НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:

«Лабораторная работа №17»

Выполнил: Осипов Лев,

студент группы БПИ121 (1).

Москва, 2015 г.

**СОДЕРЖАНИЕ**

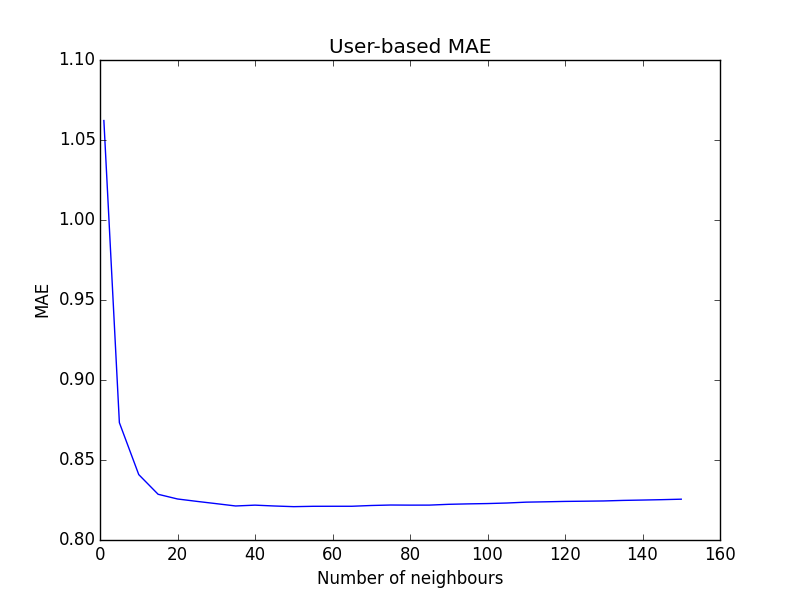
**Практическая часть3**

**Список литературы8**

**Текст программы9**

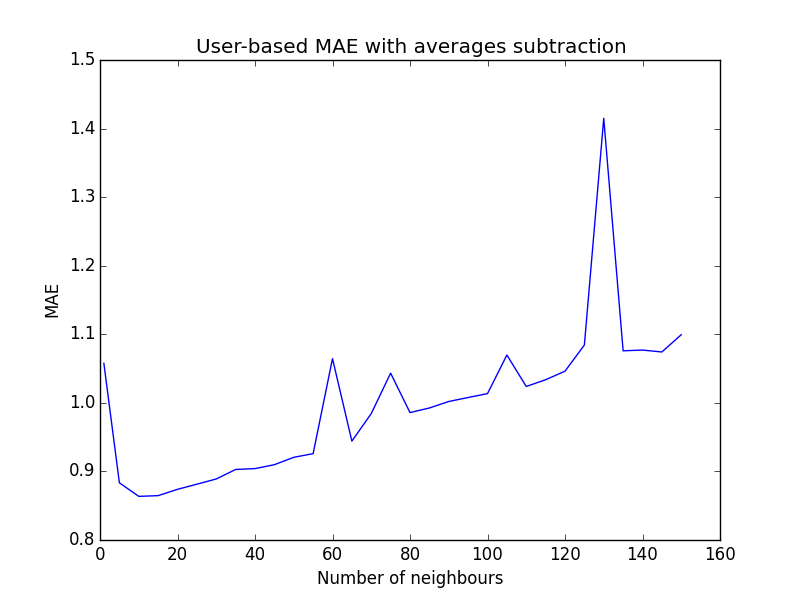
**ПРАКТИЧЕСКАЯ ЧАСТЬ**

Вначале данные были считаны из файла. Затем была составлена таблица похожести пользователей. Далее по таблице для тестовых данных были посчитаны значения ошибки при учете различного количества «ближайших пользователей»:



*Зависимость ошибки от количества ближайших соседей, user-based подход*

После этого обучающие данные были изменены – из рейтингов каждого пользователя был вычтен средний рейтинг этого пользователя. После этого была вновь выполнена операция составления таблицы похожести и анализ ошибки в зависимости от разного числа соседей:

