Implementing a KNN-Based Recommendation System

Steps:

1) **Exploring the data:**
   a) We started off by loading the data, the data in our files had no column names so we named each column by the information given in the **README** file, such that we can access the data as dataframes in **Pandas** library.

2) **Handling missing values:**
   a) Checking if there were any missing values, by calling **{dataframe}.isnull().sum()** it retrieves each data column and the number of missing values in it.

   b) In the item (movies) dataframe the “video_release_date” column was completely empty, so it won’t affect the data if we dropped it. The “release_date” column was missing 1 value, we extracted the most common year using **mode()[0]** to return the first most common value.

   c) Same in “IMDB_URL” column it had 3 missing values and we got the most common URL.

3) **Data types and summary statistics:**
   a) Using the **dataframe.describe()** method returns description of the data in the dataframe. Since our data has numerical values, the description contains count, mean, std, min, 25%, 50%, 75% and the max.

   b) Using **dataframe.dtypes** prints the data types in each column in the dataframe.

4) **Convert the data into a User-Item Interaction Matrix:**
   a) We will convert the data in file “u.data” to a “**user_item_matrix**” where its rows = users, columns = movies, values = ratings. It represents each user and his rating to each movie in the data, and if no rating it displays NaN.

   b) Next step is to normalize the “**user_item_matrix**” to ensure fair comparisons using this rule: (user_item_matrix - user_item_matrix.mean()) / user_item_matrix.std().

5) **Cosine Similarity for movie recommendations:**
   a) Applying the cosine similarity to the “**normalized_user_item_matrix**” to get the user similarity matrix which defines the levels of similarity between each user’s rating with all other users.

6) **Given a target user recommend the top-rated movies from their nearest neighbors**
   a) First, we define the function “**get_nearest_neighbors**” to retrieve the nearest k neighbors to the target user, using the “**user_similarity**” dataframe we retrieve the scores between the target user and all others, and then drop the target user similarity value since it has similarity 1 with itself then use **nlargest(k)** that retrieves the top k similarity scores.

   b) Second, we recommend the “**high_rated_movies**“ by these neighbors using the function “**recommend movies**” we get the ratings of the retrieved neighbors from the “**get_nearest_neighbors**” function, then we filter those ratings to the ones above or equal to 3.5.

      We retrieve the user rated movie and check if any of them belong to the filtered movies, if so remove them since he already watched them, if there are no movies left to recommend it’s better to increase

the neighbors' number or decrease the rating threshold. The last part is retrieving the movies titles from their ids.

7) **Implement the KNN Algorithm for Recommendations:**
   a) First, we load the training and testing data, since there are many base and test files, so we'll split them into folds, train and test the data on each fold.

   b) Second, we preprocess the data and convert it into a matrix using the **pivot()** method and return X_train, y_train, X_test, y_test.

   c) Third, we train the knn model and evaluate it using accuracy, precision and recall metrics to evaluate the quality of recommendations.

   d) Fourth, we test different values of k to see its effect on recommendations for example k=[1,3,5].

8) **Discuss the effectiveness of the recommendations:**
   a) Accuracy analysis:
      i) Accuracy is consistently high across all folds and different values of k, ranging from 0.9605 to 0.9870.
      ii) This suggests that the model is making correct predictions most of the time.

   b) Precision analysis:
      i) Precision increases as k increases (0.1426 -> 0.2118 -> 0.2481).
      ii) This suggests that using a higher k improves the model ability to make fewer false positive recommendations.
      iii) In recommendation systems, precision is crucial because it reflects how many recommended movies are actually relevant. A low precision indicates many irrelevant recommendations.

c) Precision analysis:
   i) Recall decreases slightly as k increases (0.7046 -> 0.6961 -> 0.6889).
   ii) A lower recall means that the model is missing some relevant recommendations.
   iii) This is a common trade-off as precision increases recall tends to decrease, since recall is fairly high the model is retrieving a reasonable number of relevant movies.

9) Effectiveness of this approach:

a) Your approach is effective in terms of accuracy, but recommendation systems require balancing precision and recall.

b) The choice of k affects whether you prioritize making more recommendations (recall) or making only highly relevant ones (precision).

c) If precision is low, users might receive too many irrelevant recommendations, which reduces trust in the system.

d) If recall is too low, users might miss out on good recommendations, leading to dissatisfaction.