**CPMS242 Machine Learning**

**Homework 1 Report**

Group Members: Yunzhe Li (#1571061), Yong Deng (#1571065)

Date: Apr.10, 2018

# **10\_Fold Cross Validation**

* Random shuffle the 100 data in the trainset, then divide it into 10 partitions equally.
* Discrete lambda from 0 to 5 with 0.1 interval.
* For each lambda, proceed the 10 fold cross validation. Then I get one test error and one train error for each lambda.
* Find the lambda with minimum test error.
* Use that lambda along with the 100 data in trainset to compute the coefficients w. That’s my optimal 9-th order polynomial model.
* Use the optimal polynomial model to predict the data in testset, and compute the sum of the square error.

Our result shows that,

* The optimal w:

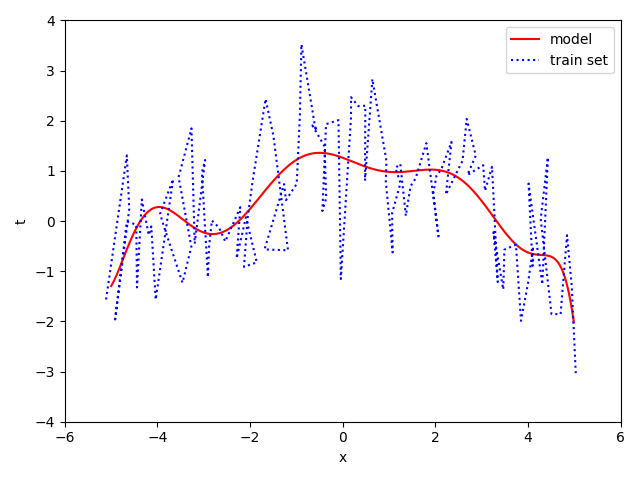
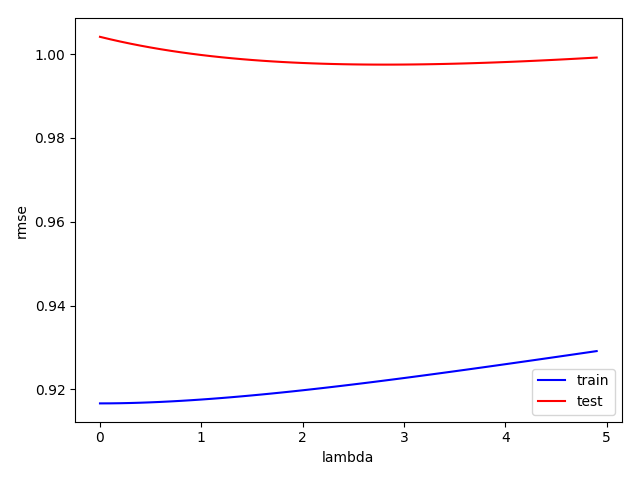
[ 1.17748975e+00 -3.02485606e-01 -1.11894191e-01 2.40628377e-01

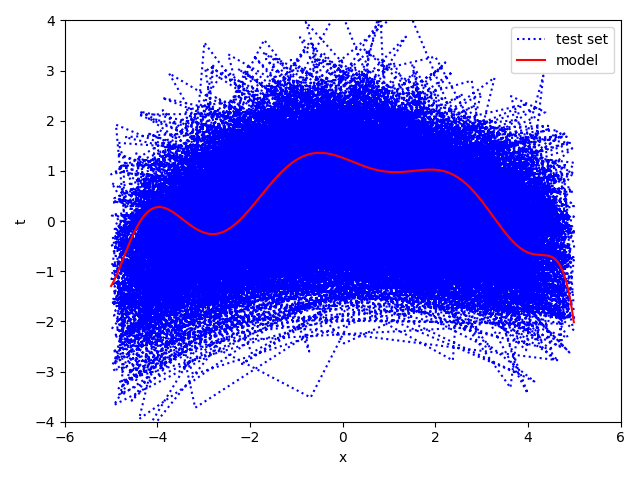
-9.48726052e-03 -3.59021142e-02 1.26988438e-03 1.85847109e-03

-3.58009594e-05 -3.17265738e-05]

* The optimal lambda is 2.8
* The minimum train RMSE is 0.9166735913611381
* The minimum test RMSE is 0.9974757376472851
* The loss in test set of my model is 5941.966776085015

Our plots are following:





## **LOOCV**

* Random shuffle the 100 data in the trainset.
* Discrete lambda from 0 to 5 with 0.1 interval.
* For each lambda, proceed the leave one out cross validation. Then I get one test error and one train error for each lambda.
* Find the lambda with minimum test error.
* Use that lambda along with the 100 data in trainset to compute the coefficients w. That’s my optimal 9-th order polynomial model.
* Use the optimal polynomial model to predict the data in testset, and compute the sum of the square error.