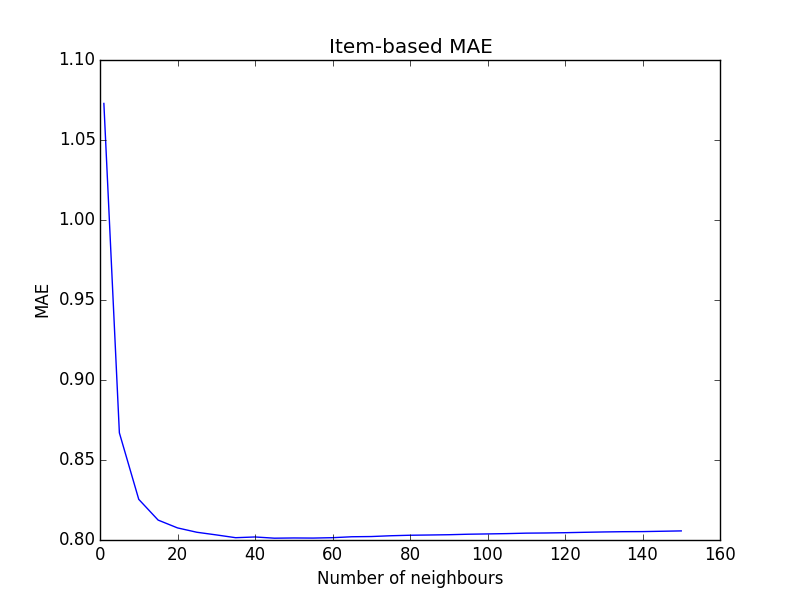


*Зависимость ошибки от количества ближайших соседей (с вычитанием среднего), user-based подход*

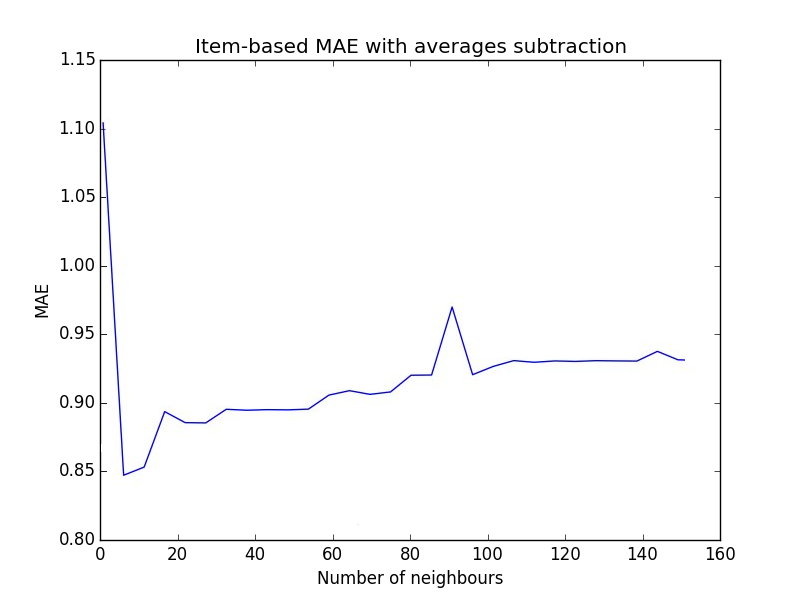
В целом следует выделить тенденцию резкого снижения количества ошибок до момента, когда число соседей становится примерно 15 (в общем случае от 10 до 20). После этого оно начинает увеличиваться.

Особенно стоит выделить нестабильность графика с экспериментом с вычитанием среднего. Это связано с тем, что относительная оценка может получаться одинаковой даже при разнице в ее абсолютных значениях. Особенно это становится заметно при большом числе пользователей. Это приводит к переобучению.

Далее, эти 2 исследования повторились, но уже с item-based подходом:



*Зависимость ошибки от количества ближайших соседей, item-based подход*



*Зависимость ошибки от количества ближайших соседей (с вычитанием среднего), item-based подход*

Здесь можно вновь повторить, что в интервале приблизительно от 10 до 20 соседей наблюдается минимум ошибки при разных видах экпериментов.

Видно, что по сравнению с экспериментом с вычитанием среднего в user-based подходе, здесь нет таких резких скачков. Это, скорее всего, происходит из-за того, что неоднозначность вследствие относительности оценок пользователей гораздо сильнее влияет на принцип user-based подхода (когда идет похожесть непосредственно пользователей).

