Music Recommendation System

Milestone 2

Now that we have explored the data, let's apply different algorithms to build recommendation systems

Note: Use the shorter version of the data i.e. the data after the cutoffs as used in Milestone 1.

Popularity-Based Recommendation Systems

Let's take the count and sum of play counts of the songs and build the popularity recommendation systems on the basis of the sum of play counts.

```
#Calculating average play_count
average_count = df_final._____________ #Hint: Use groupby function on the song_id column.

#Calculating the frequency a song is played.
play_freq = df_final.________________#Hint: Use groupby function on the song_id column
```

```
#Making a dataframe with the average_count and play_freq
final_play = pd.DataFrame({'avg_count':_________, 'play_freq':______})
final_play.head()
```

```

```

Now, let's create a function to find the top n songs for a recommendation based on the average play count of song. We can also add a threshold for a minimum number of playcounts for a song to be considered for recommendation.

```
#Build the function for finding top n songs
```

```
#Recommend top 10 songs using the function defined above
```

User User Similarity-Based Collaborative Filtering

To build the user-user-similarity based and subsequent models we will use the "surprise" library.

```
#Install the surprise package using pip. Uncomment and run the below code to do the same.
#!pip install surprise
```

```
# Import necessary libraries
# To compute the accuracy of models
from surprise import accuracy

# class is used to parse a file containing play_counts, data should be in structure - user; item ; play_count
from surprise.reader import Reader

# class for loading datasets
from surprise.dataset import Dataset

# for tuning model hyperparameters
from surprise.model_selection import GridSearchCV

# for splitting the data in train and test dataset
from surprise.model_selection import train_test_split

# for implementing similarity-based recommendation system
from surprise.prediction_algorithms.knns import KNNBasic

# for implementing matrix factorization based recommendation system
from surprise.prediction_algorithms.matrix_factorization import SVD

# for implementing KFold cross-validation
from surprise.model_selection import KFold

#For implementing clustering-based recommendation system
from surprise import CoClustering
```

Some useful functions

The below is the function to calculate precision@k and recall@k, RMSE and F1_Score@k to evaluate the model performance.

Think About It: Which metric should be used for this problem to compare different models?

```
#The function to calulate the RMSE, precision@k, recall@k and F_1 score.
def precision_recall_at_k(model, k=30, threshold=1.5):
    """Return precision and recall at k metrics for each user"""

    # First map the predictions to each user.
    user_est_true = defaultdict(list)

    #Making predictions on the test data
    predictions=model.test(testset)

    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():

        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Number of recommended items in top k
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

        # Number of relevant and recommended items in top k
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                              for (est, true_r) in user_ratings[:k])

        # Precision@K: Proportion of recommended items that are relevant
        # When n_rec_k is 0, Precision is undefined. We here set Precision to 0 when n_rec_k is 0.

        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0

        # Recall@K: Proportion of relevant items that are recommended
        # When n_rel is 0, Recall is undefined. We here set Recall to 0 when n_rel is 0.

        recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 0

    #Mean of all the predicted precisions are calculated.
    precision = round((sum(prec for prec in precisions.values()) / len(precisions)),3)
    #Mean of all the predicted recalls are calculated.
    recall = round((sum(rec for rec in recalls.values()) / len(recalls)),3)

    accuracy.rmse(predictions)
    print('Precision: ', precision) #Command to print the overall precision
    print('Recall: ', recall) #Command to print the overall recall
    print('F_1 score: ', round((2*precision*recall)/(precision+recall),3)) # Formula to compute the F-1 score.
```

Think About It: In the function precision_recall_at_k above the threshold value used is 1.5. How precision and recall are affected by chaning the threshold? What is the intuition behind using the threshold value 1.5?

```
# Instantiating Reader scale with expected rating scale
reader = Reader(rating_scale=____) #use rating scale (0,5)

# loading the dataset
data = Dataset.load_from_df(df_final[[______, _______, ____]], reader) #Take only "user_id","song_id", and "play_count"

# splitting the data into train and test dataset
trainset, testset = train_test_split(data, test_size=____, random_state=42) # Take test_size=0.4
```

