Xuenan Wang (xw336)

December 2, 2019

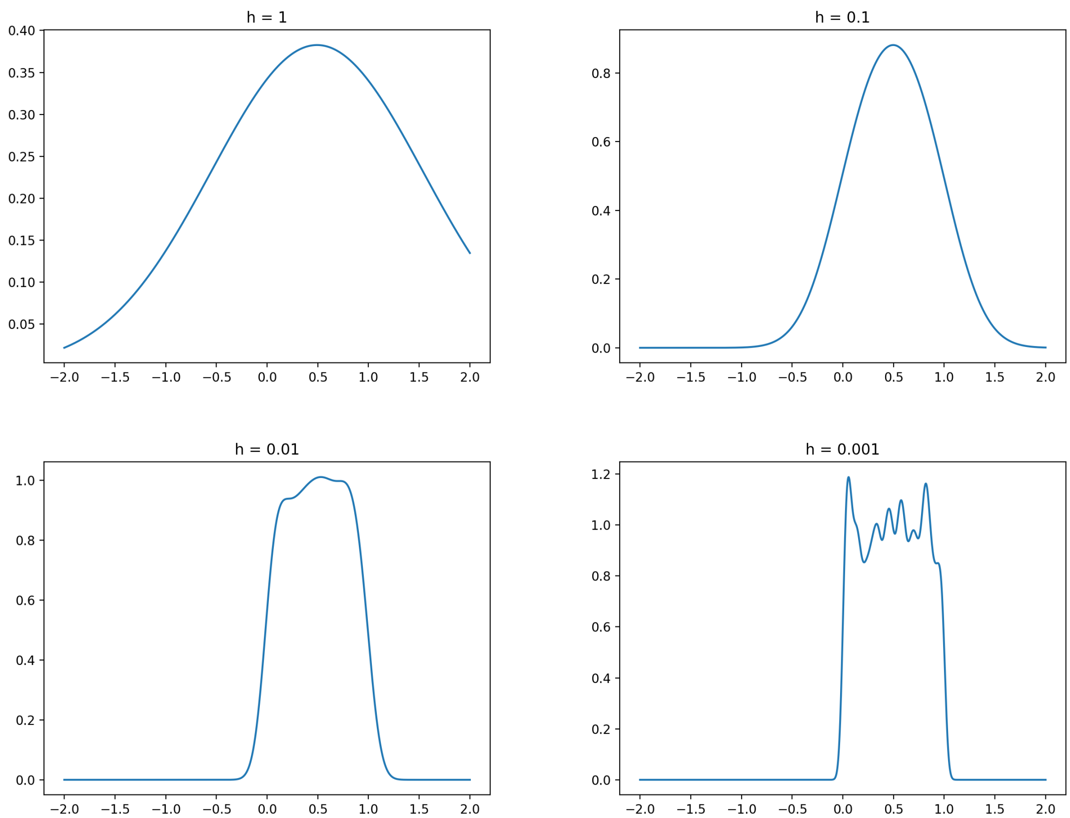
Homework #4

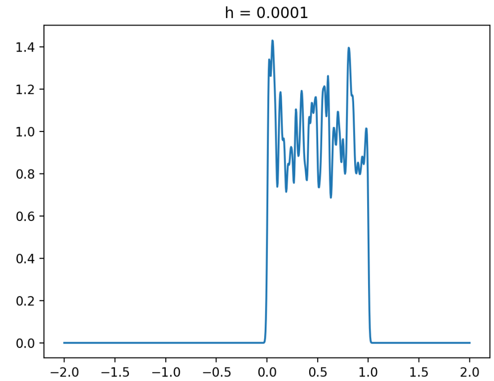
Foundations of Computer and Data Science CS-596

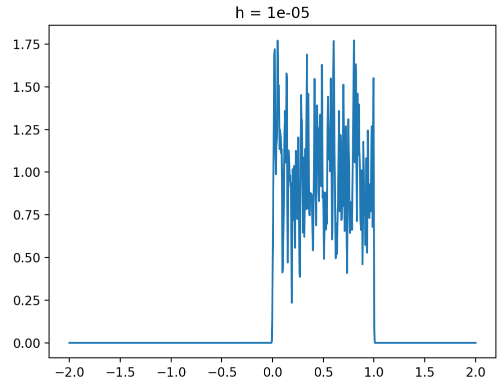
Problem 1:

a) Gaussian Kernel.