**СПИСОК ЛИТЕРАТУРЫ**

1. **Анализ данных (Программная инженерия)** –   
   http://wiki.cs.hse.ru/Анализ\_данных\_(Программная\_инженерия)

**ТЕКСТ ПРОГРАММЫ**

\_\_author\_\_ = 'Lev Osipov'  
  
  
**import** pandas **as** pd  
**import** numpy **as** np  
**import** math  
**import** matplotlib.pyplot **as** plt  
  
  
**def count\_mae\_ub**(train\_user\_item\_matrix, matrix, sim\_sorted, user\_numb, item\_numb, neigh\_numb):  
 mae = 0  
 count = 0  
 **for** i **in** xrange(user\_numb):  
 **for** j **in** xrange(item\_numb):  
 **if** matrix[i, j] != 0:  
 mult\_sum = 0  
 sim\_sum = 0  
 numb\_found = 0  
 numb\_seen = 0  
 **while** numb\_found < neigh\_numb **and** numb\_seen < user\_numb:  
 sim = sim\_sorted[i, numb\_seen, 0]  
 user\_ind = sim\_sorted[i, numb\_seen, 1]  
 **if** train\_user\_item\_matrix[user\_ind, j] != 0:  
 mult\_sum += sim \* train\_user\_item\_matrix[user\_ind, j]  
 sim\_sum += sim  
 numb\_found += 1  
 numb\_seen += 1  
 **if** sim\_sum != 0 **and** numb\_found != 0:  
 mae += abs((mult\_sum / sim\_sum) - matrix[i, j])  
 count += 1  
 **if** count == 0:  
 **return** 0  
 **return** mae / count  
  
  
**def count\_averages\_users**(matrix, user\_numb, item\_numb):  
 averages = np.zeros(user\_numb)  
 **for** i **in** xrange(user\_numb):  
 count = 0  
 sum = 0  
 **for** j **in** xrange(item\_numb):  
 **if** matrix[i, j] != 0:  
 sum += matrix[i, j]  
 count += 1  
 **if** count == 0:  
 averages[i] = 0  
 **else**:  
 averages[i] = sum / float(count)  
 **return** averages  
  
  
**def count\_averages\_items**(matrix, user\_numb, item\_numb):  
 averages = np.zeros(item\_numb)  
 **for** i **in** xrange(item\_numb):  
 count = 0  
 sum = 0  
 **for** j **in** xrange(user\_numb):  
 **if** matrix[j, i] != 0:  
 sum += matrix[j, i]  
 count += 1  
 **if** count == 0:  
 averages[i] = 0  
 **else**:  
 averages[i] = sum / float(count)  
 **return** averages  
  
  
**def user\_based\_test**(name, train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb):  
 maes = np.zeros(31)  
 ks = np.zeros(31)  
  
 maes[0] = count\_mae\_ub(train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb, 1)  
 ks[0] = 1  
  
 **for** i **in** xrange(1, 31):  
 **print** i \* 5  
 maes[i] = count\_mae\_ub(train\_user\_item\_matrix,  
 test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb, i \* 5)  
 ks[i] = i \* 5  
  
 plt.title(name)  
 plt.xlabel('Number of neighbours')  
 plt.ylabel('MAE')  
 plt.plot(ks, maes)  
 plt.savefig(name + '.png')  
 plt.clf()  
  
  
**def user\_based**(name, train\_user\_item\_matrix\_a, train\_user\_item\_matrix, user\_numb, train\_item\_numb):  
 sim\_matrix = np.zeros(shape=(user\_numb, user\_numb))  
  
