НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:
«Лабораторная работа №17»

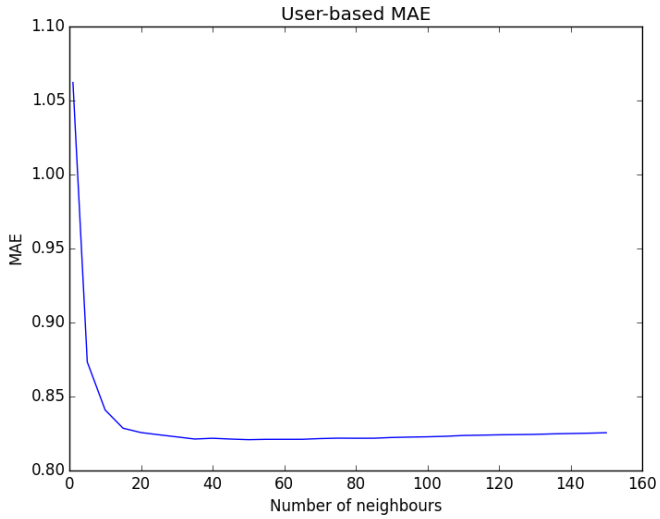
Выполнил: Осипов Лев,
студент группы БПИ121 (1).

Москва, 2015 г.

СОДЕРЖАНИЕ

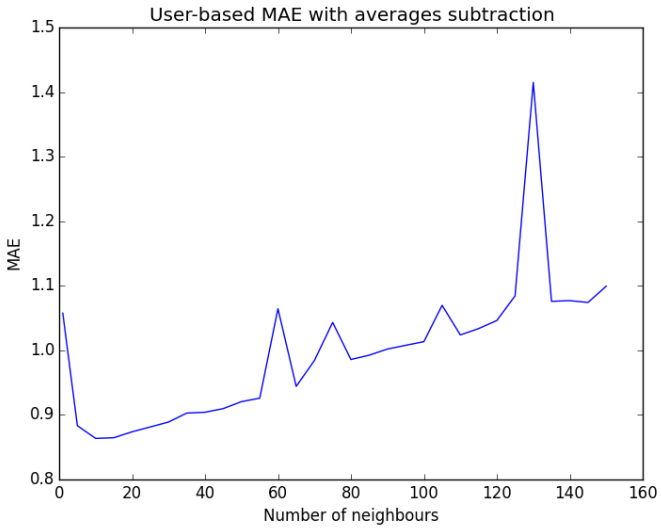
ПРАКТИЧЕСКАЯ ЧАСТЬ

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:

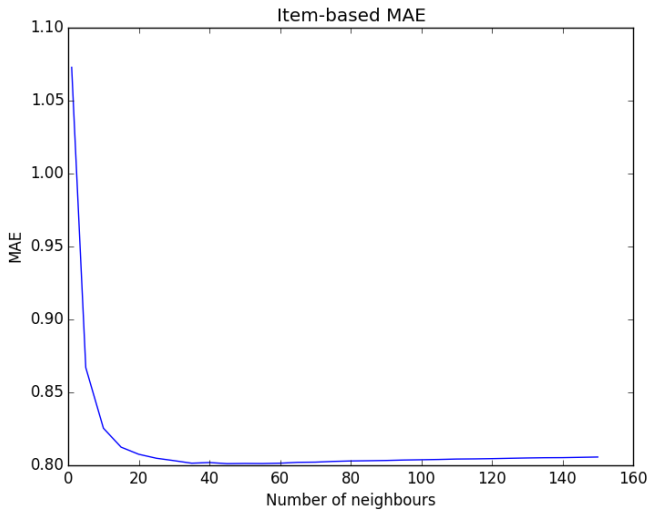


Зависимость ошибки от количества ближайших соседей (с вычитанием среднего), user-based подход

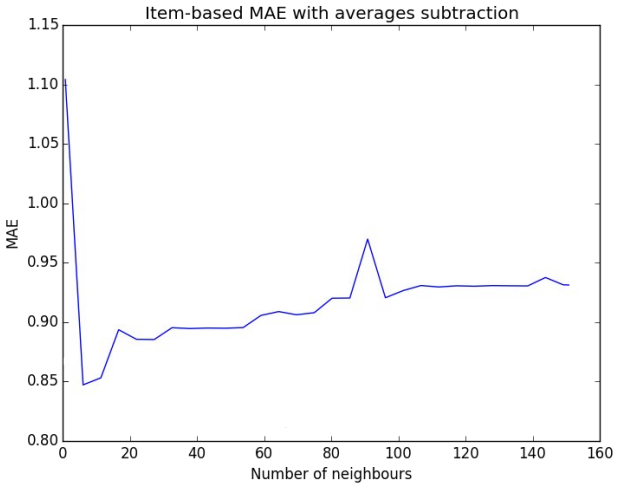
В целом следует выделить тенденцию резкого снижения количества ошибок до момента, когда число соседей становится примерно 15 (в общем случае от 10 до 20). После этого оно начинает увеличиваться.

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()
```