Think About It: How changing the test size would change the results and outputs?

```
#Build the default user-user-similarity model
sim_options = {'name': _________,
               'user_based':______}

#KNN algorithm is used to find desired similar items.
sim_user_user = KNNBasic(_________) #use random_state=1

# Train the algorithm on the trainset, and predict play_count for the testset
sim_user_user.fit(_________)

# Let us compute precision@k, recall@k, and f_1 score with k =30.
precision_recall_at_k(_____________) #Use sim_user_user model
```

Observations and Insights:_

```
#predicting play_count for a sample user with a listened song.
sim_user_user.predict(_____, ______, r_ui=2, verbose=True) #use user id 6958 and song_id 1671
```

```
#predicting play_count for a sample user with a song not-listened by the user.
sim_user_user.predict(____,_____, verbose=True) #Use user_id 6958 and song_id 3232
```

Observations and Insights:_

Now, let's try to tune the model and see if we can improve the model performance.

```
# setting up parameter grid to tune the hyperparameters
param_grid = {'k': [10, 20, 30], 'min_k': [3, 6, 9],
              'sim_options': {'name': ["cosine",'pearson',"pearson_baseline"],
                              'user_based': [True], "min_support":[2,4]}
              }

# performing 3-fold cross validation to tune the hyperparameters

# fitting the data
gs.fit(______) #Use entire data for GridSearch

# best RMSE score

# combination of parameters that gave the best RMSE score
```

```
# Train the best model found in above gridsearch.
```

Observations and Insights:_

```
#Predict the play count for a user who has listened to the song. Take user_id 6958, song_id 1671 and r_ui=2
```

```
#Predict the play count for a song that is not listened by the user (with user_id 6958)
```

Observations and Insights:__

Think About It: Along with making predictions on listened and unknown songs can we get 5 nearest neighbors (most similar) to a certain user?

```
#Use inner id 0.
```

Below we will be implementing a function where the input parameters are -

- data: a **song** dataset
- user_id: a user id **against which we want the recommendations**
- top_n: the **number of songs we want to recommend**
- algo: the algorithm we want to use **for predicting the play_count**
- The output of the function is a **set of top_n items** recommended for the given user_id based on the given algorithm

```
def get_recommendations(data, user_id, top_n, algo):

    # creating an empty list to store the recommended product ids
    recommendations = []

    # creating an user item interactions matrix
    user_item_interactions_matrix = data.pivot_table(______________)

    # extracting those business ids which the user_id has not visited yet
    non_interacted_products = user_item_interactions_matrix.loc[user_id]
[user_item_interactions_matrix.loc[user_id].isnull()].index.tolist()

    # looping through each of the business ids which user_id has not interacted yet
    for item_id in non_interacted_products:

        # predicting the ratings for those non visited restaurant ids by this user
        est = ___________________

        # appending the predicted ratings
        recommendations.append(_______________)

    # sorting the predicted ratings in descending order
    recommendations.sort(key=lambda x: x[1], reverse=True)

    return recommendations[:top_n] # returing top n highest predicted rating products for this user
```

```
#Make top 5 recommendations for user_id 6958 with a similarity-based recommendation engine.
recommendations =_____________________________
```

```
#Building the dataframe for above recommendations with columns "song_id" and "predicted_ratings"
pd.DataFrame(_________________)
```

Observations and Insights:__

Correcting the play_counts and Ranking the above songs

```
def ranking_songs(recommendations, final_rating):
  # sort the songs based on play counts
  ranked_songs = final_rating.loc[[items[0] for items in recommendations]].sort_values('play_freq',
ascending=False)[['play_freq']].reset_index()

  # merge with the recommended songs to get predicted play_count
  ranked_songs = ranked_songs.merge(pd.DataFrame(recommendations, columns=[__________, ___________]),
on=____________, how='inner')

  # rank the songs based on corrected play_counts
  ranked_songs['corrected_ratings'] = ranked_songs['predicted_ratings'] - 1 / np.sqrt(ranked_songs['play_freq'])

  # sort the songs based on corrected play_counts
  ranked_songs = ________________

  return ranked_songs
```