$$K(x,h) = \frac{1}{\sqrt{2\pi h}} e^{-\frac{1}{2h}x^2}$$

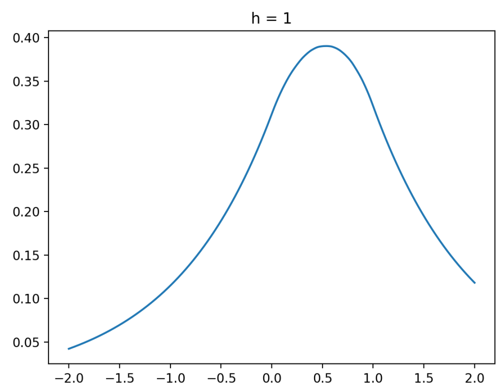


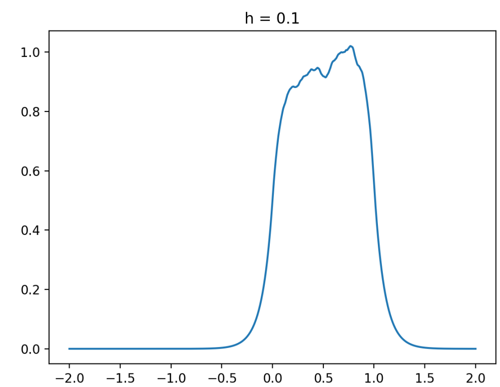


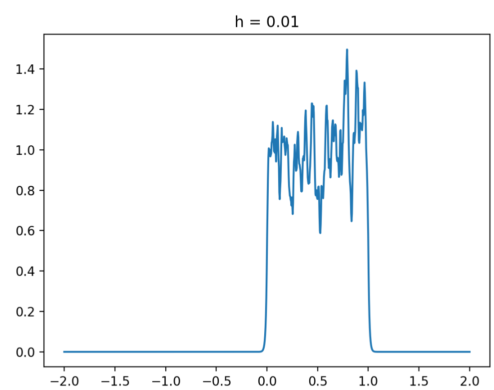


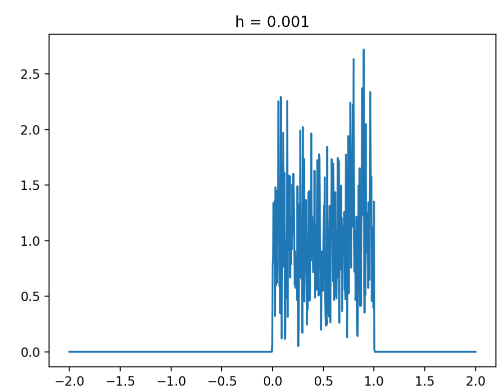
b) Laplacian Kernel.

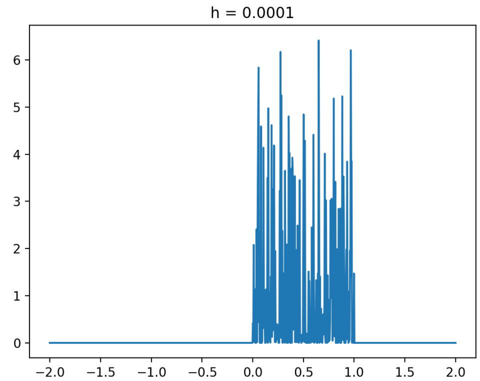
$$K(x,h) = \frac{1}{2h}e^{-\frac{1}{h}|x|}$$

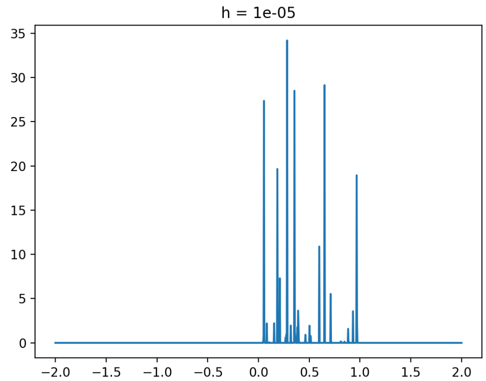












Problem 2:

a)

$$\min\{\sum_{X_i \in stars}(1-\phi(X_i))^2 + \sum_{X_j \in circles}(1+\phi(X_j))^2 + \lambda\|\phi(X)\|^2\}$$

$$= \min\{\sum_{X_i \in stars}(1-\sum q_i K(X_i, X_t))^2 + \sum_{X_j \in circles}(1+\sum q_i K(X_j, X_t))^2 + \lambda\|\phi(X)\|^2\}$$

