Question 1

a) MAE for user based collaborative filtering: 1.705

b) MAE for item based collaborative filtering: 2.105

Question 2

a) For User Based Precision: .95 Recall: .57 MAP: 0.5635416666666666 nDCG: 0.9918882873518092

b) For Item Based Precision: 0 Recall: 0 MAP: 0 nDCG: 0

My item based implementation did not yield any scores above 4.0. Therefore, the scores all compute 0.

notes: some code is based on the tutorial: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/ (https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/)

and ndcg_at_k implemented from: https://gist.github.com/bwhite/3726239 (https://gist.github.com/bwhite/3726239)

In [227]:
```
import os
import pandas as pd

workdir = os.getcwd()
def film_data():
    with open(os.path.join(workdir, 'selected_films.txt')) as file:
        lines = file.readlines()

    lines = map(str.strip, lines)
    lines = [[line[:4], line[5:]] for line in lines]
    lines = [[int(line[0])] + [line[1]] + [i] for i, line in enumerate(l
ines, 1)]
    lines = list(lines)
    df = pd.DataFrame(lines, columns=['Year', 'Title', 'Index'])
    return df

def user_data():
    with open(os.path.join(workdir, 'm_u_ratings.txt')) as file:
        lines = file.readlines()

    lines = map(str.strip, lines)
    lines = [map(int, line.split()) for line in lines]
    df = pd.DataFrame(lines, columns=['MovieID', 'UserID', 'Rating'])
    return df
```

In [228]:
```
ratings = user_data()
ratings.head()
```

Out[228]:

	MovieID	UserID	Rating
0	1	2897	3
1	1	6549	4
2	1	389	4
3	1	287	3
4	1	8867	3

In [3]:
```
films = film_data()
films.head()
```

Out[3]:

	Year	Title	Index
0	2000	Miss Congeniality	1
1	1996	Independence Day	2
2	2000	The Patriot	3
3	2004	The Day After Tomorrow	4
4	2003	Pirates of the Caribbean: The Curse of the Bla...	5

In [229]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(ratings, test_size=0.1)
```

In [5]:
```
test.head()
```

Out[5]:

	MovieID	UserID	Rating
2434915	300	3409	1
9750030	1949	3541	4
6906852	1090	3045	3
8937814	1647	845	3
5220134	739	9261	4

In [30]:
```
n_users = max(ratings.UserID.unique())
n_items = max(ratings.MovieID.unique())
```

In [31]:
```
import numpy as np

def calc_data_matrix(train):

    data_matrix = np.zeros((n_users, n_items))

    for line in train.itertuples():
        rating = line.Rating
        data_matrix[line.UserID-1, line.MovieID-1] = rating
    return data_matrix
```

In [32]:
```
from sklearn.metrics.pairwise import pairwise_distances

def get_similarities(data_matrix):
    user_similarity = pairwise_distances(data_matrix, metric='cosine')
    item_similarity = pairwise_distances(data_matrix.T, metric='cosine')
    return user_similarity, item_similarity
```

In [230]:
```
# This function based on tutorial https://www.analyticsvidhya.com/blog/2
018/06/comprehensive-guide-recommendation-engine-python/
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_
diff) / np.array([np.abs(similarity).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).su
m(axis=1)])
    return pred
```

In [10]:
```
from sklearn.metrics import mean_absolute_error as mse
def get_y_pred(prediction, test):

    y_pred = []
    for i, movieID, userID, Rating in test.itertuples():
        rating = prediction[userID-1][movieID-1]
        y_pred.append(rating)
    return y_pred

def predict_mse(prediction, test):
    y_pred = get_y_pred(prediction, test)
    y_true = test['Rating'].values
    return mse(y_true, y_pred, sample_weight=None, multioutput='uniform_
average')


def pred_mse(train, test):
    """ returns mse for user_prediction and item_prediction """
    data_matrix = calc_data_matrix(train)
    user_similarity, item_similarity = get_similarities(data_matrix)

    user_prediction = predict(data_matrix, user_similarity, type='user')
    item_prediction = predict(data_matrix, item_similarity, type='item')

    mse_user = predict_mse(user_prediction, test)
    mse_item = predict_mse(item_prediction, test)

    return mse_user, mse_item
```

In [34]:
```
from sklearn.metrics import mean_absolute_error as mse
from sklearn.model_selection import KFold

kf = KFold(n_splits=10)

mse_user_error = 0
mse_item_error = 0

i = 0
for train_index, test_index in kf.split(ratings):
    i+=1
    train, test = ratings.iloc[train_index], ratings.iloc[test_index]
    train, test = train_test_split(ratings, test_size=0.1)
    mse_user, mse_item = pred_mse(train, test)
    mse_user_error += mse_user
    mse_item_error += mse_item

mse_user_error /= i
mse_item_error /= i
```

In [35]:
```
mse_user_error, mse_item_error
```

Out[35]:
```
(1.7061068800781376, 2.1062108905575863)
```

In [51]:
```
import random
from functools import reduce

