COMS W4721 HW3

March 26, 2017

0.0.1 1. Gaussian process

a) The following code implement the Gaussian process method and make predictions on test data.

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import math
        import pandas as pd
        xtrain = np.loadtxt('X_train.csv',delimiter = ',', dtype = 'float')
        xtest = np.loadtxt('X_test.csv', delimiter = ',', dtype = 'float')
        ytrain = np.loadtxt('y_train.csv', delimiter = ',', dtype = 'float')
        ytest = np.loadtxt('y_test.csv', delimiter = ',', dtype = 'float')

In [3]: n = xtrain.shape[0]
        def Kernelmatrix(x, b):
            kernelmatrix = []
            for i in range(n):
                temp = []
                for j in range(n):
                    temp.append(math.exp(-(1.0/b) * np.dot(x[i] - x[j], x[i] - x[j])))
                kernelmatrix.append(temp)
            return np.array(kernelmatrix)

        def Kernelentity(x, d, b):
            k_xd = []
            for i in range(n):
                k_xd.append(math.exp(-(1.0/b) * np.dot(x - d[i], x - d[i])))
            return np.array(k_xd)

In [4]: #calculate
        n1 = xtest.shape[0]
        def predict(sigma, kernelmatrix, b):
            prediction = [0 for i in range(n1)]
            for i in range(n1):
                temp = np.linalg.inv(sigma * np.identity(n) + kernelmatrix)
                prediction[i] = np.dot(Kernelentity(xtest[i], xtrain, b), np.dot(temp, ytrain))
            return np.array(prediction)
```

```
#rmse table
b = [5,7,9,11,13,15]
sigma2 = [.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
rmse = 0
result = []
for i in range(len(b)):
    tempresult = []
    for j in range(len(sigma2)):
        kernelmatrix = Kernelmatrix(xtrain, b[i])
        #print predict(sigma2[j], kernelmatrix, b[i])
        error = predict(sigma2[j], kernelmatrix, b[i]) - ytest
        #print error
        rmse = math.sqrt(np.dot(error, error)/(1.0 * n1))
        tempresult.append(rmse)
    result.append(tempresult)
```

b). Below is the table of RMSE in different parameters

```
In [13]: table = {'5' : pd.Series(result[0], index=sigma2),'7' : pd.Series(result[1], index=sigm
         show = pd.DataFrame(table, columns=['5', '7', '9', '11', '13', '15'])
         show
```

Out[13]:

	5	7	9	11	13	15
0.1	1.966276	1.920163	1.897649	1.890507	1.895849	1.909603
0.2	1.933135	1.904877	1.902519	1.914981	1.935586	1.959549
0.3	1.923420	1.908080	1.917648	1.938849	1.964597	1.990804
0.4	1.922198	1.915902	1.932514	1.957936	1.985502	2.011915
0.5	1.924769	1.924804	1.945699	1.973216	2.001314	2.027370
0.6	1.929213	1.933701	1.957235	1.985764	2.013878	2.039465
0.7	1.934634	1.942254	1.967403	1.996375	2.024310	2.049463
0.8	1.940583	1.950380	1.976492	2.005603	2.033307	2.058105
0.9	1.946820	1.958093	1.984741	2.013835	2.041317	2.065845
1.0	1.953213	1.965438	1.992341	2.021345	2.048642	2.072976

c). The best value is when $b = 11$ and $\sigma^2 = 0.1$. The $RMSE = 1.890507$, so the result is better than the result in HW1 (when implement ridge regression with $\lambda = 23$ and $p = 2$, we get $RMSE = 2.19$).
However, the GP have some drawbacks. Normally, the GP method takes more time. Especially, when the dimension of variable is high, the complexity of the calculation on kernel matrix will be high too. When we try to find the best hyperparameters, we have to implement the GP method several times, the whole process will be much more time consuming.

d). The following is the visualization on GP.

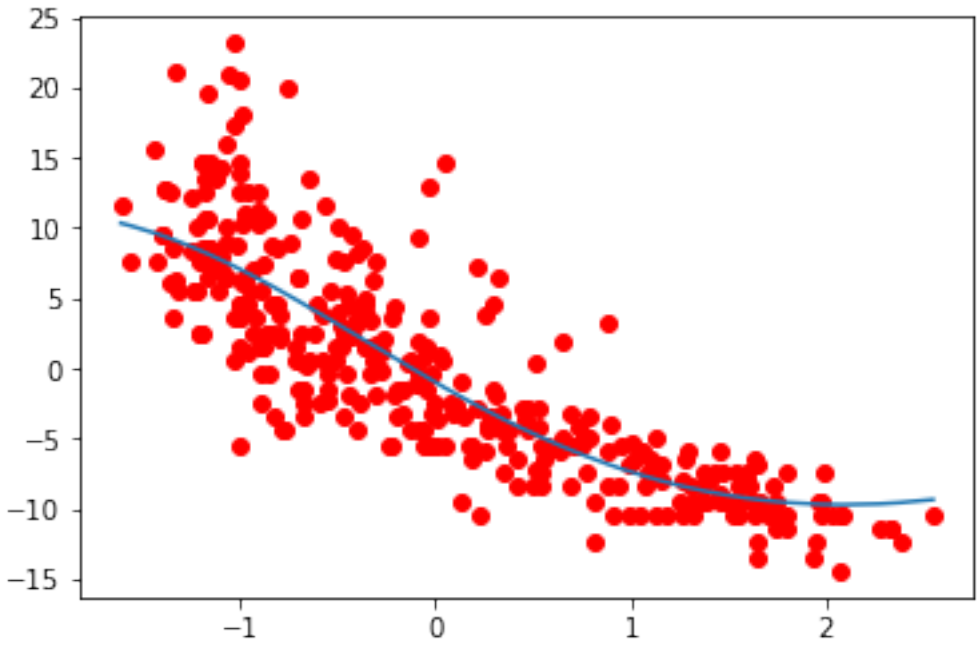
```
In [6]: # question d
        xd = np.transpose(xtrain)[3].reshape(350, 1)
        kernelmatrix_d = Kernelmatrix(xd, 5)
        def predict_d(sigma, kernelmatrix, b):
            prediction = [0 for i in range(n)]
```

