```
import numpy as np
from sklearn.datasets import load_boston
from sklearn import preprocessing
boston = load_boston()
X = boston.data
y = boston.target



X_train = X[0:400,:]
X_test = X[400:, :]
y_train = y[0:400]
y_test = y[400:]


X_train_mean = np.mean(X_train, axis=0)
X_train_std = np.std(X_train, axis=0)

X_train_mean = np.reshape(X_train_mean, (1,13))
X_train_std = np.reshape(X_train_std, (1,13))

for index in range(X_train.shape[0]):
	X_train[index,:] = np.divide(np.subtract(X_train[index,:], X_train_mean), X_train_std)
	if index < 106:
		X_test[index,:] = np.divide(np.subtract(X_test[index,:], X_train_mean), X_train_std)


#X_train_mean = np.mean(X_train, axis=0)
#X_train_std = np.std(X_train, axis=0)
#X_train_var = np.var(X_train, axis=0)
#print X_train_mean
#print X_train_var
#print X_train_std


##########################################################################################
########################
# Question 1.a

A = np.concatenate((np.ones((X_train.shape[0], 1)), X_train), axis=1)
theta = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(A), A)), np.transpose(A)), y_train)

#A_T_A_inv = np.linalg.inv(A_T_A)
#A_T_A_inv_A_T = np.dot(A_T_A_inv, np.transpose(A))
#theta = np.dot(A_T_A_inv_A_T, y_train)
X_test_tilde = np.concatenate((np.ones((X_test.shape[0], 1)), X_test), axis=1)
MSE_LS = 1/float(X_test.shape[0])*np.dot(np.transpose(y_test - np.dot(X_test_tilde, theta)), (y_test -
np.dot(X_test_tilde, theta)))
```

```
#test = y_test - np.dot(X_test_tilde, theta)
#test1 = np.dot(np.transpose(test), test)
print 'MSE Least Square: ', MSE_LS

#############################################################################################
#########################
# Question 1.b

X_train_fit = X_train[0:300,:]
X_train_hold = X_train[300:,:]
y_train_fit = y_train[0:300]
y_train_hold = y_train[300:]

#Lambda = 0.001

A1 = np.concatenate((np.ones((X_train_fit.shape[0], 1)), X_train_fit), axis=1)
A1_T_A1 = np.dot(np.transpose(A1),A1);
#while Lambda < 2:
#	theta1 = np.dot(np.dot(np.linalg.inv(A1_T_A1 + Lambda * np.identity(A1_T_A1.shape[0])), np.transpose(A1)), y_train_fit)
#	X_train_hold_tilde = np.concatenate((np.ones((X_train_hold.shape[0], 1)), X_train_hold), axis=1)

# Lambda = 1.5 is the value that minimize this MSE, so we will use it on our
#	MSE_R_hold = 1/float(X_train_hold.shape[0])*np.dot(np.transpose(y_train_hold - np.dot(X_train_hold_tilde, theta1)), (y_train_hold - np.dot(X_train_hold_tilde, theta1)))
#	print MSE_R_hold, Lambda
#	Lambda += 0.001


# Getting theta using all the training data and lambda = 1.507
Lambda = 1.507
A_T_A = np.dot(np.transpose(A), A)
theta1 = np.dot(np.dot(np.linalg.inv(A_T_A + Lambda * np.identity(A_T_A.shape[0])), np.transpose(A)), y_train)

# Compute MSE for ridge regression

MSE_R = 1/float(X_test.shape[0])*np.dot(np.transpose(y_test - np.dot(X_test_tilde, theta1)), (y_test - np.dot(X_test_tilde, theta1)))

print 'MSE Ridge Regression: ', MSE_R

#############################################################################################
#############################
# Question 1.c

# uncomment this part to run the code used to deterine Lambda
```

```
#Lambda =0.001
#from sklearn import linear_model
#while Lambda < 10:
#	reg = linear_model.Lasso(alpha = Lambda)
#	reg.fit(np.concatenate((np.ones((X_train_fit.shape[0], 1)),X_train_fit), axis=1), y_train_fit)
#	MSE_Lasso = 1/float(X_train_fit.shape[0])*np.dot(np.transpose(y_train_hold - reg.predict(X_train_hold_tilde)), (y_train_hold - reg.predict(X_train_hold_tilde)))
#	print MSE_Lasso, Lambda
#	Lambda += 0.001
# The best Lambda minimizing the MSE on the hold data I was able to compute is 3.966

#Lambda = 3.966
Lambda = 0.17
from sklearn import linear_model
reg = linear_model.Lasso(alpha = Lambda)
reg.fit(np.concatenate((np.ones((X_train.shape[0], 1)),X_train), axis=1), y_train)
MSE_Lasso = 1/float(X_test.shape[0])*np.dot(np.transpose(y_test - reg.predict(X_test_tilde)), (y_test - reg.predict(X_test_tilde)))
number_none_zero_theta = np.count_nonzero(reg.coef_)

print 'MSE Lasso: ' , MSE_Lasso
print 'number of non zero theta', number_none_zero_theta
```

```
import numpy as np

np.random.seed(2017)
n = 100
X_train = np.random.rand(n)
y_train = 0.25 + 0.5*X_train + np.sqrt(0.1)*np.random.randn(n)
idx = np.random.randint(0, 100, 10)
y_train[idx] = y_train[idx] + np.random.randn(10)

##########################################################################################
# Question 3.a

X_train_fit = X_train[:50]
X_train_hold = X_train[50:]
y_train_fit = y_train[:50]
y_train_hold = y_train[50:]
X_train_fit = X_train_fit.reshape(X_train_fit.shape[0],1)
X_train_hold = X_train_hold.reshape(X_train_hold.shape[0],1)
X_train = X_train.reshape(X_train.shape[0],1)
X_train_hold_tilde = np.concatenate((np.ones((X_train_hold.shape[0], 1)), X_train_hold), axis=1)

#Lambda = 0.001

#A1 = np.concatenate((np.ones((X_train_fit.shape[0],1)), X_train_fit), axis=1)
#A1_T_A1 = np.dot(np.transpose(A1),A1);
#while Lambda < 20:
#	theta1 = np.dot(np.dot(np.linalg.inv(A1_T_A1 + Lambda * np.identity(A1_T_A1.shape[0])), np.transpose(A1)), y_train_fit)
#	X_train_hold_tilde = np.concatenate((np.ones((X_train_hold.shape[0], 1)), X_train_hold), axis=1)
#	MSE_R_hold = 1/float(X_train_hold.shape[0])*np.dot(np.transpose(y_train_hold - np.dot(X_train_hold_tilde, theta1)), (y_train_hold - np.dot(X_train_hold_tilde, theta1)))
#	print MSE_R_hold, Lambda
#	Lambda += 0.001

#Lambda = 15.887

#Lambda = 0.000000001
#A = np.concatenate((np.ones((X_train.shape[0], 1)), X_train), axis=1)
#A_T_A = np.dot(np.transpose(A), A)
#theta1 = np.dot(np.dot(np.linalg.inv(A_T_A + Lambda * np.identity(A_T_A.shape[0])), np.transpose(A)), y_train)

#print 'intercept: ', theta1[0], 'slope: ', theta1[1]
##############################################################################################
# Question 3.b
```

```
from sklearn import linear_model
#epsilon = 1.0
#alpha = 0.001
#count = 1
#reg = linear_model.HuberRegressor(epsilon, alpha)
#reg.fit(np.concatenate((np.ones((X_train_fit.shape[0], 1)),X_train_fit), axis=1), y_train_fit)
#lowest_MSE = 1/float(X_train_fit.shape[0])*np.dot(np.transpose(y_train_hold -
reg.predict(X_train_hold_tilde)), (y_train_hold - reg.predict(X_train_hold_tilde)))

#while (alpha < 20):
#	reg = linear_model.HuberRegressor(epsilon, alpha)
#	reg.fit(np.concatenate((np.ones((X_train_fit.shape[0], 1)),X_train_fit), axis=1), y_train_fit)
#	MSE = 1/float(X_train_fit.shape[0])*np.dot(np.transpose(y_train_hold -
reg.predict(X_train_hold_tilde)), (y_train_hold - reg.predict(X_train_hold_tilde)))
#	print 'MSE: ', MSE, 'epsilon: ', epsilon, 'alpha: ', alpha
#	if lowest_MSE > MSE:
#		lowest_MSE = MSE
#		epsilon_at_lowest_MSE = epsilon
#		alpha_at_lowest_MSE = alpha
#
#	alpha += 0.001
#print 'lowest MSE: ', lowest_MSE, 'epsilon: ', epsilon_at_lowest_MSE, 'alpha: ',