Think About It: In the above function to make the correction in the predicted play_count a quantity 1/np.sqrt(n) is subtracted. What is the intuition behind it? Is it also possible to add this quantity instead of subtracting?

```
#Applying the ranking_songs function on the final_play data.
```

Observations and Insights:__

Item Item Similarity-based collaborative filtering recommendation systems

```
#Apply the item-item similarity collaborative filtering model with random_state=1 and evaluate the model performance.
```

Observations and Insights:__

```
#predicting play count for a sample user_id 6958 and song (with song_id 1671) heard by the user.
```

```
#Predict the play count for a user that has not listened to the song (with song_id 1671)
```

Observations and Insights:__

```
#Apply grid search for enhancing model performance

# setting up parameter grid to tune the hyperparameters
param_grid = {'k': [10, 20, 30], 'min_k': [3, 6, 9],
              'sim_options': {'name': ["cosine",'pearson',"pearson_baseline"],
                              'user_based': [False], "min_support":[2,4]}
              }

# performing 3-fold cross validation to tune the hyperparameters
gs =

# fitting the data
gs.fit(_____)

# find best RMSE score

# Extract the combination of parameters that gave the best RMSE score
```

Think About It: How do the parameters affect the performance of the model? Can we improve the performance of the model further? Check the list of hyperparameter here.

```
#Apply the best modle found in the grid search.
```

Observations and Insights:__

```
#Predict the play_count by a user(user_id 6958) for the song (song_id 1671)
```

```
#predicting play count for a sample user_id 6958 with song_id 3232 which is not heard by the user.
```

Observations and Insights:__

```
#Find five most similar users to the user with inner id 0
```

```
#Making top 5 recommendations for user_id 6958 with item_item_similarity-based recommendation engine.
recommendations =
```

```
#Building the dataframe for above recommendations with columns "song_id" and "predicted_play_count"
pd.DataFrame(_______________)
```

```
#Applying the ranking_songs function.
```

Observations and Insights:_

Model Based Collaborative Filtering - Matrix Factorization

Model-based Collaborative Filtering is a **personalized recommendation system**, the recommendations are based on the past behavior of the user and it is not dependent on any additional information. We use **latent features** to find recommendations for each user.

```
# Build baseline model using svd
```

```
# Making prediction for user (with user_id 6958) to song (with song_id 1671), take r_ui=2
```

```
# Making prediction for user who has not listened the song (song_id 3232)
```

Improving matrix factorization based recommendation system by tuning its hyperparameters

```
# set the parameter space to tune
param_grid = {'n_epochs': [10, 20, 30], 'lr_all': [0.001, 0.005, 0.01],
              'reg_all': [0.2, 0.4, 0.6]}

# performe 3-fold gridsearch cross validation
gs =

# fitting data

# best RMSE score

# combination of parameters that gave the best RMSE score
```

Think About It: How do the parameters affect the performance of the model? Can we improve the performance of the model further? Check the available hyperparameters here.

```
# Building the optimized SVD model using optimal hyperparameters
```

Observations and Insights:_

```
#Using svd_algo_optimized model to recommend for userId 6958 and song_id 1671.
```

```
#Using svd_algo_optimized model to recommend for userId 6958 and song_id 3232 with unknown baseline rating.
```

Observations and Insights:_

```
# Getting top 5 recommendations for user_id 6958 using "svd_optimized" algorithm.
```

```
#Ranking songs based on above recommendations
```

