ai2335 - HW3-Copy1

March 26, 2017

```
In [1]: import numpy as np
        from numpy.linalg import inv
        import math
        import itertools
        import pandas as pd

        import matplotlib
        import matplotlib.pyplot as plt
        %matplotlib inline
```

0.1 1-A

```
In [3]: # kernel matrix
        def kernel(X_1, X_2, b):
            k = np.linalg.norm(X_1[None,:,:]-X_2[:,None,:],axis=2)
            return np.exp(-1/b * (k**2))


        # Gaussian process
        def G_process(X_train, y_train, X_test, b, sigma):
            K_n = kernel(X_train, X_train, b)

            I = np.identity(X_train.shape[0])
            c = np.linalg.inv((sigma) * I + K_n)

            K_k = kernel(X_test, X_train, b).T
            w = np.dot(K_k, c)
            predict = np.dot(w, y_train)

            return predict
```

0.2 1-B

```
In [5]: rmse = RMSE(y_predict, y_test, c)
        rmse_table = pd.DataFrame(
            {'parameters': c, 'rmse_value': rmse})
        rmse_table
```

Out[5]:

	parameters	rmse_value
0	(5, 0.1)	1.966276
1	(5, 0.2)	1.933135
2	(5, 0.3)	1.923420
3	(5, 0.4)	1.922198
4	(5, 0.5)	1.924769
5	(5, 0.6)	1.929213
6	(5, 0.7)	1.934634
7	(5, 0.8)	1.940583
8	(5, 0.9)	1.946820
9	(5, 1)	1.953213
10	(7, 0.1)	1.920163
11	(7, 0.2)	1.904877
12	(7, 0.3)	1.908080
13	(7, 0.4)	1.915902
14	(7, 0.5)	1.924804
15	(7, 0.6)	1.933701
16	(7, 0.7)	1.942254
17	(7, 0.8)	1.950380
18	(7, 0.9)	1.958093
19	(7, 1)	1.965438
20	(9, 0.1)	1.897649
21	(9, 0.2)	1.902519
22	(9, 0.3)	1.917648
23	(9, 0.4)	1.932514
24	(9, 0.5)	1.945699
25	(9, 0.6)	1.957235
26	(9, 0.7)	1.967403
27	(9, 0.8)	1.976492
28	(9, 0.9)	1.984741
29	(9, 1)	1.992341
30	(11, 0.1)	1.890507
31	(11, 0.2)	1.914981
32	(11, 0.3)	1.938849
33	(11, 0.4)	1.957936
34	(11, 0.5)	1.973216
35	(11, 0.6)	1.985764
36	(11, 0.7)	1.996375
37	(11, 0.8)	2.005603
38	(11, 0.9)	2.013835
39	(11, 1)	2.021345
40	(13, 0.1)	1.895849
41	(13, 0.2)	1.935586
42	(13, 0.3)	1.964597
43	(13, 0.4)	1.985502
44	(13, 0.5)	2.001314
45	(13, 0.6)	2.013878
46	(13, 0.7)	2.024310

```
47  (13, 0.8)    2.033307
48  (13, 0.9)    2.041317
49    (13, 1)    2.048642
50  (15, 0.1)    1.909603
51  (15, 0.2)    1.959549
52  (15, 0.3)    1.990804
53  (15, 0.4)    2.011915
54  (15, 0.5)    2.027370
55  (15, 0.6)    2.039465
56  (15, 0.7)    2.049463
57  (15, 0.8)    2.058105
58  (15, 0.9)    2.065845
59    (15, 1)    2.072976
```

0.3 1-C

```
In [6]: rmse_table.ix[rmse_table['rmse_value'].idxmin()]

Out[6]: parameters    (11, 0.1)
        rmse_value      1.89051
        Name: 30, dtype: object
```

