

- Generation of unique and sequential movieID and userID
- Displaying the total number of unique users and movies.



```
# Getting the number of users and movies

n_users = len(user_id_encoded)
n_movies = len(Movie_id_encoded)
print("Number of users in the merged_df data is " + str(n_users))
print("Number of movies in the merged_df data is " + str(n_movies))
```

```
Number of users in the merged_df data is 6040
Number of movies in the merged_df data is 2006
```

- Splitting the data into training and testing in the ratio of 80:20

Task 3: Better Recommended System

Building Neural Collaborative model

Process involved

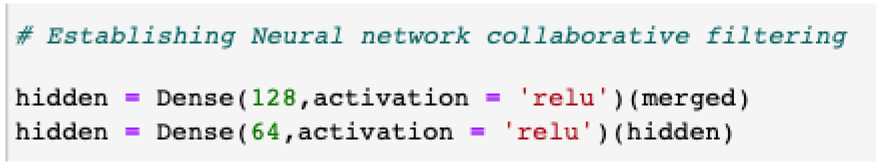
Embeddings: Generation of embeddings for both users and movies

Vector representation: The embeddings are flatten to create vector representation **user_vec** and **movie_vec.**

Merging: The two vectors user_vec and movie_vec are meged to form a feature vector matrix containg the user vector representation and the movie vector representation.

Neural Network Layer:

This part contains two layers for collaborative filtering

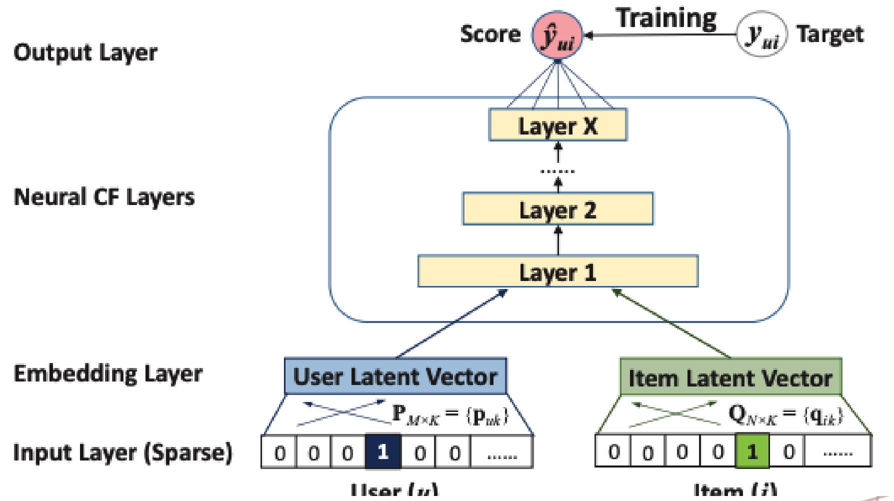


```
# Establishing Neural network collaborative filtering

hidden = Dense(128,activation = 'relu')(merged)
hidden = Dense(64,activation = 'relu')(hidden)
```

The parameter of the dense layer is set to relu for the

activation funtion and with the first line creating 128 neurons

And the second line of code creating 64 neurons.



Task 3: Better Recommended System

Training the Neural Collaborative Filtering model and Making Predictions

When compiling the model:

- We use the Mean Squared Error as the loss function and the Adam Optimizer

```
model.compile(optimizer = Adam(0.001), loss = 'mean_squared_error')
```

- We fit the model to the trained userID and movieID and the rating and used the trained model to predict the ratings for the user-item pair.

```
history = model.fit(
    [df_train["UserID"], df_train["MovieID"]],
    df_train["Rating"],
    batch_size=2048,
    epochs=10,
    verbose=1,
    validation_split=0.2,
)
```

- Predicting rating for user using the model.predict funtion on the test data

	UserID	MovieID	Rating	Timestamp	MovieTitle	MovieGenre
908725	2785	3185	3	972988761	Snow Falling on Cedars (1999)	Drama

```
The predicted rating for user 2785 and movie 3185 is 2.9553656578063965
```