Our result shows that,

* The optimal w:

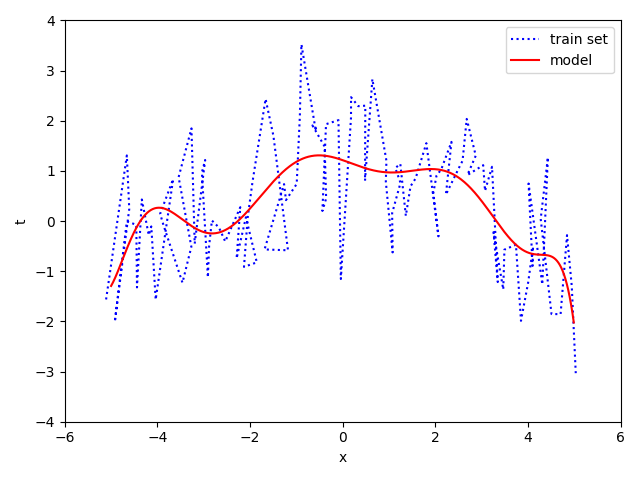
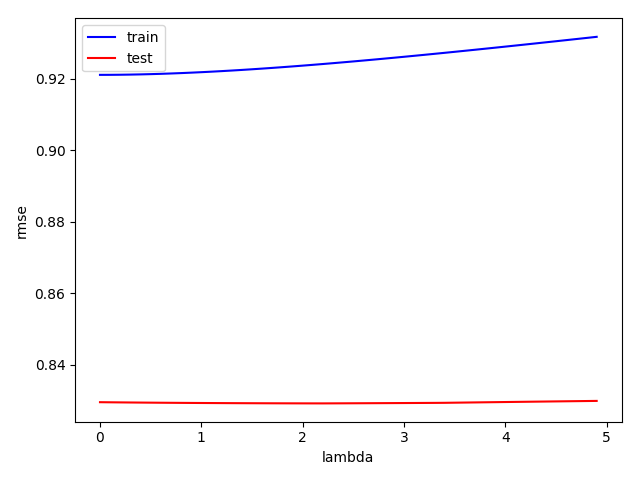
[ 1.21518339e+00 -3.18628809e-01 -1.33349631e-01 2.47243865e-01

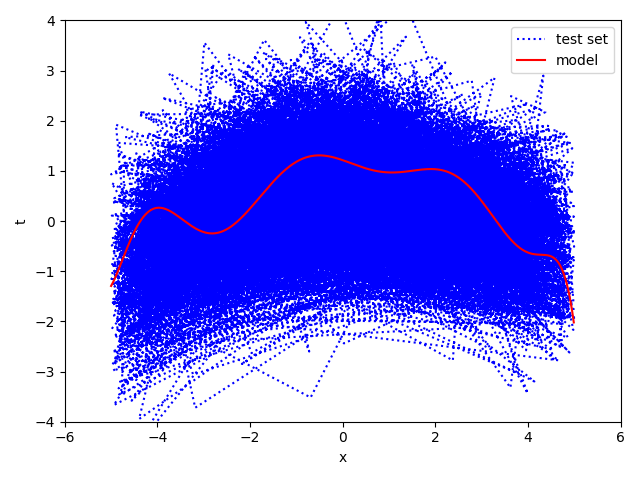
-6.27459951e-03 -3.67221444e-02 1.09492465e-03 1.89780423e-03

-3.26640365e-05 -3.23687829e-05]

* The optimal lambda is 2.2
* The minimum train RMSE is 0.9211138558950875
* The minimum test RMSE is 0.8291492656762018
* The loss in test set of my model is 5962.329077615319

The corresponding plots are following:





## **Appendix**

The code of the 10 fold CV:

* **Train.py**

import numpy as np  
import matplotlib.pyplot as plt  
  
out\_fn = 'w.npz'  
  
x, t = [], []  
  
with open("train.txt", "r") as train:  
 for line in train:  
 row = line.split(',')  
 x.append(float(row[0]))  
 t.append(float(row[1]))  
  
x = np.asarray(x)  
t = np.asarray(t)  
  