$$\phi = \sum q_i \hat{\phi}(X_i) + U, \ < U, \hat{\phi}(X_i) > = 0$$

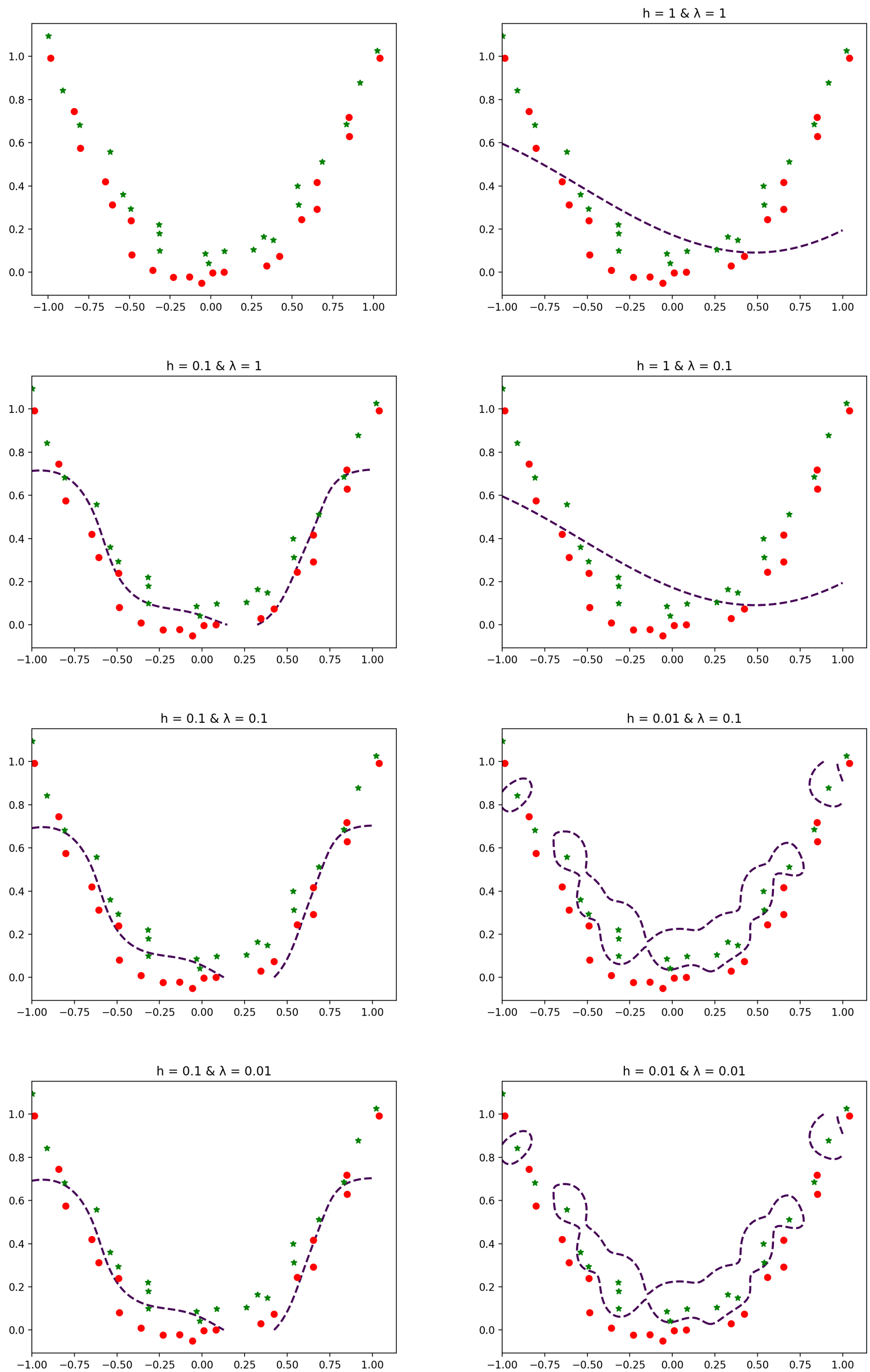
$$\hat{\phi}(X_j) = < \sum q_i \hat{\phi}(X_i) + U, \hat{\phi}(X_j) > = \sum q_i < \hat{\phi}(X_i), \hat{\phi}(X_j) >$$

b)

$$\|\phi(X)\|^2$$
$$= \|\phi(X) - \hat{\phi}(X) + \hat{\phi}(X)\|^2$$
$$= \|\phi(X) - \hat{\phi}(X)\|^2 + \|\hat{\phi}(X)\|^2$$
$$\geqslant \|\phi(X)\|^2$$

c) Figure 1: input data(**RED**: circles, **GREEN**: stars)

