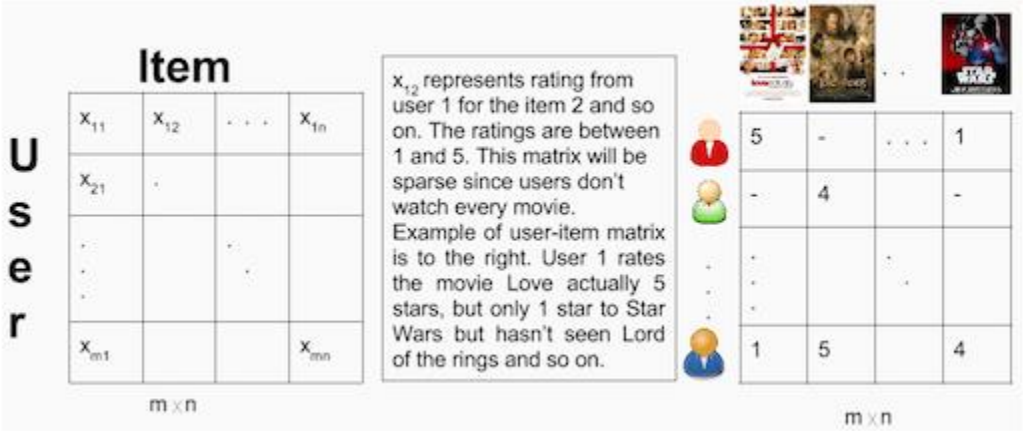
Recommender System based on K-Nearest Neighbors

This implementation is based on codes and ideas from attached references [2, 3].

This algorithm is built on MovieLens dataset on ratings.csv. The method will not estimate the ratings although it will determine the k closest users and their ratings. Then it averages the k users' interaction rows. First the data will be set in a matrix form as in the picture.



The algorithm is implemented in Python 3.6 using numpy, pandas, and sklearn libraries [2]

First, we loaded the data and then split the data into training set and test set to use it in an offline evaluation later. Then we created The previously mentioned user-item matrix.

```
# Create User-Item Matrix

header = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('ratings.dat', sep='::', names=header,engine='python')
#print (df.item_id)
n_users = df.user_id.unique().shape[0]
#n_items = df.item_id.unique().shape[0]
n_items = max(df.item_id)

print ('Number of users = ' + str(n_users) + ' | Number of movies = ' + str(n_items))
train_data, test_data = cv.train_test_split(df, test_size=0.1)

#Create two user-item matrices, one for training and another for testing
train_data_matrix = np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]
#print (train_data.itertuples())
test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]
```

We compared two methods of measuring similarity for specifying k-nearest neighbors, namely Euclidean distance and Cosine distance

$$sim(u, v)^{COS} = \frac{\vec{r}_u \cdot \vec{r}_v}{\|\vec{r}_u\| \cdot \|\vec{r}_v\|}$$

We implemented the k-NN in two ways: one with the ratings(0-5) and the other with a binary measure (1 if the user watched the movie and 0 otherwise). [3]

By adding the following before KNN iterations

```
W = train_data_matrix>0.5
W[W == True] = 1
W[W == False] = 0
# To be consistent with our Q matrix
W = W.astype(np.float64, copy=False)

Y =test_data_matrix>0.5
Y[Y == True] = 1
Y[Y == False] = 0
# To be consistent with our Q matrix
Y = Y.astype(np.float64, copy=False)
```

Then prepare KNN function

```
#
metric ='cosine'
ks=list(range(5,200,5))
training=W
testing=Y
print ("Training and scoring")
scores = []
knn = NearestNeighbors(metric=metric, algorithm="brute")
knn.fit(training)
aucmat=[]
for k in ks:
    print ("Evaluating for", k, "neighbors")
    neighbor_indices = knn.kneighbors(testing,
                                      n_neighbors=k,
                                      return_distance=False)

```

For each query user, we randomly chose an equal number of movies they had and had not interacted with. Thus, each positive training or test example had a corresponding negative example (no class imbalance). For each movie, we used the interaction vectors' entry as the true label. We then computed the area under the curve over all users, using the interaction vectors' entries as the true labels and the rankings for the chosen movies based on the predicted interaction scores. [3]

```
71
72      all_predicted_scores = []
73      all_labels = []
74      for user_id in range(testing.shape[0]):
75          user_row = testing[user_id, :]
76          nonZ=user_row.nonzero()
77
78          interaction_indices= list(nonZ[0])
79          interacted = set(interaction_indices)
80          non_interacted = set(range(testing.shape[1])) - interacted
81
82          n_samples = min(len(non_interacted), len(interacted))
83          sampled_interacted = random.sample(interacted, n_samples)
84          sampled_non_interacted = random.sample(non_interacted, n_samples)
85
86          indices = list(sampled_interacted)
87          indices.extend(sampled_non_interacted)
88          labels = [1] * n_samples
89          labels.extend([0] * n_samples)
90
91          neighbors = training[neighbor_indices[user_id, :], :]
92          predicted_scores = neighbors.mean(axis=0)
```

We then compute The Area under the curve for each k iterated.