# Selected 100 users ratings,
users = [random.randint(1, n_users) for _ in range(100)]

# Select 10% of each user's ratings
test = [ratings[ratings.UserID==userid].sample(frac=.1) for userid in us
ers]
test = pd.concat(test)
```

In [52]:
```
# Drop the test from the training set
train = ratings.drop(test.index)
```

In [54]:
```
data_matrix = calc_data_matrix(train)
user_similarity, item_similarity = get_similarities(data_matrix)

user_prediction = predict(data_matrix, user_similarity, type='user')
item_prediction = predict(data_matrix, item_similarity, type='item')
```

In [55]:
```
def movie_ids_for_user(userID):
    return test[test.UserID == userID].MovieID.values

test.head()
```

Out[55]:

	MovieID	UserID	Rating
4916667	683	8550	3
7269502	1178	8550	4
1345843	160	8550	2
1310131	155	8550	5
5013318	700	8550	4

In [56]:
```
from collections import namedtuple


def datamatrix_to_df(data_matrix):
    return pd.DataFrame(data_matrix)


def user_movie_mask(userID, predictions):
    movie_ids = movie_ids_for_user(userID)
    mask = (predictions == predictions) & (predictions.columns.isin(movi
e_ids))
    mask = mask[mask.index==userID]
    return mask


def users_mask(predictions):
    masks = [user_movie_mask(userID, predictions) for userID in set(test
.UserID.values)]
    return masks


def get_predictions(prediction):
    df = datamatrix_to_df(prediction)
    df.index = np.arange(1, len(df)+1)
    df.columns = np.arange(1, len(df.columns.values) + 1)

    # Predcitions for the 100 users
    predictions = df[df.index.isin(test.UserID.values)]
    return predictions


def get_ratings_over(predictions, over=4):
    masks = users_mask(predictions)
    masked = pd.DataFrame()

    for mask in masks:
        index = mask.index.values[0]
        df = predictions[predictions.index == index].mask(~mask)
        masked = pd.concat([masked, df])

    return masked[masked > over]

test_ratings_users = get_predictions(user_prediction)
test_ratings_items = get_predictions(item_prediction)
```

In [143]:
```
pred_ratings_users = get_ratings_over(test_ratings_users, 4)
pred_ratings_items = get_ratings_over(test_ratings_items, 4)
```

In [144]:
```
def test_split(rating_df):
    y_pred = []
    y_true = []
    true_ratings = []
    for userID, movie_ratings in rating_df.iterrows():
        user_ratings_true = ratings[ratings.UserID == userID]
        pred = []
        true = []
        for movieID, rating in enumerate(movie_ratings, 1):
            if np.isnan(rating):
                continue
            true.append(user_ratings_true[user_ratings_true.MovieID == m
ovieID].Rating.values[0])
            pred.append(rating)

        y_true.append(true)
        y_pred.append(pred)
        true_ratings.append(true_ratings_above(userID))

    return y_true, y_pred, true_ratings

def true_ratings_above(userID, threshold=4):
    return sum(test[test.UserID == userID].Rating.values >= threshold)

y_true, y_pred, true_ratings = test_split(pred_ratings_users)
```

In [163]:
```
# Precision: recommended items / relevant recommended items
# Recall: recommened items / total # relevant items

def calc_precision(y_true, y_pred, total_true):
    precision = 0
    for true, pred, total in zip(y_true, y_pred, total_true):
        wrong = list(filter(lambda x: x < 4, true))

        if len(pred) == 0 and len(true) == 0:
            precision += 1
            continue
        try:
            precision += (len(pred) - len(wrong)) / len(pred)
        except Exception:
            precision += 0

    return precision/len(total_true)

def calc_recall(y_true, y_pred, total_true):
    total = 0
    for pred, total in zip(y_pred, total_true):
        total += len(pred) / total
    return total / len(total_true)

prec = calc_precision(y_true, y_pred, true_ratings)
recall = calc_recall(y_true, y_pred, true_ratings)
```

In [164]:
```
prec, recall
```

Out[164]:
```
(0.9549166666666667, 0.57)
```

In [202]:
```
from sklearn.metrics import average_precision_score

def compute_map_scores(y_true, y_pred):
    avg = 0
    i = 0
    for true, pred in zip(y_true, y_pred):
        if len(true) == 0 or len(pred)==0: continue
        true_wrong = [True if x < 4 else False for x in true]
        pred = [1 for _ in pred]
        score = average_precision_score(true_wrong, [1 for _ in pred])
        if np.isnan(score):
            continue
        avg+=score
        i+=1
    return avg / i


score = compute_map_scores(y_true, y_pred)
score
```

```
/Users/audretjm/anaconda3/lib/python3.6/site-packages/sklearn/metrics/r
anking.py:444: RuntimeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
```

Out[202]:
```
0.5635416666666666
```

In [226]:
```
def compute_ndcg_scores(y_true, y_pred):
    avg = 0
    i = 0
    for true, pred in zip(y_true, y_pred):
        relevance = [1 - abs(a-b)/5 for a,b in zip(true, pred)]
        if not relevance:
            continue
        i += 1
        score = ndcg_at_k(relevance, k=len(relevance))
        avg += score
    return avg / i

score = compute_ndcg_scores(y_true, y_pred)
score
```

Out[226]:
```
0.9918882873518092
```

In [215]:
```
# Code taken from: https://gist.github.com/bwhite/3726239 for ndcg@k rat
ings
import numpy as np


def dcg_at_k(r, k, method=0):

    r = np.asfarray(r)[:k]
    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1
)))
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    return 0.


def ndcg_at_k(r, k, method=0):

    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max
```