```
            for i in range(n):
                temp = np.linalg.inv(sigma * np.identity(n) + kernelmatrix)
                prediction[i] = np.dot(Kernelentity(xd[i], xd, b), np.dot(temp, ytrain))
            return prediction
```

In [7]:
```
prediction_d = predict_d(2, kernelmatrix_d, 5)
```

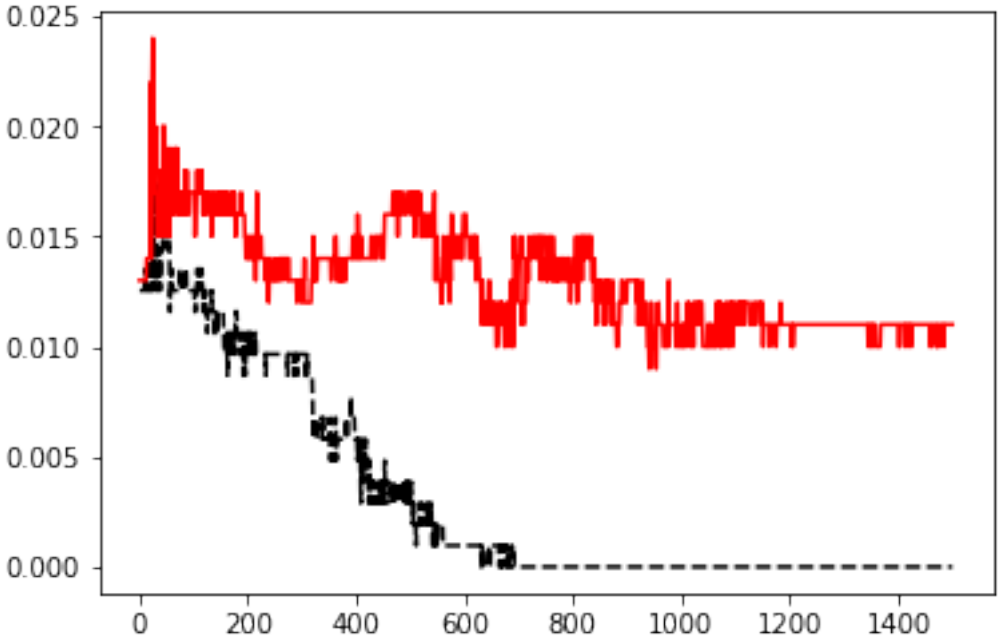
