SCHOOL OF IT AND ENGINEERING

PROJECT REPORT

for CSCI-4734 Machine Learning

Project title: ***"MovRec"***

Team Member 1: **Ali Gasimov / (CS)**

Team Member 2: **Safahat Sardarov / (CS)**

Team Member 3: **Aykhan Nazimzada / (CS)**

Baku 2020

MovRec

What is Recommender System:

Goal of the recommender system is to provide users with information that they are most likely to be relevant from other users' ratings by using various patterns in a dataset. The algorithm finds the most appropriate items and provide to users which they would rate highly. Today, most of the e-commerce websites own different recommender systems. For instance, when you enter to "Amazon", you encounter with several items are being recommended to you according to other users ratings, your previous purchases, or your search history. In addtion, "Netflix" also suggesting movies to you by its recommender system. This kind of systems are also being used by music platforms such as "Spotify" and "Deezer".

Project/Methods Overview:

Our project is to build a movie recommender system. There are different algorithms of recommender systems. In project, we have used Item-based Collaborative Filtering, Used-based Collaborative Filtering, and Content-based Filtering. For IBCF and UBCF, implemented algorithms are Cosine Similarity, Matrix Factorization, and K-Nearest Neighbors. For each dataset, we get different results according to our dataset. In the web application, we have picked UBCF together with KNN algorithm because of obtained results. Accuracy and comparisons of algorithms will be discussed in upcoming sections.

Dataset:

We would like to build a recommender system for Azerbaijanian movies. However, we could not find a suitable data for Azerbaijanian movies. For instance, the datas on 'tvseans.az' are rated not by users. It just shows IMDB ratings. With this kind of dataset, it is not possible to build UBCF recommender system. For IBCF, we also need features for movies which dataset does not contain.

In this project, we are going to use "MovieLens" dataset to build recommender system for movies. This dataset contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. All this datas are collected by University of Minnesota. Dataset is allowed to for educational and researchal purposes. Almost all junior Machine Learning projects based on movies are using MovieLens dataset.

In the end, we developed web application using Django. It also has Admin panel to manipulate data. Data is transferred to SQLite3. For website front-end, HTML, CSS, and BootStrap is used. Application can be run using local host.

Algorithms:

In this section, we will talk about several types of Recommendation systems, their implemented algorithms, advantages and disadvantages with together accuracy as well.

Content-based Filtering

Content-based filtering approaches utilize a series of discrete, pre-tagged characteristics of an item in order to recommend additional items with similar properties. As you see from description, we need pre-tagged features of movies. For our dataset, we have ‘genres’ and ‘tags’. However only small part of movies have been tagged in dataset whichd does not help content based-recommendation system in this case. What we have left is ‘genres’ column.

For building content-based recommendation system, we will use from “sklearn” library “RandomForestClassifier”. First we split data into train and test dataset, and scale it using “StandardScaler”. Last step is fitting our data into model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 500, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

Let’s look at the result with confusion matrix:

Confusion Matrix	Actual Positive	Actual Negative
Predicted Positive	1517	6066
Predicted Negative	1161	11257

What we derive from confusion matrix is that out of 7583 movies user likes just 1517 movies and user likes 1161 movies from 12418 unsuggested movies. So our accuracy score is $\frac{12774}{20001}$ x 100 = 64% which is quite low.

To sum up, having 1 feature is not enough for content-based recommendation systems. User likes number of comedy films doesn’t mean that he/she will like all comedy films. Here, we need tags for movies or comments made by users. For all the reasons stated, implementing content-based filtering with our dataset is not a good choice.

Item-based Collaborative Filtering

Collaborative filtering approaches build a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in.

Item-based collaborative filtering is a model-based algorithm for making recommendations. In the algorithm, the similarities between different items in the dataset are calculated by using one of a number of similarity measures, and then these similarity values are used to predict ratings for user-item pairs not present in the dataset.

For IBCF, we have implemented Cosine Similarity, Matrix Factorization, and K-Nearest Neighbors algorithms. Cosine similarity measures similarity between 2 nonzero vectors. If degree is close to 0, then cosine similarity is 1, and vice-versa.

```
def sim_movies_to(title):
    count = 1
    print('Similar movies to {} are :'.format(title))
    for item in item_sim_df.sort_values(by = title, ascending = False).index[1:11]:
        print('No. {} : {}'.format(count, item))
        count += 1

sim_movies_to('Avengers, The (2012)')
```

Here, we simply find similar movies to “Avengers” movie using Cosine Similarity algorithm.

Another method is called Matrix Factorization. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

```
def sim_movies_to(movieId):
    count = 1
    movieIndex = movies_df.index[movies_df['movieId'] == movieId]
    print('Similar movies to {} are :'.format(movies_df.loc[movieIndex].title))
    for item in item_sim_df.sort_values(by = movieId, ascending = False).index[1:11]:
        itemIndex = movies_df.index[movies_df['movieId']==item]
        print('No. {} : {}'.format(count, movies_df.loc[itemIndex].title))
        count += 1

sim_movies_to(89745)
```

In this piece of code, Matrix Factorization function finds similar movies to movie with id 89745.

Last method is called K-Nearest Neighbors. KNN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. Distance functions decides closest neighbors. One of the widely used function is Euclidean distance function and cosine metrics which we have used.

```
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:'.format(movieRatings.index[movie]))
    else:
        print('{0} : {1}, with distances of {2}:'.format(i, movieRatings.index[indices.flatten()[i]], distances.flatten()[i]))
```

Result of this code is most closest movies to our input.