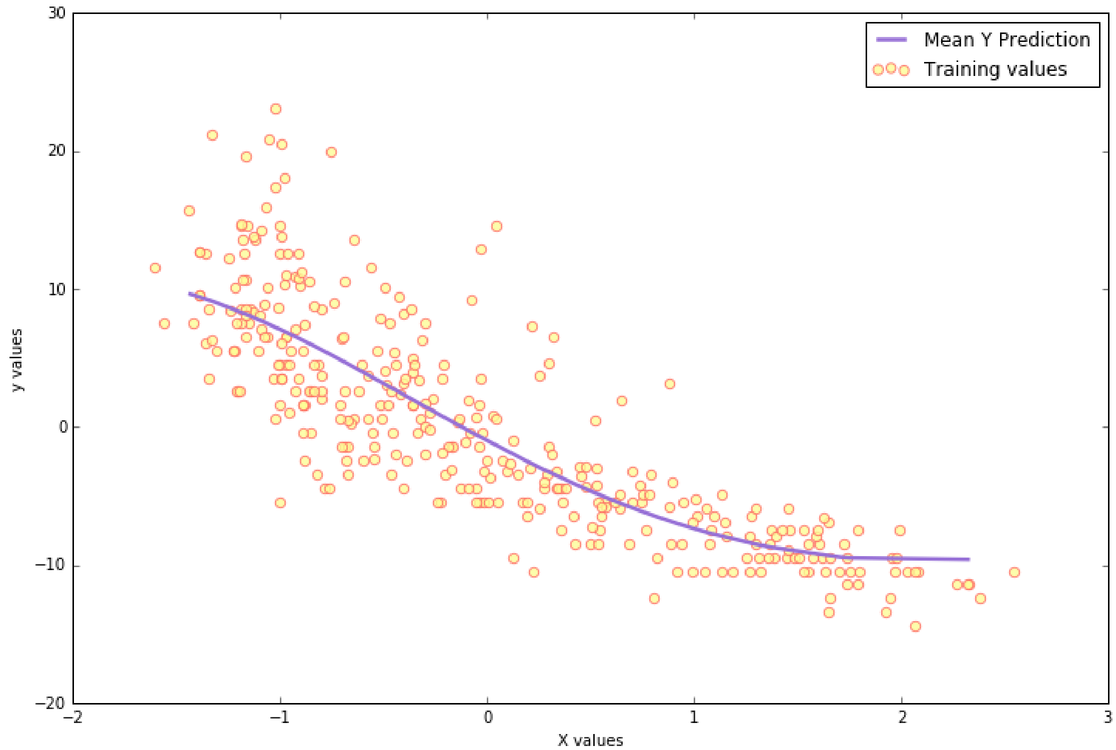
The best solution is for b= 11 and sigma = 0.1 with rmse value of 1.89051.

This approach comapring to homework 1 gives lower rmse, therefore we got a more accurate result using Gaussian Process. We can also have confidence intervals for predictions if we calculate covaraince.

However, Gaussian Procecss is computationally more expensive comparing to ridge and polynomial regression specially with large data. therefore there is an issue of scaling.

