CS589 Homework3

Yifu Liu

April 2018

1 Question 1

1. Base on the program, the $\bar{x} = 48$, and $\bar{y} = 48.3$. Calculating by the β function, the value of $\beta_1 = 1$. If the point(48, 50) was changed to (48, 60), the $\bar{y} = 48.3 + (10\ /\ 10) = 49.3$, the $_1$ will not be change since the function:
   $\beta_1 = \frac{\sum_1^{10}(x_i - \bar{x})(y_i - (\bar{y}+1))}{\sum_1^{10}(x_i - \bar{x})^2} = 1$, since when we factor out the 1 from the function, we got $\frac{\sum_1^{10}(x_i - \bar{x})(1)}{\sum_1^{10}(x_i - \bar{x})^2} = 0$, where $x_i = 48$. Since $\beta_1 = 1.0$, $\beta_0 = 1.0 + 0.3 = 1.3$

2 Question 2

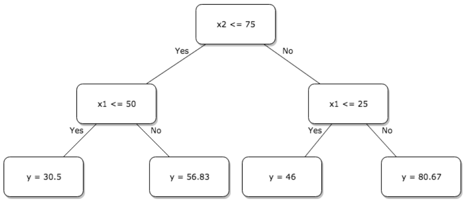

Figure 1: Graph for question 2

1. Please look at the code for with comment "For question 2", the result of the program is ==> 75 y-axis, 50 x-axis, 25 x-axis.

3 Question 3

1. If all the points are in the shaded area, the loss function will not be change, which is true. However, the outside points will affect the loss and have different loss if the boundaries of the shaded region. So, that's possible the loss function will be changed.

2. I2 is included in the loss function, so even slightly different slope will make the result changed.

4 Question 4

The answer of question 4 is: 0.287619686546, please look at the program for detail.

```python
import math
import scipy.stats
import numpy as np
import sklearn.metrics as met


def question_one(x):
    y = x + 0.3
    array = np.array([(x_i, y_i) for (x_i, y_i) in zip(x, y)])
    print("the average of x = ", np.average(array[:, 0]), "; the average of y = ", np.average(array[:, 1]))
    return beta_function(array[:, 0], array[:, 1])


def question_two(dimensions, maxD):
    values = split_helper(dimensions, maxD)
    print(values)
    return values


def split_helper(dimensions, maxD):
    maes = []
    split_points_x = [0, 25, 50, 75, 100]
    if maxD != 0:
        for x in range(0, 2):
            for i in split_points_x:
                d1 = {}
                d2 = {}
                for k, v in dimensions.items():
                    if k[x] <= i:
                        d1.update({k: v})
                    else:
                        d2.update({k: v})
                mae_value = mean_absolute_error(d1, d2)
                maes.append([i, x, mae_value])
        minimum = sorted((el for el in maes), key=lambda L: L[2], reverse = False)[0]
        print(minimum)
        cut(minimum[0], minimum[1], dimensions, maxD)
    else:
        return


def cut(split_value, axis, dimensions, maxD):
    d1 = {}
    d2 = {}
    if axis == 0:
        print(split_value, "x")
    else:
        print(split_value, "y")
    for k, v in dimensions.items():
        if k[axis] <= split_value:
            d1.update({k: v})
        else:
            d2.update({k: v})
    split_helper(d1, maxD - 1)
    split_helper(d2, maxD - 1)


def mean_absolute_error(part1, part2):
    if len(part1) == 0 or len(part2) == 0:
        return float("inf")
    else:
        part1_ave = [sum(part1.values()) / len(part1)] * len(part1)
        part2_ave = [sum(part2.values()) / len(part2)] * len(part2)
        mae_part1 = met.mean_absolute_error(list(part1.values()), part1_ave)
        mae_part2 = met.mean_absolute_error(list(part2.values()), part2_ave)
        return mae_part1 + mae_part2

    # if len(part1) == 0 and len(part2) != 0:
    #     part2_ave = [sum(part2.values()) / len(part2)] * len(part2)
    #     mae_part2 = met.mean_absolute_error(list(part2.values()), part2_ave)
    #     return mae_part2
    # elif len(part2) == 0 and len(part1) != 0:
    #     part1_ave = [sum(part1.values()) / len(part1)] * len(part1)
    #     mae_part1 = met.mean_absolute_error(list(part1.values()), part1_ave)
    #     return mae_part1
    # elif len(part2) != 0 and len(part1) != 0:
    #     part1_ave = [sum(part1.values()) / len(part1)] * len(part1)
    #     part2_ave = [sum(part2.values()) / len(part2)] * len(part2)
    #     mae_part1 = met.mean_absolute_error(list(part1.values()), part1_ave)
    #     mae_part2 = met.mean_absolute_error(list(part2.values()), part2_ave)
    #     return mae_part1 + mae_part2
    # else:
    #     return float(0)


def question_four(x_values, y_values):
    beta_1 = beta_function(x_values, y_values)
    x_ave = np.average(x_values)
    y_ave = np.average(y_values)
    beta_0 = y_ave - beta_1 * x_ave
    temp = 54 * beta_1 + beta_0
    y_hat = x_values * beta_1 + beta_0
    sig_square = 0
    for a, b in zip(y_values, y_hat):
        sig_square = sig_square + (((a - b) ** 2) / len(y_values))
    p = scipy.stats.norm(temp, math.sqrt(sig_square)).cdf(temp - (65 - temp))
    return p


def beta_function(x, y):
    return sum([(x_i - np.average(x)) * (y_i - np.average(y)) for (x_i, y_i) in zip(x, y)]) / sum([(x_i - np.average(x)) ** 2 for x_i in x])


if __name__ == '__main__':
    x_values_1 = np.array([8, 21, 33, 35, 37, 48, 58, 77, 79, 84])                #for question 1
    x_values_4 = np.array([88, 98, 31, 15, 66, 49, 9, 69, 55, 60])                #for question 4
    y_values_4 = np.array([93, 90, 41, 17, 75, 51, 18, 85, 78, 50])               #for question 4
    points = {(12, 12): 12, (12, 37): 24, (12, 62): 42, (12, 87): 46,
              (37, 12): 18, (37, 37): 40, (37, 62): 47, (37, 87): 71,
              (62, 12): 37, (62, 37): 53, (62, 62): 58, (62, 87): 80,
              (87, 12): 49, (87, 37): 63, (87, 62): 81, (87, 87): 91}
    question_two(points, 3)
    answer1 = question_one(x_values_1)
    print("without the value change, the value of beta_1 is: ", answer1)
    answer4 = question_four(x_values_4, y_values_4)
    print("the answer of question 4 is: ", answer4)
```