COMP9417 HomeWork1

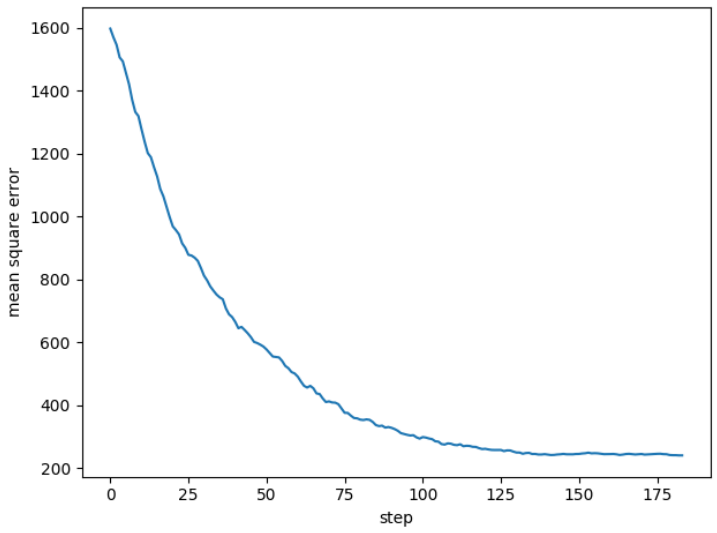
Ping GAO, zid: z5163482

March 16, 2020

1 Q1

The θ_0 using house age i get is 23.6 and θ_1 is 24.1.

2 Q2



3 Q3

The RMSE for traning set when i use house age feature is 15.49.

4 Q4

The RMSE for test set when i use house age feature is 32.39.

5 Q5

The RMSE for test set when i use distance to the nearest station feature is 23.38.

6 Q6

The RMSE for test set when i use number of convenience stores feature is 14.55.

7 Q7

All three my RMSE results from the test sets are really close to the RMSE results i get from the traning sets (< 20) that means all three features fit linear regression model really well. 1st feature is convenience stores, 2th feature is nearest MRT station, 3th feature is house age. This is done by first rank their test and training RMSE sum and then inpect the difference between test RMSE and training RMSE, if the difference is large it means this fearture is less coherent.

8 My Program Result From Terminal

RMSE for test set using house age 32.39107298710752
RESE for training set usinghouse age 15.494062138722894
RMSE for test set using distance to the nearest MRT station 23.37940467352361
RESE for training set usingdistance to the nearest MRT station 16.662894453630656
RMSE for test set using number of convenience stores 14.553252774984013
RESE for training set usingnumber of convenience stores 9.829368792205239

9 My Code

```
import sys
from math import sqrt

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
import numpy as np


class TrainningSet:
    def __init__(self, filename):
        self.df = pd.read_csv(filename)
        self.normalized_df = None
        self.data_train = None
        self.data_test = None
        self.target_train = None
        self.target_test = None
        self.theta = None
        self.rmse_train = None
        self.rmse_test = None

    def preprocessing(self):
        self.normalized_feature()
        self.split_dataframe()

    def normalized_feature(self):
        self.normalized_df = self.df.copy()
        for feature_name in self.normalized_df.columns:
            # skip non-feature
            if feature_name == 'house_price_of_unit_area':
                continue
            if feature_name == 'No':
                continue
            max_value = self.normalized_df[feature_name].max()
            mini_value = self.normalized_df[feature_name].min()
            self.normalized_df[feature_name] = (self.normalized_df[feature_name] - mini_value) / (
                    max_value - mini_value)

    def split_dataframe(self):
        data = self.normalized_df[
            [col for col in self.normalized_df.columns if col != 'house_price_of_unit_area']]
        target = self.normalized_df['house_price_of_unit_area']
        # split the test and train data set
        self.data_train, self.data_test, self.target_train, self.target_test = train_test_split(data, target,
                                                                                                 test_size=0.25
                                                                                                 shuffle=False)

        # should use data_train ans target_train

    def stochastic_gradient_descent(self, feature_name, data, target):
        # initialize with random value
        self.theta = np.array([-1, -0.5])
        max_iteration = 50
        learning_rate = 0.01
        eps = 0.001
        converge = False
        iteration = 0
        steps = 0
        mean_square_error = []
        while not converge and iteration < max_iteration:
            for index, row in data.iterrows():
                x = np.array([1, row[feature_name]])
                theta_old = self.theta
                self.theta = theta_old + learning_rate * (target[index] - np.dot(x, theta_old))
                # print(self.theta, theta_old)
                mean_square_error.append(self.compute_cost_function(feature_name, data, target))
                steps += 1
                if (abs(theta_old[0] - self.theta[0]) < eps) and (abs(theta_old[1] - self.theta[1]) < eps):
                    converge = True
                    break
            iteration += 1
        # fig = plt.figure()
        d = {'step': np.array(list(range(steps))), 'mean_square_error': np.array(mean_square_error)}
        pd_plot = pd.DataFrame(d)
        sns.lineplot(x='step', y='mean_square_error', data=pd_plot)
        # plt.show()
```

```
        # fig.savefig('plot.png')
        # print(self.theta)

    def compute_cost_function(self, feature_name, data, target):
        sum = 0
        for index, row in data.iterrows():
            x = np.array([1, row[feature_name]])
            sum += pow(target[index] - np.dot(x, self.theta), 2)
        # print(sum/self.data_train.shape[0])
        return sum / data.shape[0]

    def evaluation(self, feature_name):
        self.rmse_train = sqrt(self.compute_cost_function(feature_name, self.data_train, self.target_train))
        self.stochastic_gradient_descent(feature_name, self.data_test, self.target_test)
        self.rmse_test = sqrt(self.compute_cost_function(feature_name, self.data_test, self.target_test))
        print("RMSE_for_test_set_using_" + feature_name + "_" + str(self.rmse_test))
        print("RESE_for_training_set_using" + feature_name + "_" + str(self.rmse_train))

    def print_normalized_df(self):
        print(self.normalized_df.head().to_string())


if __name__ == '__main__':
    train_set = TrainningSet(filename='house_prices.csv')
    train_set.preprocessing()
    # use house age to predict house price
    train_set.stochastic_gradient_descent('house_age', train_set.data_train, train_set.target_train)
    train_set.evaluation('house_age')
    # use distance to the nearest MRT station to predict house price
    train_set.stochastic_gradient_descent('distance_to_the_nearest_MRT_station', train_set.data_train,
                                          train_set.target_train)
    train_set.evaluation('distance_to_the_nearest_MRT_station')
    # use number of convenience stores to predict house price
    train_set.stochastic_gradient_descent('number_of_convenience_stores', train_set.data_train,
                                          train_set.target_train)
    train_set.evaluation('number_of_convenience_stores')
```