

# Final Milestone

December 13, 2016

```
In [6]: from collections import defaultdict
        from datetime import datetime
        from math import sqrt
```

```
import numpy as np
import pandas as pd
```

```
%matplotlib inline
```

## 1 Read data

```
In [15]: EUCLIDEAN = 'euclidean'
        MANHATTAN = 'manhattan'
        PEARSON = 'pearson'
```

```
def read_ratings_df():
    date_parser = lambda time_in_secs: datetime.utcfromtimestamp(float(time_in_secs))
    return pd.read_csv('ml-latest-small/ratings.csv', parse_dates=['timestamp'])
```

```
class MovieData(object):
    def __init__(self):
        self.ratings_df = read_ratings_df()
        self.ratings = defaultdict(dict)
        self.init_ratings()

    def init_ratings(self):
        for _, row in self.ratings_df.iterrows():
            self.ratings[row['userId']][row['movieId']] = row['rating']

    def get_movies(self, user_id):
        return set(self.ratings[user_id].keys())

    def get_unique_user_ids(self):
        return self.ratings_df['userId'].unique()
```

```

def get_shared_ratings(self, user1_id, user2_id):
    movies1 = self.get_movies(user1_id)
    movies2 = self.get_movies(user2_id)

    shared_movies = movies1 & movies2

    ratings = {}

    for movie_id in shared_movies:
        ratings[movie_id] = (
            self.ratings[user1_id][movie_id]['rating'],
            self.ratings[user2_id][movie_id]['rating'],
        )

    return ratings

@staticmethod
def shared_ratings_to_np_arrays(shared_ratings):
    return np.array(shared_ratings.values()).T

def get_euclidean_distance(self, user1_id, user2_id):
    shared_ratings = self.get_shared_ratings(user1_id, user2_id)

    if len(shared_ratings) == 0:
        return 0

    ratings = self.shared_ratings_to_np_arrays(shared_ratings)

    ratings1 = ratings[0]
    ratings2 = ratings[1]

    sum_of_squares = np.power(ratings1 - ratings2, 2).sum()

    return 1 / (1 + sqrt(sum_of_squares))

def get_manhattan_distance(self, user1_id, user2_id):
    shared_ratings = self.get_shared_ratings(user1_id, user2_id)

    if len(shared_ratings) == 0:
        return 0

    ratings = self.shared_ratings_to_np_arrays(shared_ratings)

    ratings1 = ratings[0]
    ratings2 = ratings[1]

    manhattan_sum = np.abs(ratings1 - ratings2).sum()

```

```

        return 1 / (1 + manhattan_sum)

def get_pearson_correlation(self, user1_id, user2_id):
    shared_ratings = self.get_shared_ratings(user1_id, user2_id)

    num_ratings = len(shared_ratings)

    if num_ratings == 0:
        return 0

    ratings = self.shared_ratings_to_np_arrays(shared_ratings)

    ratings1 = ratings[0]
    ratings2 = ratings[1]

    mean1 = ratings1.mean()
    mean2 = ratings2.mean()

    std1 = ratings1.std()
    std2 = ratings2.std()

    if std1 == 0 or std2 == 0:
        return 0

    std_scores_1 = (ratings1 - mean1) / std1
    std_scores_2 = (ratings2 - mean2) / std2

    # numerically stable calculation of the Pearson correlation coefficient
    return abs((std_scores_1 * std_scores_2).sum() / (num_ratings - 1))

def get_similar_users(self, user_id, metric=EUCLIDEAN):
    metrics = {
        EUCLIDEAN: self.get_euclidean_distance,
        MANHATTAN: self.get_manhattan_distance,
        PEARSON: self.get_pearson_correlation,
    }

    distance_f = metrics[metric]

    similar_users = {}

    for similar_user_id in self.ratings:
        if similar_user_id == user_id:
            continue
        distance = distance_f(user_id, similar_user_id)
        if distance > 0:
            similar_users[similar_user_id] = distance

```

```

        return similar_users

    def predict_score(self, user_id, movie_id):
        similar_users = self.get_similar_users(user_id)

        total_rating_sum = 0
        similarity_sum = 0

        for similar_user_id, similarity in similar_users.items():
            user_ratings = self.ratings[similar_user_id]
            if movie_id in user_ratings:
                total_rating_sum += similarity * user_ratings[movie_id]
                similarity_sum += similarity

        if similarity_sum == 0:
            return 0

        return total_rating_sum / similarity_sum