alpha_at_lowest_MSE

#epsilon = 2.42
#alpha = 3.001

epsilon = 2.42
alpha = 9
reg = linear_model.HuberRegressor(epsilon, alpha)
reg.fit(np.concatenate((np.ones((X_train.shape[0], 1)),X_train), axis=1), y_train)
print reg.intercept_, reg.coef_
###########################################################################################
######################
# Question 3.c

from sklearn.svm import SVR
#reg = SVR(C=5, epsilon = 0.847, kernel = 'poly', degree = 1, coef0 = 1.0, gamma = 2.191)
#C = 2.211
#epsilon = 0.847
#gamma = 2.191
#reg.fit(np.concatenate((np.ones((X_train_fit.shape[0], 1)),X_train_fit), axis=1), y_train_fit)
#lowest_MSE = 1/float(X_train_fit.shape[0])*np.dot(np.transpose(y_train_hold -
reg.predict(X_train_hold_tilde)), (y_train_hold - reg.predict(X_train_hold_tilde)))

#while (gamma < 20):
#	reg = SVR(C = C, epsilon = epsilon, kernel = 'poly', degree = 1, coef0 = 1.0, gamma = gamma)
#	reg.fit(np.concatenate((np.ones((X_train_fit.shape[0], 1)),X_train_fit), axis=1), y_train_fit)
```

```
#	MSE = 1/float(X_train_fit.shape[0])*np.dot(np.transpose(y_train_hold - 
reg.predict(X_train_hold_tilde)), (y_train_hold - reg.predict(X_train_hold_tilde)))
#	print 'MSE: ', MSE, 'epsilon: ', epsilon, 'C: ', C
#	if lowest_MSE > MSE:
#		lowest_MSE = MSE
#		epsilon_at_lowest_MSE = epsilon
#		C_at_lowest_MSE = C
#		gamma_at_lowest_MSE = gamma

#	gamma += 0.001
#print 'lowest MSE: ', lowest_MSE, 'epsilon: ', epsilon_at_lowest_MSE, 'C: ', C_at_lowest_MSE, 
'gamma: ', gamma_at_lowest_MSE

#C = 14
#epsilon = 1
#gamma = 0.1
#reg = SVR(C = C, epsilon = epsilon, kernel = 'poly', degree = 1, coef0 = 1.0, gamma = gamma)
#reg.fit(np.concatenate((np.ones((X_train.shape[0], 1)),X_train), axis=1), y_train)
#x_i = reg.support_vectors_
#dual_coef = reg.dual_coef_
#intercept = reg.intercept_

#coef = np.dot(dual_coef, x_i)
#print coef, intercept


##########################################################################################
####################
##########################################################################################
####################

import numpy as np
from matplotlib import pyplot as plt
np.random.seed(2017)
n = 100
xtrain = np.random.rand(n)
ytrain = np.sin(9*xtrain) + np.sqrt(1/3.0)*np.random.randn(n)

xtest = np.linspace(0,1,1001)
ytest = np.sin(9*xtest)


gamma = 0.25
Lambda = 0.02

K = np.zeros((xtrain.shape[0], xtrain.shape[0]))
for i in range(xtrain.shape[0]):
	K[i,:] = np.exp(-gamma*np.abs(xtrain[i] - xtrain))
```

```
y_hat = np.zeros(xtest.shape[0])
K_I_inv = np.linalg.inv(K + Lambda*np.ones((xtrain.shape[0], xtrain.shape[0])))
k = np.empty((xtest.shape[0], xtrain.shape[0]))
for i in range(xtest.shape[0]):
    k[i,:] = np.exp(-gamma*np.abs(xtest[i] - xtrain))

y_hat = np.dot(np.dot(np.transpose(ytrain), K_I_inv), k.T)

plt.figure(1)
plt.scatter(xtest,ytest, label = 'test data')
plt.scatter(xtrain, ytrain, label = 'train data')
plt.scatter(xtest, y_hat, label = 'estimated data')
plt.legend()
plt.show()


from sklearn.svm import SVR
reg = SVR(C=100, epsilon=0.1, kernel='rbf', gamma=1.0)
reg.fit(xtrain.reshape(-1,1),ytrain)

y_predict = reg.predict(xtest.reshape(-1,1))

plt.figure(2)
plt.scatter(xtest, ytest, label = 'test data')
plt.scatter(xtrain, ytrain, label = 'train data')
plt.scatter(xtest, y_predict, label = 'estimated data')
plt.legend()
plt.show()
```