 **for** i **in** xrange(user\_numb):  
 **for** j **in** xrange(i + 1, user\_numb):  
 mult\_sum = 0  
 r1\_sum = 0  
 r2\_sum = 0  
 **for** k **in** xrange(train\_item\_numb):  
 r1 = train\_user\_item\_matrix\_a[i, k]  
 r2 = train\_user\_item\_matrix\_a[j, k]  
 **if** r1 != 0 **and** r2 != 0:  
 mult\_sum += r1 \* r2  
 r1\_sum += r1 \*\* 2  
 r2\_sum += r2 \*\* 2  
 **if** r1\_sum != 0 **and** r2\_sum != 0:  
 sim = mult\_sum / (math.sqrt(r1\_sum) \* math.sqrt(r2\_sum))  
 sim\_matrix[i, j] = sim  
 sim\_matrix[j, i] = sim  
  
 sim\_sorted = np.zeros(shape=(user\_numb, user\_numb, 2))  
  
 **for** i **in** xrange(user\_numb):  
 temp = np.zeros(shape=(user\_numb, 2))  
 **for** j **in** xrange(user\_numb):  
 temp[j, 0] = sim\_matrix[i, j]  
 temp[j, 1] = j  
 sim\_sorted[i] = np.array(sorted(temp, key=**lambda** x: x[0], reverse=True))  
  
 test\_data = pd.read\_csv('ua.test', sep='\t')  
 test\_data = np.array(test\_data)  
  
 test\_item\_numb = 0  
 **for** i **in** xrange(len(test\_data)):  
 **if** test\_data[i][1] > test\_item\_numb:  
 test\_item\_numb = test\_data[i][1]  
  
 test\_user\_item\_matrix = np.zeros(shape=(user\_numb, test\_item\_numb))  
  
 **for** i **in** xrange(len(test\_data)):  
 test\_user\_item\_matrix[test\_data[i][0] - 1, test\_data[i][1] - 1] = test\_data[i][2]  
  
 user\_based\_test(name, train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb)  
  
  
**def count\_mae\_ib**(train\_user\_item\_matrix, matrix, sim\_sorted, user\_numb, item\_numb, neigh\_numb):  
 mae = 0  
 count = 0  
 **for** i **in** xrange(item\_numb):  
 **for** j **in** xrange(user\_numb):  
 **if** matrix[j, i] != 0:  
 mult\_sum = 0  
 sim\_sum = 0  
 numb\_found = 0  
 numb\_seen = 0  
 **while** numb\_found < neigh\_numb **and** numb\_seen < user\_numb:  
 sim = sim\_sorted[i, numb\_seen, 0]  
 item\_ind = sim\_sorted[i, numb\_seen, 1]  
 **if** train\_user\_item\_matrix[j, item\_ind] != 0:  
 mult\_sum += sim \* train\_user\_item\_matrix[j, item\_ind]  
 sim\_sum += sim  
 numb\_found += 1  
 numb\_seen += 1  
 **if** sim\_sum != 0 **and** numb\_found != 0:  
 mae += abs((mult\_sum / sim\_sum) - matrix[j, i])  
 count += 1  
 **if** count == 0:  
 **return** 0  
 **return** mae / count  
  
  
**def item\_based\_test**(name, train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb):  
 maes = np.zeros(31)  
 ks = np.zeros(31)  
  
 maes[0] = count\_mae\_ib(train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb, 1)  
 ks[0] = 1  
  
 **for** i **in** xrange(1, 31):  
 **print** i \* 5  
 maes[i] = count\_mae\_ib(train\_user\_item\_matrix,  
 test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb, i \* 5)  
 ks[i] = i \* 5  
  
 plt.title(name)  
 plt.xlabel('Number of neighbours')  
 plt.ylabel('MAE')  
 plt.plot(ks, maes)  
 plt.savefig(name + '.png')  
 plt.clf()  
  
  
**def item\_based**(name, train\_user\_item\_matrix\_a, train\_user\_item\_matrix, user\_numb, train\_item\_numb):  
 sim\_matrix = np.zeros(shape=(train\_item\_numb, train\_item\_numb))  
  
 **for** i **in** xrange(train\_item\_numb):  
 **for** j **in** xrange(i + 1, train\_item\_numb):  
 mult\_sum = 0  
 r1\_sum = 0  
 r2\_sum = 0  
 **for** k **in** xrange(user\_numb):  
 r1 = train\_user\_item\_matrix\_a[k, i]  
 r2 = train\_user\_item\_matrix\_a[k, j]  
 **if** r1 != 0 **and** r2 != 0:  
 mult\_sum += r1 \* r2  
 r1\_sum += r1 \*\* 2  
 r2\_sum += r2 \*\* 2  
 **if** r1\_sum != 0 **and** r2\_sum != 0:  
 sim = mult\_sum / (math.sqrt(r1\_sum) \* math.sqrt(r2\_sum))  
 sim\_matrix[i, j] = sim  
 sim\_matrix[j, i] = sim  
  
 sim\_sorted = np.zeros(shape=(train\_item\_numb, train\_item\_numb, 2))  
  