Area under the curve is computed from the true positive-false positive curve computed from sampled vectors (SKlearn). Then we will plot AUC against k values

```
93          for idx in indices:
94              all_predicted_scores.append(predicted_scores[idx])
95          all_labels.extend(labels)
96
97      print (len(all_labels), len(all_predicted_scores))
98
99      auc = roc_auc_score(all_labels, all_predicted_scores)
100
101     print ("k", k, "AUC", auc)
102     aucmat=np.append(aucmat,auc)
```

After implementation Euclidean distance for KNN for the same data, we plot the AUC graph for each one of them. As it, appears the cosine similarity is better.

```
152 plt.plot(ks,aucmat)
153 plt.plot(ks, aucmat2)
154 plt.legend(['cosine', 'euclidean'])
155 plt.xlabel('K')
156 plt.ylabel('Area Under Curve')
```

There are 2 implementation one for 0-5 ratings and the other for binary ratings.

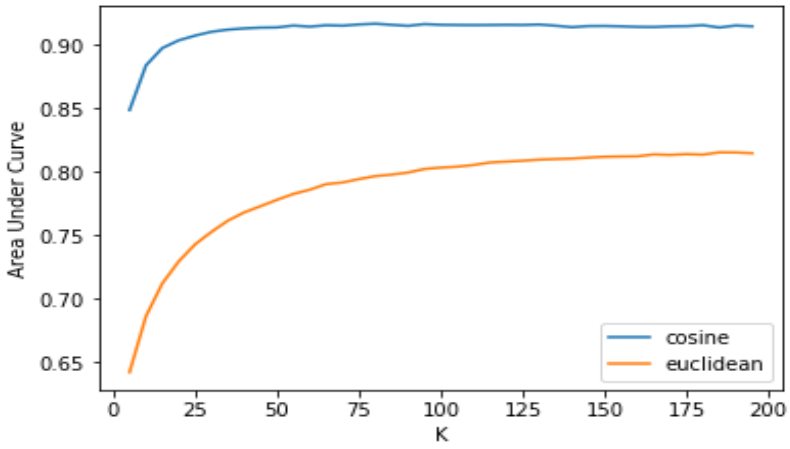


Figure 2 KNN for binary ratings

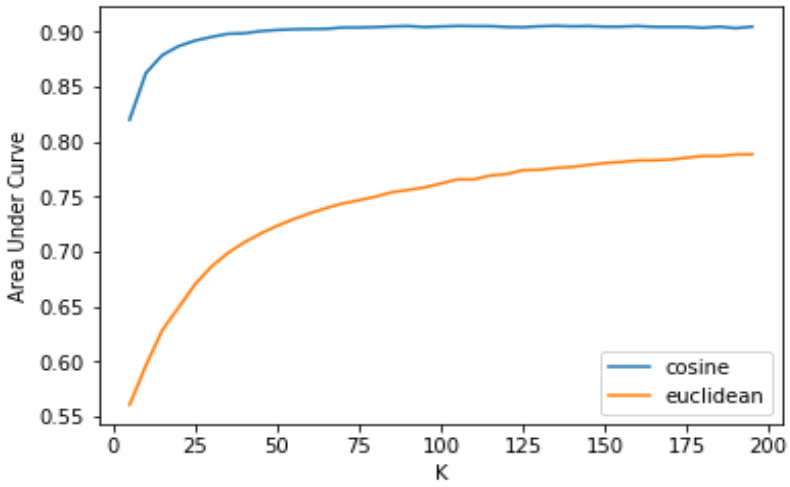


Figure 3 KNN for (0 to 5) ratings

Conclusion:

This implementation is not a good recommender but it gives a good comparison for the similarity methods here.

The maximum area under the curve is still not good due to the high sparsity level in the data.

References

[1] A new user similarity model to improve the accuracy of collaborative filtering. (2014). In ,. Z. Haifeng Liu, *Knowledge-Based Systems* (Vol. 56, pp. 156-166).

[2] Johannsdottir, A. (n.d.). *tutorials*. Retrieved from cambridgespark.com: https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-

systems-in-Python/index.html

[3] Nowling, R. (2016, 10 29). *rnowling*. (R. Nowling, Editor) Retrieved from github: http://rnowling.github.io/data/science/2016/10/29/knn-recsys.html