Observations and Insights:_

Cluster Based Recommendation System

In **clustering-based recommendation systems**, we explore the **similarities and differences** in people's tastes in songs based on how they rate different songs. We cluster similar users together and recommend songs to a user based on play_counts from other users in the same cluster.

```
# Make baseline clustering model
```

```
#Making prediction for user_id 6958 and song_id 1671.
```

```
#Making prediction for user (userid 6958) for a song(song_id 3232) not heard by the user.
```

Improving clustering-based recommendation system by tuning its hyper-parameters

```
# set the parameter space to tune
param_grid = {'n_cltr_u':[5,6,7,8], 'n_cltr_i': [5,6,7,8], 'n_epochs': [10,20,30]}

# performing 3-fold gridsearch cross validation

# fitting data

# best RMSE score

# combination of parameters that gave the best RMSE score
```

Think About It: How do the parameters affect the performance of the model? Can we improve the performance of the model further? Check the available hyperparameters here.

```
# Train the tuned Coclustering algorithm
```

Observations and Insights:_

```
#Using co_clustering_optimized model to recommend for userId 6958 and song_id 1671.
```

```
#Use Co_clustering based optimized model to recommend for userId 6958 and song_id 3232 with unknown baseline rating.
```

Observations and Insights:_

Implementing the recommendation algorithm based on optimized CoClustering model

```
#Getting top 5 recommendations for user_id 6958 using "Co-clustering based optimized" algorithm.
clustering_recommendations = _______
```

Correcting the play_count and Ranking the above songs

```
#Ranking songs based on above recommendations
```

Observations and Insights:_

Content Based Recommendation Systems

Think About It: So far we have only used the play_count of songs to find recommendations but we have other information/features on songs as well. Can we take those song features into account?

```
df_small=df_final
```

```
# Concatenate the "title","release","artist_name" columns to create a different column named "text"
```

```
#Select the columns 'user_id', 'song_id', 'play_count', 'title', 'text' from df_small data

#drop the duplicates from the title column

#Set the title column as the index

# see the first 5 records of the df_small dataset
```

```
# Create the series of indices from the data
indices =_______________
indices[:5]
```

```
#Importing necessary packages to work with text data
import nltk
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
import re
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

We will create a **function to pre-process the text data:**

```
# Function to tokenize the text
def tokenize(text):
    text = re.sub(r"[^a-zA-Z]"," ",text.lower())
    tokens = word_tokenize(text)
    words = [word for word in tokens if word not in stopwords.words(____)] #Use stopwords of english
    text_lems = [WordNetLemmatizer().lemmatize(lem).strip() for lem in words]

    return text_lems
```

```
#Create tfidf vectorizer

# Fit_transfrom the above vectorizer on the text column and then convert the output into an array.
```

```
# Compute the cosine similarity for the tfidf above output
```

Finally, let's create a function to find most similar songs to recommend for a given song

```
# function that takes in song title as input and returns the top 10 recommended songs
def recommendations(title, similar_songs):

    recommended_songs = []

    # gettin the index of the song that matches the title
    idx = indices[indices == title].index[0]

    # creating a Series with the similarity scores in descending order
    score_series = pd.Series(similar_songs[idx]).sort_values(ascending = False)

    # getting the indexes of the 10 most similar songs
    top_10_indexes = list(score_series.iloc[1:11].index)
    print(top_10_indexes)

    # populating the list with the titles of the best 10 matching songs
    for i in top_10_indexes:
        recommended_songs.append(list(df_small.index)[i])

    return recommended_songs
```

Recommending 10 songs similar to Learn to Fly

```
# Make the recommendation for the song with title 'Learn To Fly'
```

Observations and Insights:_

Conclusion and Recommendations:

- **Refined Insights -** What are the most meaningful insights from the data relevant to the problem?
- **Comparison of various techniques and their relative performance -** How do different techniques perform? Which one is performing relatively better? Is there scope to improve the performance further?
- **Proposal for the final solution design -** What model do you propose to be adopted? Why is this the best solution to adopt?