```

```
movie_data = MovieData()
```

## 2 Explore shared ratings

```

In [11]: def explore_shared_ratings(movie_data):
    unique_user_ids = movie_data.get_unique_user_ids()

    n_pairs = 30
    samples = np.random.choice(unique_user_ids, size=(n_pairs, 2))

    for index, sample in enumerate(samples):
        user1_id = sample[0]
        user2_id = sample[1]

        num_movies_1 = len(movie_data.get_movies(user1_id))
        num_movies_2 = len(movie_data.get_movies(user2_id))

        num_shared_ratings = len(movie_data.get_shared_ratings(user1_id, user2_id))

        print 'pair %2d, user1 movies: %4d, user2 movies: %4d, shared movies: %4d' % (
            index + 1, num_movies_1, num_movies_2, num_shared_ratings)

    explore_shared_ratings(movie_data)

```

```

pair 1, user1 movies: 111, user2 movies: 485, shared movies: 28
pair 2, user1 movies: 21, user2 movies: 20, shared movies: 0

```

```

pair 3, user1 movies: 63, user2 movies: 23, shared movies: 5
pair 4, user1 movies: 483, user2 movies: 159, shared movies: 87
pair 5, user1 movies: 22, user2 movies: 72, shared movies: 3
pair 6, user1 movies: 50, user2 movies: 20, shared movies: 0
pair 7, user1 movies: 22, user2 movies: 385, shared movies: 7
pair 8, user1 movies: 263, user2 movies: 129, shared movies: 26
pair 9, user1 movies: 300, user2 movies: 22, shared movies: 7
pair 10, user1 movies: 38, user2 movies: 61, shared movies: 0
pair 11, user1 movies: 87, user2 movies: 36, shared movies: 2
pair 12, user1 movies: 427, user2 movies: 79, shared movies: 31
pair 13, user1 movies: 20, user2 movies: 522, shared movies: 6
pair 14, user1 movies: 114, user2 movies: 87, shared movies: 14
pair 15, user1 movies: 28, user2 movies: 51, shared movies: 1
pair 16, user1 movies: 215, user2 movies: 223, shared movies: 26
pair 17, user1 movies: 713, user2 movies: 44, shared movies: 19
pair 18, user1 movies: 82, user2 movies: 21, shared movies: 2
pair 19, user1 movies: 73, user2 movies: 59, shared movies: 7
pair 20, user1 movies: 617, user2 movies: 255, shared movies: 98
pair 21, user1 movies: 138, user2 movies: 99, shared movies: 2
pair 22, user1 movies: 99, user2 movies: 28, shared movies: 3
pair 23, user1 movies: 24, user2 movies: 155, shared movies: 2
pair 24, user1 movies: 291, user2 movies: 25, shared movies: 0
pair 25, user1 movies: 617, user2 movies: 205, shared movies: 92
pair 26, user1 movies: 100, user2 movies: 31, shared movies: 8
pair 27, user1 movies: 91, user2 movies: 26, shared movies: 2
pair 28, user1 movies: 487, user2 movies: 51, shared movies: 20
pair 29, user1 movies: 26, user2 movies: 133, shared movies: 1
pair 30, user1 movies: 1291, user2 movies: 194, shared movies: 126

```

We are looking at 30 random user pairs. We can notice how small on average is the intersection of the movies they rated (compared to their total number of ratings). It's not unusual to see zero intersection or just a couple of movies.

We could build a histogram of the distribution of number of shared movies if we generate a lot of random pairs.

### 3 Explore distances

```

In [12]: def explore_distances(movie_data):
           unique_user_ids = movie_data.get_unique_user_ids()

           n_pairs = 30
           samples = np.random.choice(unique_user_ids, size=(n_pairs, 2))

           for index, sample in enumerate(samples):
               user1_id = sample[0]
               user2_id = sample[1]

```

```

num_shared_ratings = len(movie_data.get_shared_ratings(user1_id, u

euclidean_distance = movie_data.get_euclidean_distance(user1_id, u
manhattan_distance = movie_data.get_manhattan_distance(user1_id, u
pearson_correlation = movie_data.get_pearson_correlation(user1_id, u

print 'pair %2d, shared movies: %3d, euclidean: %.3f, manhattan: %
      index + 1, num_shared_ratings, euclidean_distance, manhattan_c

```

```

explore_distances(movie_data)