0.4 1-D

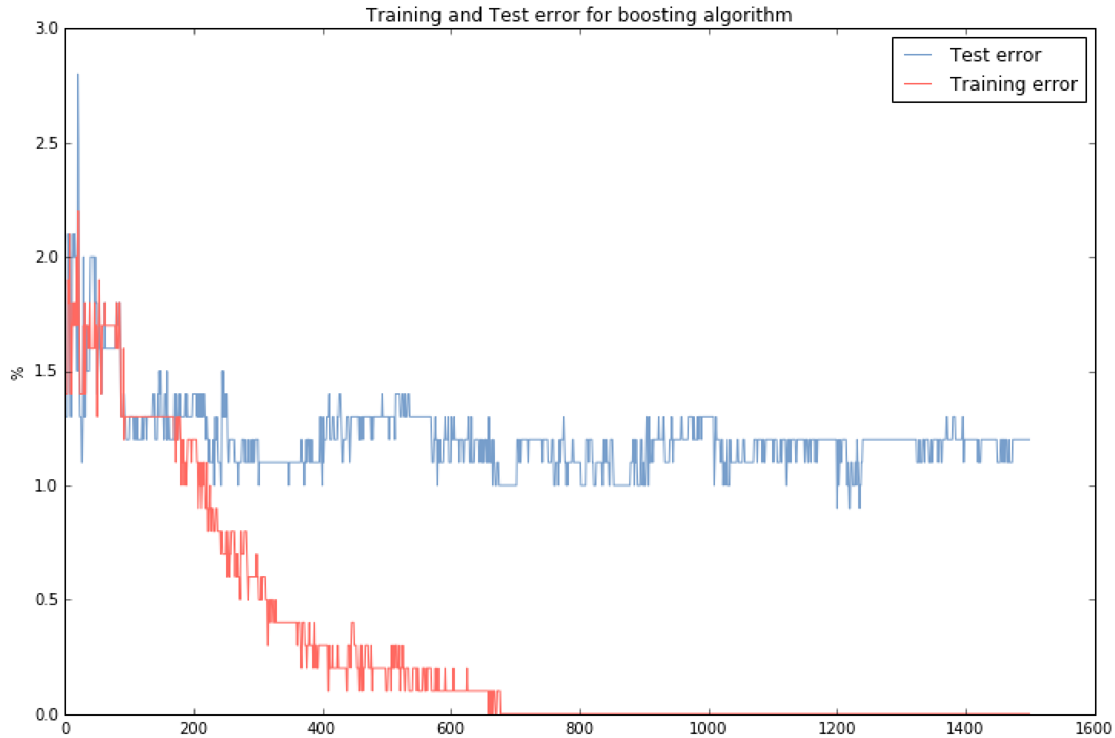
```
In [8]: plt.figure(figsize=(12, 8))
        plt.scatter(X_train_car_weight, y_train, alpha='0.8', facecolors='#fdfd96',
        plt.plot(df_4['x'], df_4['y'], '#966fd6', linewidth=2.5)

        plt.ylabel('y values')
        plt.xlabel('X values')

        labels = ['Mean Y Prediction', 'Training values']
        plt.legend(labels)

Out[8]: <matplotlib.legend.Legend at 0x11092fb00>
```



1 2

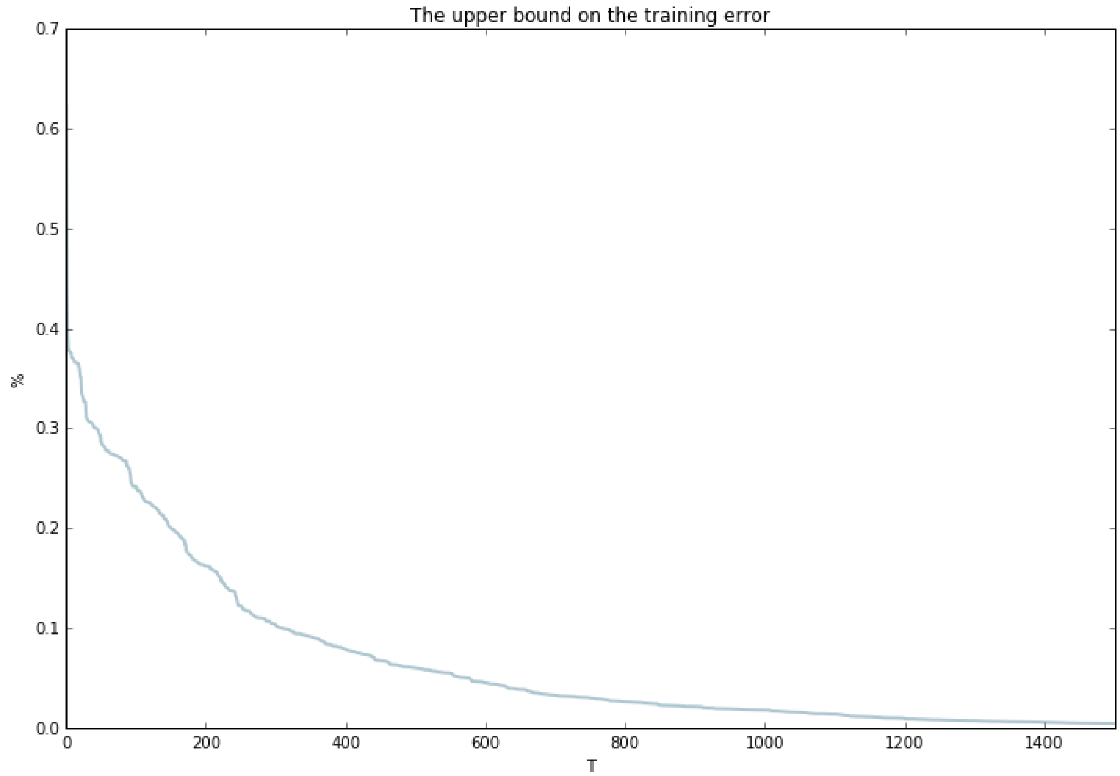
2 - A

```
In [12]: plt.figure(figsize=(12, 8))

         plt.plot(test_error, '#779ECB', train_error, '#FF6961')
         plt.title('Training and Test error for boosting algorithm')
         plt.ylabel('%')
         labels = ['Test error', 'Training error']
         plt.legend(labels)

Out[12]: <matplotlib.legend.Legend at 0x10318edd8>
```