In [8]:
```
#question d
xy_plot = np.column_stack((xd, prediction_d))
xy_plot = xy_plot[xy_plot[:,0].argsort()]
```

In [9]:
```
fig, ax = plt.subplots()
ax.plot(xd, ytrain, 'ro')
ax.plot(xy_plot[:,0], xy_plot[:,1])
#ax.plot(x, y2, 'ro')
plt.show()
```

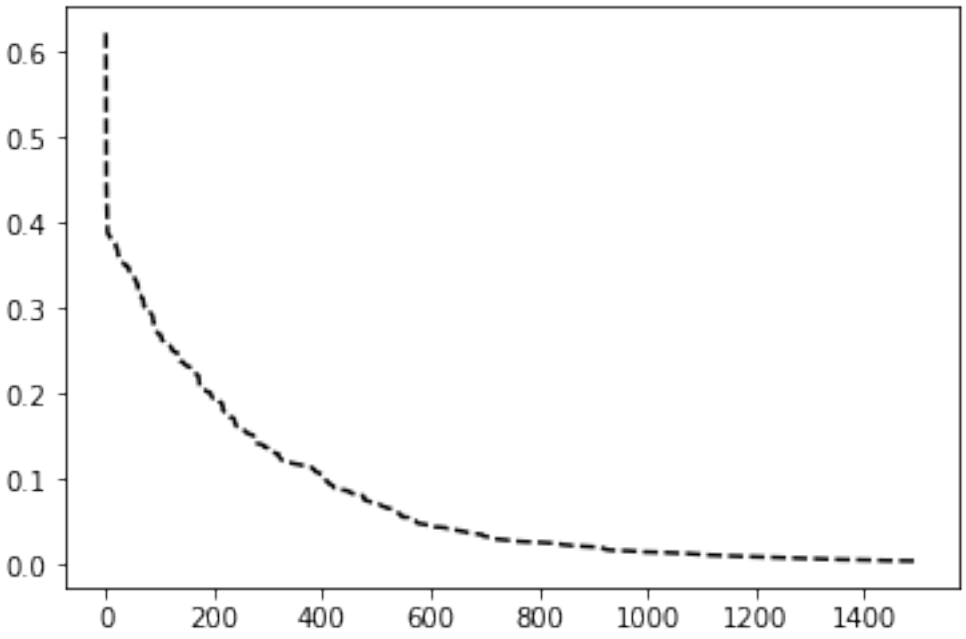


2. Boosting