Figure 2-8: classifier(**dashed PURPLE line**) with different h and λ


h = 1 & λ = 1
h = 0.1 & λ = 1
h = 1 & λ = 0.1
h = 0.1 & λ = 0.1
h = 0.01 & λ = 0.1
h = 0.1 & λ = 0.01
h = 0.01 & λ = 0.01

Problem 3:

a) $\theta_{optimum} = \theta_*$

$E[y] = E[\theta_*^T X + \omega] = E[\theta_*^T X] + E[\omega]$

We know that ω is a scalar random variable with mean 0, therefore $E[\omega] = 0$, $E[y] = E[\theta_*^T X]$.

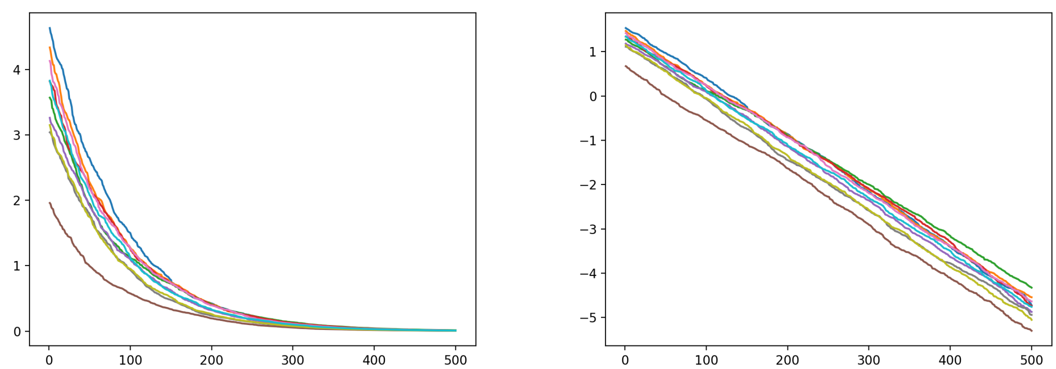
$(y - \theta^T X)^2 \geqslant 0$, which means that $min(y - \theta^T X)^2 = min(y - \theta^T X) = 0$. If and only if $y = \theta^T X$ can $(y - \theta^T X)^2$ have minimal value of 0.

$E[y] = E[\theta_*^T X] = E[\theta^T X]$

We have that the optimum θ is equal to θ_*.

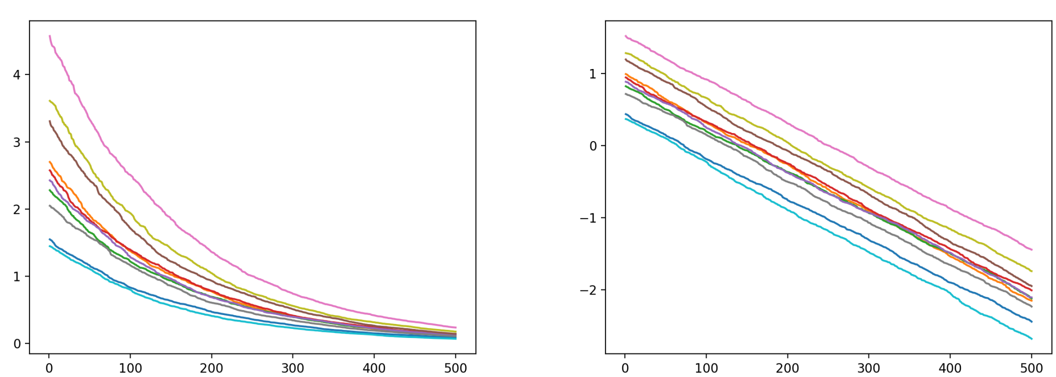
b) LMS: $\theta_t = \theta_{t-1} - \mu(y_t - \theta_{t-1}^T X_t)X_t$