 **for** i **in** xrange(train\_item\_numb):  
 temp = np.zeros(shape=(train\_item\_numb, 2))  
 **for** j **in** xrange(train\_item\_numb):  
 temp[j, 0] = sim\_matrix[i, j]  
 temp[j, 1] = j  
 sim\_sorted[i] = np.array(sorted(temp, key=**lambda** x: x[0], reverse=True))  
  
 test\_data = pd.read\_csv('ua.test', sep='\t')  
 test\_data = np.array(test\_data)  
  
 test\_item\_numb = 0  
 **for** i **in** xrange(len(test\_data)):  
 **if** test\_data[i][1] > test\_item\_numb:  
 test\_item\_numb = test\_data[i][1]  
  
 test\_user\_item\_matrix = np.zeros(shape=(user\_numb, test\_item\_numb))  
  
 **for** i **in** xrange(len(test\_data)):  
 test\_user\_item\_matrix[test\_data[i][0] - 1, test\_data[i][1] - 1] = test\_data[i][2]  
  
 item\_based\_test(name, train\_user\_item\_matrix, test\_user\_item\_matrix, sim\_sorted, user\_numb, test\_item\_numb)  
  
  
**def task**():  
  
 train\_data = pd.read\_csv('ua.base', sep='\t')  
 train\_data = np.array(train\_data)  
  
 train\_item\_numb = 0  
 user\_numb = 0  
 **for** i **in** xrange(len(train\_data)):  
 **if** train\_data[i][0] > user\_numb:  
 user\_numb = train\_data[i][0]  
 **if** train\_data[i][1] > train\_item\_numb:  
 train\_item\_numb = train\_data[i][1]  
  
 train\_user\_item\_matrix = np.zeros(shape=(user\_numb, train\_item\_numb))  
  
 **for** i **in** xrange(len(train\_data)):  
 train\_user\_item\_matrix[train\_data[i][0] - 1, train\_data[i][1] - 1] = train\_data[i][2]  
  
 user\_based('User-based MAE', train\_user\_item\_matrix, train\_user\_item\_matrix, user\_numb, train\_item\_numb)  
 item\_based('Item-based MAE', train\_user\_item\_matrix, train\_user\_item\_matrix, user\_numb, train\_item\_numb)  
  
 averages\_users = count\_averages\_users(train\_user\_item\_matrix, user\_numb, train\_item\_numb)  
  
 train\_user\_item\_matrix\_au = np.zeros(shape=(user\_numb, train\_item\_numb))  
  
 **for** i **in** xrange(user\_numb):  
 **for** j **in** xrange(train\_item\_numb):  
 **if** train\_user\_item\_matrix[i, j] != 0:  
 train\_user\_item\_matrix\_au[i, j] = train\_user\_item\_matrix[i, j] - averages\_users[i]  
  
 user\_based('User-based MAE with averages subtraction', train\_user\_item\_matrix\_au,  
 train\_user\_item\_matrix, user\_numb, train\_item\_numb)  
  
 averages\_items = count\_averages\_items(train\_user\_item\_matrix, user\_numb, train\_item\_numb)  
  
 train\_user\_item\_matrix\_ai = np.zeros(shape=(user\_numb, train\_item\_numb))  
  
 **for** i **in** xrange(train\_item\_numb):  
 **for** j **in** xrange(user\_numb):  
 **if** train\_user\_item\_matrix[j, i] != 0:  
 train\_user\_item\_matrix\_ai[j, i] = train\_user\_item\_matrix[j, i] - averages\_items[i]  
  
 item\_based('Item-based MAE with averages subtraction', train\_user\_item\_matrix\_ai, train\_user\_item\_matrix,  
 user\_numb, train\_item\_numb)  
  
task()