a). The following graph show the training error and testing error of boosting for T = 1500.

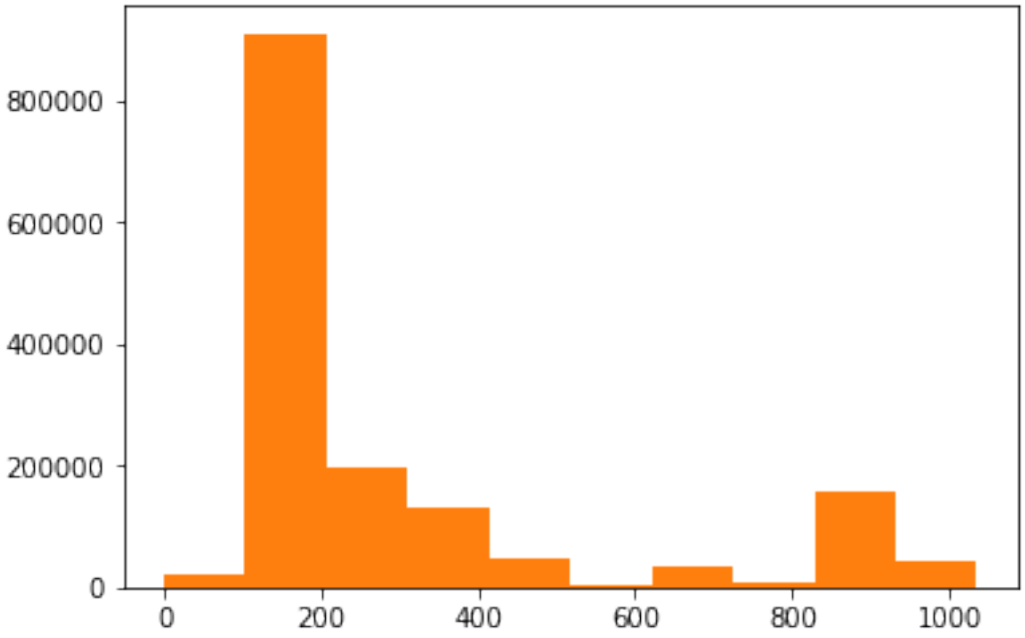


b). The following graph show the upper bound of training error.



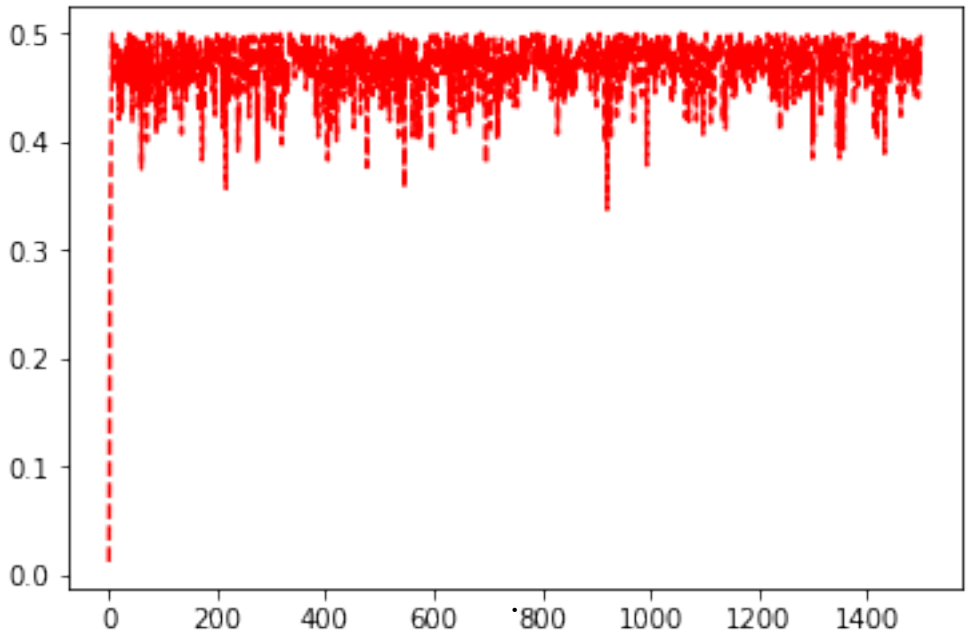
.

c). The histograms of bootstrap:

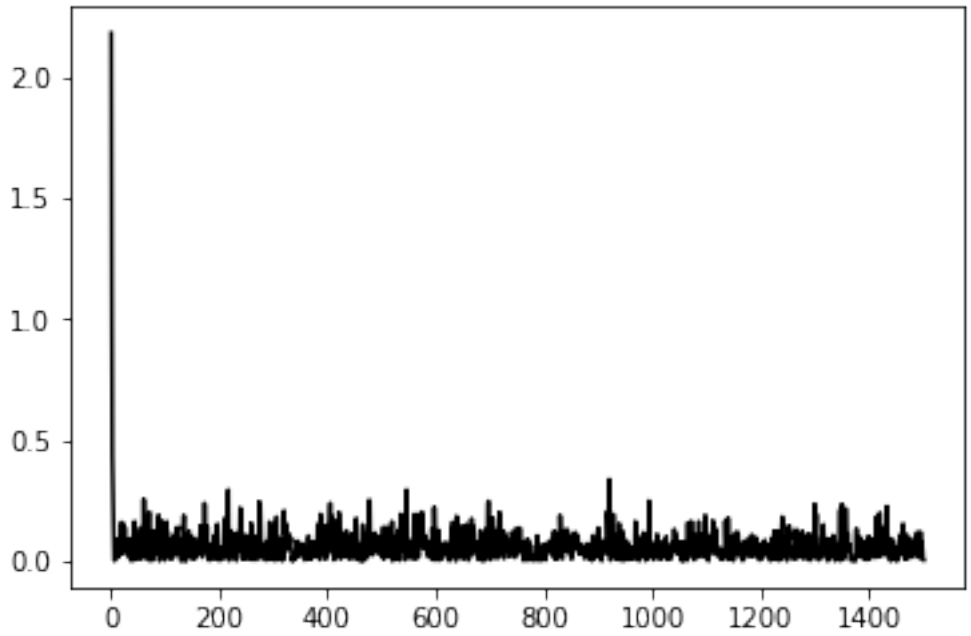


d).

ε_t as a function of t



α_t as a function of t



.