X = np.zeros((10, 100))  
  
train = np.zeros((10, 90))  
test = np.zeros((10, 10))  
  
  
train\_t = np.zeros(90)  
test\_t = np.zeros(10)  
  
index = np.arange(100)  
np.random.shuffle(index)  
index = index.reshape((10, 10))  
  
  
for i in range(0, 10):  
 X[i, :] = np.power(x, i)  
  
test\_err = []  
test\_err = np.asarray(test\_err)  
train\_err = []  
train\_err = np.asarray(train\_err)  
  
Eye = np.eye(10)  
  
  
def rmse(predictions, targets):  
 return np.sqrt(((predictions - targets) \*\* 2).mean())  
  
lamda = []  
lamda = np.asarray(lamda)  
lamda\_i = 0  
log\_lamda = []  
log\_lamda = np.asarray(log\_lamda)  
  
for step in range(0, 50):  
 lamda = np.append(lamda, 0.1\*step)  
  
 test\_err\_temp = []  
 test\_err\_temp = np.asarray(test\_err\_temp)  
 train\_err\_temp = []  
 train\_err\_temp = np.asarray(train\_err\_temp)  
  
 for k in range(0, 10):  
 test\_index = index[k]  
 train\_index = np.delete(index, k, 0).reshape(90)  
  
 ct = 0  
 for n in test\_index:  
 test[:, ct] = X[:, n]  
 test\_t[ct] = t[n]  
 ct = ct + 1  
  
 ct = 0  
 for m in train\_index:  
 train[:, ct] = X[:, m]  
 train\_t[ct] = t[m]  
 ct = ct + 1  
 w = np.dot(np.dot(np.linalg.inv((np.dot(train, train.T) + lamda[lamda\_i]\*Eye)), train), train\_t)  
  
 train\_err\_temp = np.append(train\_err\_temp, rmse(np.dot(train.T, w), train\_t))  
  
 test\_err\_temp = np.append(test\_err\_temp, rmse(np.dot(test.T, w), test\_t))  
  
 lamda\_i = lamda\_i + 1  
 test\_err = np.append(test\_err, test\_err\_temp.mean())  
 train\_err = np.append(train\_err, train\_err\_temp.mean())  
  
# print(test\_err)  
print(np.argmin(test\_err))  
print(np.amin(test\_err))  
# print(train\_err)  
print(np.argmin(train\_err))  
print(np.amin(train\_err))  
  
lamda\_min = 0.1\*np.argmin(test\_err)  
print(lamda\_min)  
  
w\_opt = np.dot(np.dot(np.linalg.inv((np.dot(X, X.T) + lamda\_min\*Eye)), X), t)  
print(w\_opt)  
  
plt.plot(lamda, train\_err, 'b', label='train')  
plt.plot(lamda, test\_err, 'r', label='test')  
  
plt.ylabel('rmse')  
plt.xlabel('lambda')  
plt.legend()  
plt.show()  
  
np.savez(out\_fn, w\_opt=w\_opt, lamda\_min=lamda\_min, test\_err\_min=np.amin(test\_err), train\_err\_min=np.amin(train\_err))  
  
  
exit()

* **Test.py**

import numpy as np  
import matplotlib.pyplot as plt  
  
  
def serr(predictions, targets):  
 return np.sum((predictions - targets) \*\* 2)  
  
  
x, t = [], []  
  
with open("test.txt", "r") as test:  
 for line in test:  
 row = line.split(',')  
 x.append(float(row[0]))  
 t.append(float(row[1]))  
  
x = np.asarray(x)  
t = np.asarray(t)  
  
parameters = np.load('w.npz')  
w = parameters['w\_opt']  
lamda = parameters['lamda\_min']  
test\_err\_min = parameters['test\_err\_min']  
train\_err\_min = parameters['train\_err\_min']  
  
print(w)  
print(lamda)  
print(test\_err\_min)  
print(train\_err\_min)  
  
X = np.zeros((10, len(x)))  
  
for i in range(0, 10):  
 X[i, :] = np.power(x, i)  
  
loss = serr(np.dot(X.T, w), t)  
  
x\_plt = []  
x\_plt = np.asarray(x\_plt)  
  
for i in range(0, 1000):  
 x\_plt = np.append(x\_plt, -5+i\*0.01)  
  
X\_plt = np.zeros((10, len(x\_plt)))  
  
for i in range(0, 10):  
 X\_plt[i, :] = np.power(x\_plt, i)  
  
plt.plot(x, t, 'b', label='test set', linestyle=':')  
plt.plot(x\_plt, np.dot(X\_plt.T, w), 'r', label='model')  
plt.xlabel('x')  
plt.ylabel('t')  
plt.ylim((-4, 4))  
plt.xlim((-6, 6))  
plt.legend()  
plt.show()  
  
print(loss)  
  
exit()