CPMS242 Machine Learning

Homework 1 Report

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

Date: Apr.10, 2018

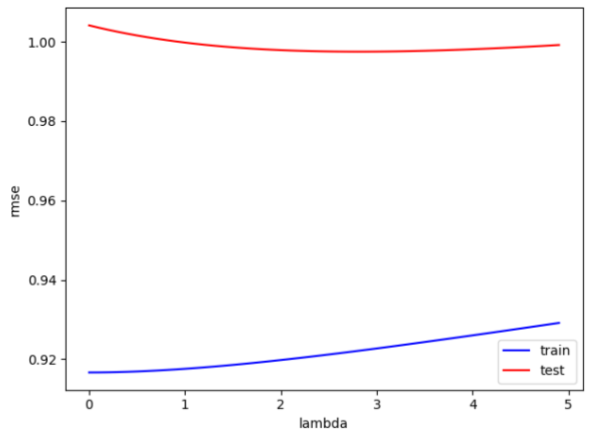
10_Fold Cross Validation

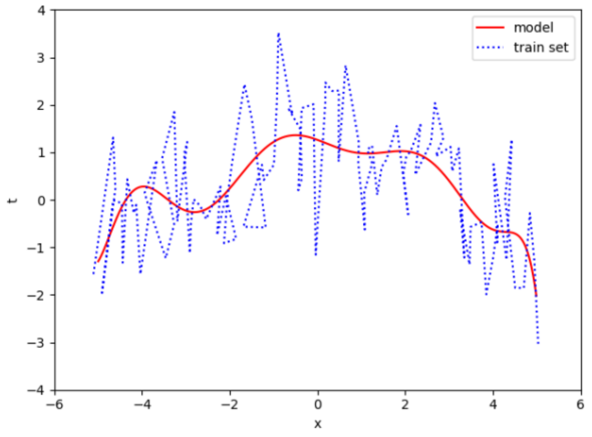
- Random shuffle the 100 data in the trainset, then divide it into 10 partitions equally.
- Discrete lambda from 0 to 5 with 0.1 interval.
- For each lambda, proceed the 10 fold cross validation. Then I get one test error and one train error for each lambda.
- Find the lambda with minimum test error.
- 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.

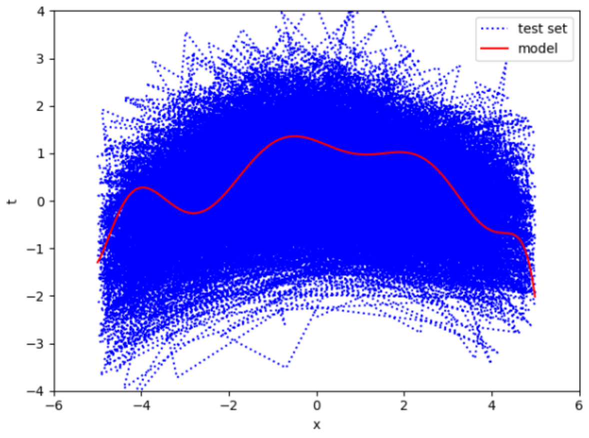
Our result shows that,

- 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,

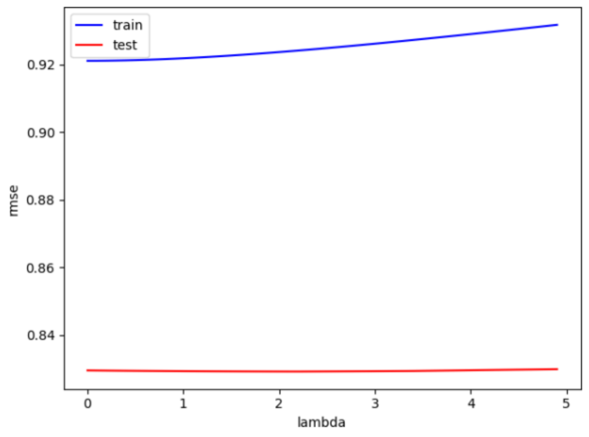
- The optimal w:

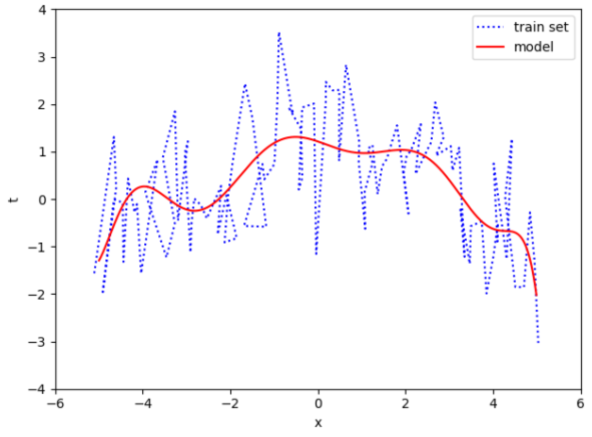
```
[ 1.21518339e+00 -3.18628809e-01 -1.33349631e-01  2.47243865e-01
 -6.27459951e-03 -3.67221444e-02  1.09492465e-03  1.89780423e-03
 -3.26640365e-05 -3.23687829e-05]
```

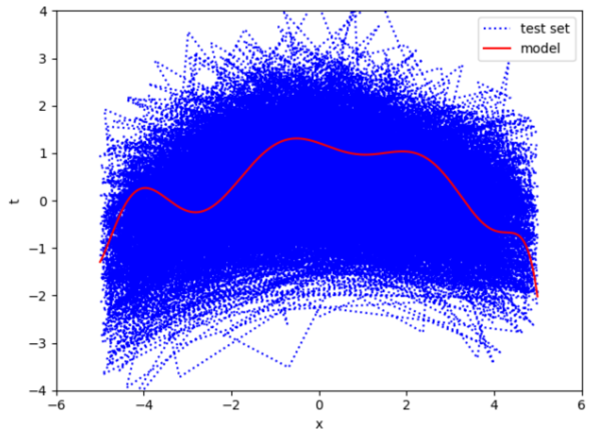
- The optimal lambda is 2.2

- 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 = [], []
```

```python
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()
```