IBCF performs much better than CBF. However our movies 100,000 ratings given by 600 users which is significant numbers to consider. So we will benefit from it using User-based Collaborative Filtering in the next section.

User-based Collaborative Filtering

The method identifies users that are similar to the queried user and estimate the desired rating to be the weighted average of the ratings of these similar users.

For, UBCF we can also implement 3 methods used for IBCF which are Cosine Similarity, Matrix Factorization, and K-Nearest Neighbors algorithms.

```
def recommendation(user):
    if user not in movieRatings.columns:
        return('No data available on this User')

    sim_user = user_sim_df.sort_values(by = user, ascending = False).index[1:11]
    best = []

    for i in sim_user:
        max_score = movieRatings.loc[:, i].max()
        best.append(movieRatings[movieRatings.loc[:, i] == max_score].index.tolist())

    user_seen_movies = movieRatings[movieRatings.loc[:, user] > 0].index.tolist()

    for i in range(len(best)):
        for j in best[i]:
            if(j in user_seen_movies):
                best[i].remove(j)

    most_common = {}

    for i in range(len(best)):
        for j in best[i]:
            if j in most_common:
                most_common[j] += 1
            else:
                most_common[j] = 1

    sorted_list = sorted(most_common.items(), key = operator.itemgetter(1), reverse = True)
    return(sorted_list)

recommendation(2)
```

In comparison to IBCF, Cosine Similarity takes user as an input and based on user's previous ratings recommends movies that user hasn't rated before.

The next code is belong to development of UBCF using Matix Factorization:

```
def recommend_movies(prediction_df, userID, movies_df, original_ratings_df, num_recommendations = 5):
    user_row_number = userID - 1
    sorted_user_predictions = predicted_rating_df.iloc[user_row_number].sort_values(ascending = False)
    user_data = original_ratings_df[original_ratings_df.userId == (userID)]
    user_full = (user_data.merge(movies_df, how = 'left', left_on = 'movieId', right_on = 'movieId')
        .sort_values(['rating'], ascending = False))
    print('user {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print('Recommending highest {0} predicted ratings movies not already rated.'.format(num_recommendations))
    recommendations = (movies_df[~movies_df['movieId'].isin(user_full['movieId'])].
                                merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
                                      left_on = 'movieId', right_on = 'movieId').
                                      rename(columns = {user_row_number:'Predictions'}).
                                      sort_values('Predictions', ascending = False).
                                      iloc[:num_recommendations, :-1])
    return user_full, recommendations

already_rated, predictions = recommend_movies(predicted_rating_df, 2, movies_df, ratings_df, 10)
predictions
```

And finally, last method is K-Nearest Neighbors for UBCF which we have used in our webapp:

```
model_knn = NearestNeighbors(algorithm = 'brute', metric = 'cosine')
model_knn.fit(movieRatings.values)

user = 2

distances, indices = model_knn.kneighbors(movieRatings.iloc[user-1, :].values.reshape(1, -1), n_neighbors = 6)
best = []
movieRatings = movieRatings.T
for i in indices.flatten():
    if(i != user-1):
        max_score = movieRatings.loc[:, i + 1].max()
        best.append(movieRatings[movieRatings.loc[:, i + 1] == max_score].index.tolist())

user_seen_movies = movieRatings[movieRatings.loc[:, user] > 0].index.tolist()
for i in range(len(best)):
    for j in best[i]:
        if(j in user_seen_movies):
            best[i].remove(j)

most_common = {}
for i in range(len(best)):
    for j in best[i]:
        if j in most_common:
            most_common[j] += 1
        else:
            most_common[j] = 1

sorted_list = sorted(most_common.items(), key = operator.itemgetter(1), reverse = True)
sorted_list[:5]
```

Firstly, we initialize our model from “sklearn” library via importing NearestNeighbor algorithm. We fit our data to model and run the main function. Function takes userId as a parameter. From users high rated movies, it construct classes and finds nearest neighbors (most closest movies).

n_neighbors = 6 means most closest 6 neighbors. If we excluded user itself, system will recommend best 5 movies using euclidean distance. Accuracy score is 77%.

Confusion Matrix	Actual Positive	Actual Negative
Predicted Positive	9023	3829
Predicted Negative	1867	10424

Conclusion:

When working with data, it is important to select most appropriate algorithm/method according to your data. Our dataset lacks from movies features except genres. To build CBF, 1 feature is not sufficient as we can't recommend movies only bearing on genres flawlessly. Between working with IBCF and UBCF, we decided to work with UBCF as we have lots of users and ratings in our hand. After selecting the system, we need to decide on classifier a.k.a. method. We have used Cosine Similarity, Matrix Factorization, and K-Nearest Neighbors. We can say that all 3 algorithms in UBCF worked better than IBCF and CBF. Their accuracy scores are almost same. It can differ from project to project and data to data. As a team we decided to build UBCF with KNN algorithm.

What to do now:

Our output is simple Movie Recommendation system. In today's industry, it doesn't meet requirements of any Movie platform. Data is everything. If you have data with enough number of features and examples, other systems can be implemented such as Memory-based Collaborative Filtering and improved techniques, for instance, Root Mean Squared Error.

References

Django: https://www.djangoproject.com

How to Build a Simple Recommender System in Python: https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d

Machine Learning Projects Recommendation System Website: https://www.udemy.com/course/machine-learning-projects-recommendation-system-website

Movie Recommendation System Web Application: https://github.com/abd1007/Movie-Recommendation-System-Web-Application

Scikit-learn: https://scikit-learn.org