c) $\theta_* = [-0.8,\ -0.3,\ 0,\ 0.3,\ 0.8]$

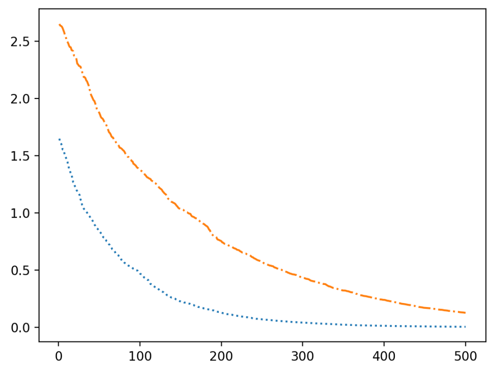


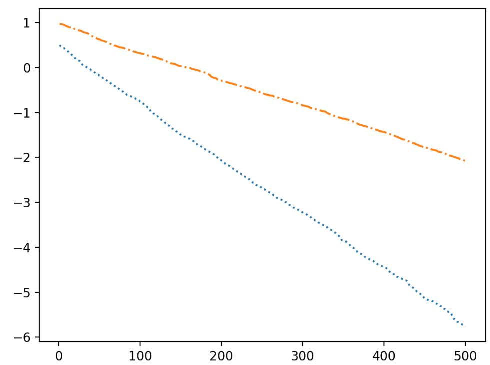
The left one is in normal scale and right one is in logarithmic scale. There are ten lines in each figure representing a one-time simulation. We can see that these lines converge at iteration of 400 times. Learning rate here is 0.001.

d) $\theta_* = [-0.8,\ -0.3,\ 0,\ 0.3,\ 0.8]$



The left one is in normal scale and right one is in logarithmic scale. Same as previous one. We can see that these lines still not converge at iteration of 500 times. Learning rate here is 0.0005.





We have here two learning rates plotted in one figure. The ‘dotted’ line is learning rate of 0.001 and the ‘dashdot’ line is learning rate of 0.0005. We can clearly see that higher learning rate will result in faster convergence. However, this is not a proof that claiming learning rate of 0.001 is better that 0.0005. To discuss the performance of an algorithm, we need to also consider the learning outcomes, which we don’t have it here. Therefore, we can not say which one is better based on our limited knowledge.

Source Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date    : Dec.05 2019
# @Author  : Xuenan(Roderick) Wang
# @Email   : roderick_wang@outlook.com
# @GitHub  : https://github.com/hello-roderickwang

import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
import pandas as pd
from sklearn import svm


class Problem1:
    def a(self):
        r = np.random.rand(1000, 1)
        x = np.linspace(-2.0, 2, num=1000)
        h = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]
        for k in range(len(h)):
            y = np.array([])
            for i in range(1000):
                temp = 0
                for j in range(1000):
                    temp += 1 / np.sqrt(2 * np.pi * h[k]) *
np.exp(-1 / (2 * h[k]) * (x[i] - r[j]) ** 2)
                temp /= 1000
                y = np.append(y, temp)
            plt.figure(k + 1)
            plt.plot(x, y)
            plt.title('h = ' + str(h[k]))
        plt.show()

    def b(self):
        r = np.random.rand(1000, 1)
        x = np.linspace(-2.0, 2, num=1000)
        h = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]
        for k in range(len(h)):
            y = np.array([])
            for i in range(1000):
                temp = 0
                for j in range(1000):
                    temp += 1 / (2 * h[k]) * np.exp(-1 / h[k] *
np.abs(x[i] - r[j]))
                temp /= 1000
                y = np.append(y, temp)
            plt.figure(k + 1)
            plt.plot(x, y)
            plt.title('h = ' + str(h[k]))
        plt.show()

class Problem2:
    def __init__(self):
        self.mat = {}
        self.K = 0

    def load_mat(self):
        self.mat = sio.loadmat('./hw4-2data.mat')
        print('mat.keys():\n')
        print(self.mat.keys())
        print('circles:\n')
        print(self.mat['circles'])
        print('len(circles): ', len(self.mat['circles']))
        print('stars:\n')
        print(self.mat['stars'])
        print('len(stars): ', len(self.mat['stars']))

    def plot_data(self):
        plt.figure(1)
        for i in range(len(self.mat['circles'])):
            plt.plot(self.mat['circles'][i][0], self.mat['circles'][i]
[1], 'ro')
            plt.plot(self.mat['stars'][i][0], self.mat['stars'][i][1],
'g*')
        # plt.show()

    def gaussian_kernel(self, X, y):
        h = 0.1
        return np.exp((-1/h)*np.linalg.norm(X, y)**2)