```

```

pair 1, shared movies: 65, euclidean: 0.098, manhattan: 0.018, pearson: 0.210
pair 2, shared movies: 0, euclidean: 0.000, manhattan: 0.000, pearson: 0.000
pair 3, shared movies: 49, euclidean: 0.112, manhattan: 0.024, pearson: 0.201
pair 4, shared movies: 8, euclidean: 0.152, manhattan: 0.083, pearson: 0.123
pair 5, shared movies: 1, euclidean: 0.400, manhattan: 0.400, pearson: 0.000
pair 6, shared movies: 0, euclidean: 0.000, manhattan: 0.000, pearson: 0.000
pair 7, shared movies: 0, euclidean: 0.000, manhattan: 0.000, pearson: 0.000
pair 8, shared movies: 6, euclidean: 0.200, manhattan: 0.111, pearson: 0.100
pair 9, shared movies: 108, euclidean: 0.053, manhattan: 0.006, pearson: 0.088
pair 10, shared movies: 27, euclidean: 0.121, manhattan: 0.031, pearson: 0.023
pair 11, shared movies: 29, euclidean: 0.131, manhattan: 0.035, pearson: 0.257
pair 12, shared movies: 7, euclidean: 0.232, manhattan: 0.125, pearson: 0.490
pair 13, shared movies: 1, euclidean: 0.333, manhattan: 0.333, pearson: 0.000
pair 14, shared movies: 12, euclidean: 0.081, manhattan: 0.026, pearson: 0.250
pair 15, shared movies: 11, euclidean: 0.240, manhattan: 0.111, pearson: 0.289
pair 16, shared movies: 0, euclidean: 0.000, manhattan: 0.000, pearson: 0.000
pair 17, shared movies: 7, euclidean: 0.240, manhattan: 0.200, pearson: 0.269
pair 18, shared movies: 1, euclidean: 0.500, manhattan: 0.500, pearson: 0.000
pair 19, shared movies: 2, euclidean: 0.271, manhattan: 0.222, pearson: 2.000
pair 20, shared movies: 2, euclidean: 0.667, manhattan: 0.667, pearson: 2.000
pair 21, shared movies: 5, euclidean: 0.327, manhattan: 0.222, pearson: 0.859
pair 22, shared movies: 7, euclidean: 0.194, manhattan: 0.095, pearson: 0.776
pair 23, shared movies: 0, euclidean: 0.000, manhattan: 0.000, pearson: 0.000
pair 24, shared movies: 24, euclidean: 0.131, manhattan: 0.037, pearson: 0.324
pair 25, shared movies: 67, euclidean: 0.099, manhattan: 0.018, pearson: 0.304
pair 26, shared movies: 2, euclidean: 0.387, manhattan: 0.333, pearson: 0.000
pair 27, shared movies: 38, euclidean: 0.203, manhattan: 0.050, pearson: 0.565
pair 28, shared movies: 3, euclidean: 0.250, manhattan: 0.167, pearson: 0.750
pair 29, shared movies: 7, euclidean: 0.286, manhattan: 0.154, pearson: 0.297
pair 30, shared movies: 9, euclidean: 0.152, manhattan: 0.077, pearson: 0.025

```

Various distances (euclidean, manhattan, pearson correlation).

Other possible distances: Tanimoto, cosine.

Jaccard distance is not really applicable in this case since we have a range of ratings.

## 4 Explore similar users

```
In [14]: def explore_similar_users(movie_data):
        unique_user_ids = movie_data.get_unique_user_ids()

        n_users = 30
        user_ids = np.random.choice(unique_user_ids, size=n_users, replace=False)

        for index, user_id in enumerate(user_ids):
            similar_users = movie_data.get_similar_users(user_id)

            distances = similar_users.values()

            print 'user %3d, similar users: %d, max similarity: %.3f, mean: %.3f' % (
                index + 1, len(similar_users), np.max(distances), np.mean(distances))

        explore_similar_users(movie_data)
```