    def classifier(self, v_h, v_lambda):
        model = svm.SVC(kernel='rbf', gamma=1/v_h,
C=v_lambda)
        xx, yy = np.meshgrid(np.linspace(-1, 1, 200),
np.linspace(0, 1, 100))
        X = np.concatenate((self.mat['circles'],
self.mat['stars']), axis=0)
        y = np.concatenate((-1*np.ones(21), np.ones(21)),
axis=0)
        model.fit(X, y)
        df = model.decision_function(np.c_[xx.ravel(),
yy.ravel()])
        df = df.reshape(xx.shape)
        plt.figure(2)
        for i in range(len(self.mat['circles'])):
            plt.plot(self.mat['circles'][i][0], self.mat['circles'][i]
[1], 'ro')
            plt.plot(self.mat['stars'][i][0], self.mat['stars'][i][1],
'g*')
        # plt.imshow(df, interpolation='nearest',
        #            extent=(xx.min(), xx.max(), yy.min(),
yy.max()),
        #            aspect='auto', origin='lower',
cmap=plt.cm.PuOr_r)
        plt.contour(xx, yy, df, levels=[0], linewidths=2,
                    linestyles='dashed')
        plt.title('h = '+str(v_h)+' & \u03BB =
'+str(v_lambda))
        plt.show()


class Problem3:
    def __init__(self):
        self.X = []
        self.y = []
```

```
        self.data = []
        self.omega = []
        self.theta_star = np.array([-0.8, -0.3, 0, 0.3, 0.8])
        self.theta_array = []
        self.e_array = []
        self.log_array = []

    def get_data(self):
        self.X = []
        self.omega = []
        self.y = []
        self.data = []
        for i in range(6):
            self.X.append(np.random.normal(loc=0,
scale=1, size=(5)))
            self.omega.append(np.random.normal(loc=0,
scale=0.1))
            self.y.append(np.dot(self.theta_star.transpose(),
self.X[i])+self.omega[i])
            self.data.append([self.y[i], self.X[i]])

    def LMS(self, theta, mu=0):
        n = 0
        e = 0
        mu_array = [0.001, 0.0005]
        e_max = 1000000000000
        n_max = 500
        self.e_array = []
        self.log_array = []
        while n<n_max and e<e_max:
            e = 0
            self.get_data()
            for pair in self.data:
                thetaX = np.dot(theta.transpose(), pair[1])
                # print('thetaX: ', thetaX)
                theta += np.dot(mu_array[mu]*(pair[0]-
thetaX), pair[1])
                # print('theta: ', theta)
                # e += np.power(pair[0]-thetaX, 2)
                e = np.power(np.linalg.norm(theta-
self.theta_star), 2)
                # print('e: ', e)
            self.e_array.append(e)
            self.log_array.append(np.log(e))
            # self.e_array.append(np.abs(pair[0]-thetaX))
            n += 1
            # print('n: ', n)
        print('total n: ', n)

    def train(self):
        # plt.figure(1)
        # for i in range(10):
        #     print('--------LMS: ' + str(i) + '--------')
        #     print('e_array: ', self.e_array)
        #     for j in range(5):
        #         self.theta_array.append(np.random.rand())
        #     self.LMS(np.array(self.theta_array))
        #     self.theta_array = []
        #     self.plot()
        for i in range(1):
            print('--------LMS: ' + str(i) + '--------')
            print('e_array: ', self.e_array)
            for j in range(5):
                self.theta_array.append(np.random.rand())
            self.LMS(np.array(self.theta_array), mu=0)
            self.theta_array = []
            self.plot('dotted')
        for i in range(1):
            print('--------LMS: ' + str(i) + '--------')
            print('e_array: ', self.e_array)
            for j in range(5):
                self.theta_array.append(np.random.rand())
            self.LMS(np.array(self.theta_array), mu=1)
            self.theta_array = []
            self.plot('dashdot')
        plt.show()

    def plot(self, linestyle='-'):
        plt.figure(1)
        plt.plot(range(1, len(self.e_array)+1), self.e_array,
linestyle=linestyle)
        plt.figure(2)
        plt.plot(range(1, len(self.log_array)+1),
self.log_array, linestyle=linestyle)

if __name__ == '__main__':
    # Problem1 = Problem1()
    # print('Solution for Problem 1 a):')
    # Problem1.a()
    # print('Solution for Problem 1 b):')
    # Problem1.b()

    Problem2 = Problem2()
    Problem2.load_mat()
    # Problem2.plot_data()
    Problem2.classifier(0.1, 1)

    # Problem3 = Problem3()
    # Problem3.train()
```