user	1,	similar users: 507,	max similarity: 1.000,	mean: 0.401,	std: 0.219
user	2,	similar users: 664,	max similarity: 1.000,	mean: 0.191,	std: 0.114
user	3,	similar users: 668,	max similarity: 1.000,	mean: 0.202,	std: 0.126
user	4,	similar users: 665,	max similarity: 1.000,	mean: 0.145,	std: 0.108
user	5,	similar users: 670,	max similarity: 1.000,	mean: 0.156,	std: 0.090
user	6,	similar users: 600,	max similarity: 1.000,	mean: 0.280,	std: 0.197
user	7,	similar users: 406,	max similarity: 1.000,	mean: 0.378,	std: 0.221
user	8,	similar users: 653,	max similarity: 1.000,	mean: 0.262,	std: 0.165
user	9,	similar users: 647,	max similarity: 1.000,	mean: 0.245,	std: 0.139
user	10,	similar users: 658,	max similarity: 1.000,	mean: 0.239,	std: 0.145
user	11,	similar users: 670,	max similarity: 0.348,	mean: 0.115,	std: 0.055
user	12,	similar users: 530,	max similarity: 1.000,	mean: 0.354,	std: 0.187
user	13,	similar users: 593,	max similarity: 1.000,	mean: 0.346,	std: 0.224
user	14,	similar users: 342,	max similarity: 1.000,	mean: 0.412,	std: 0.197
user	15,	similar users: 493,	max similarity: 1.000,	mean: 0.448,	std: 0.263
user	16,	similar users: 654,	max similarity: 1.000,	mean: 0.237,	std: 0.140
user	17,	similar users: 652,	max similarity: 1.000,	mean: 0.285,	std: 0.166
user	18,	similar users: 601,	max similarity: 1.000,	mean: 0.263,	std: 0.194
user	19,	similar users: 638,	max similarity: 1.000,	mean: 0.267,	std: 0.163
user	20,	similar users: 467,	max similarity: 1.000,	mean: 0.305,	std: 0.194
user	21,	similar users: 311,	max similarity: 1.000,	mean: 0.475,	std: 0.248
user	22,	similar users: 644,	max similarity: 1.000,	mean: 0.278,	std: 0.150
user	23,	similar users: 659,	max similarity: 1.000,	mean: 0.208,	std: 0.141
user	24,	similar users: 667,	max similarity: 1.000,	mean: 0.137,	std: 0.102
user	25,	similar users: 621,	max similarity: 1.000,	mean: 0.325,	std: 0.159
user	26,	similar users: 652,	max similarity: 1.000,	mean: 0.261,	std: 0.153
user	27,	similar users: 634,	max similarity: 1.000,	mean: 0.306,	std: 0.150
user	28,	similar users: 556,	max similarity: 1.000,	mean: 0.425,	std: 0.206
user	29,	similar users: 634,	max similarity: 1.000,	mean: 0.279,	std: 0.166

```
user 30, similar users: 628, max similarity: 1.000, mean: 0.264, std: 0.164
```

Max similarity of 1.0 in most cases is probably an intersection of one movie.

## 5 Explore predict score (user similarity model)

```
In [16]: def explore_predict_score(movie_data):
          ratings_df = movie_data.ratings_df
          rating_indices = ratings_df.index

          n_ratings = 30
          sample = np.random.choice(rating_indices, size=n_ratings, replace=False)

          for index, rating_index in enumerate(sample):
              row = ratings_df.ix[rating_index]

              user_id = row['userId']
              movie_id = row['movieId']
              rating = row['rating']

              score = movie_data.predict_score(user_id, movie_id)

              print 'rating %2d, rating: %.1f, predicted: %.3f' % (index + 1, rating, score)

          explore_predict_score(movie_data)

rating 1, rating: 3.5, predicted: 3.908
rating 2, rating: 5.0, predicted: 4.532
rating 3, rating: 2.0, predicted: 3.670
rating 4, rating: 3.0, predicted: 3.389
rating 5, rating: 3.5, predicted: 3.541
rating 6, rating: 4.0, predicted: 3.177
rating 7, rating: 1.5, predicted: 3.747
rating 8, rating: 5.0, predicted: 3.993
rating 9, rating: 3.0, predicted: 3.329
rating 10, rating: 2.0, predicted: 3.590
rating 11, rating: 1.5, predicted: 3.154
rating 12, rating: 4.0, predicted: 3.322
rating 13, rating: 2.5, predicted: 2.950
rating 14, rating: 2.5, predicted: 2.934
rating 15, rating: 3.5, predicted: 0.000
rating 16, rating: 4.5, predicted: 3.824
rating 17, rating: 3.0, predicted: 3.453
rating 18, rating: 4.0, predicted: 4.117
rating 19, rating: 5.0, predicted: 3.971
rating 20, rating: 1.0, predicted: 3.307
```



```
rating 21, rating: 4.0, predicted: 3.485
rating 22, rating: 4.0, predicted: 4.173
rating 23, rating: 3.5, predicted: 3.815
rating 24, rating: 4.0, predicted: 3.399
rating 25, rating: 5.0, predicted: 4.134
rating 26, rating: 2.0, predicted: 3.791
rating 27, rating: 5.0, predicted: 3.858
rating 28, rating: 4.5, predicted: 4.108
rating 29, rating: 4.0, predicted: 3.758
rating 30, rating: 5.0, predicted: 4.526
```

```
In [ ]:
```