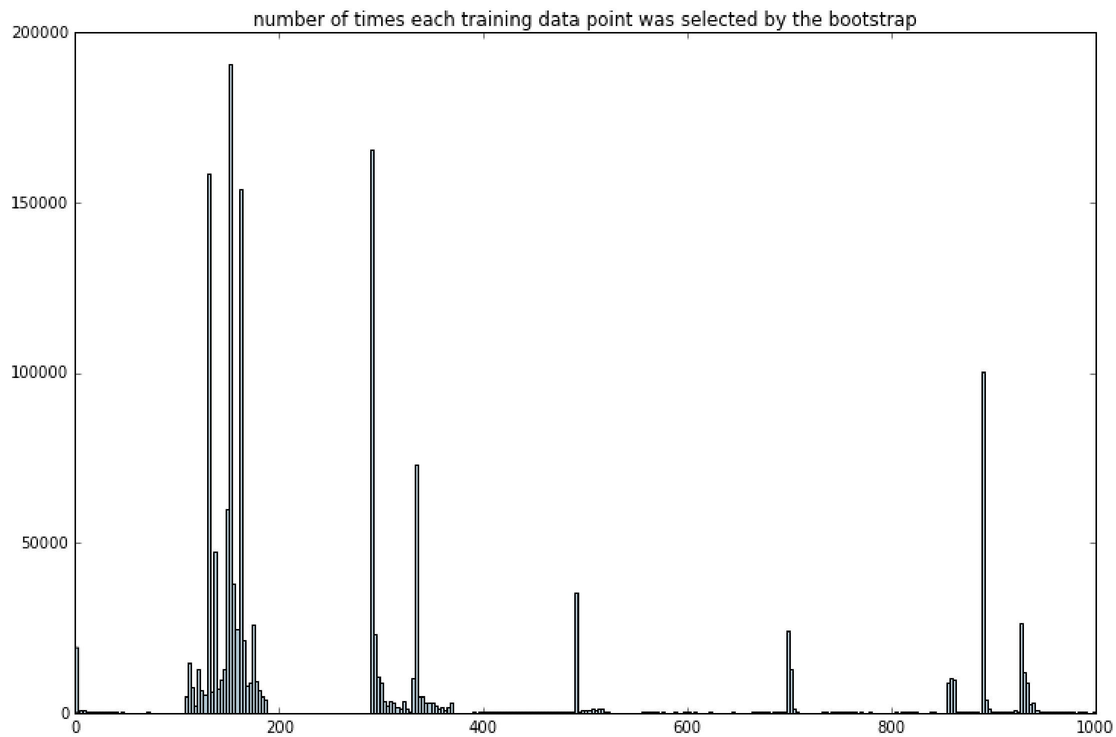
1.1 2-B

```
In [13]: epsilon_array = np.array(epsilon_list)
         epsilon = (0.5 - epsilon_array) ** 2

         ss = []

         for i in range(1, 1501):
             s = np.sum(epsilon[0:i])
             ss.append(s)
         ss2 = np.array(ss)
         ss2 = np.exp(-2 * ss2)
         plt.figure(figsize=(12, 8))

         plt.plot(ss2, '#AEC6CF', linewidth=2)
         plt.title('The upper bound on the training error')
         plt.ylabel('%')
         plt.xlabel('T')
         plt.xlim(-0.1, 1501)

Out[13]: (-0.1, 1501)
```



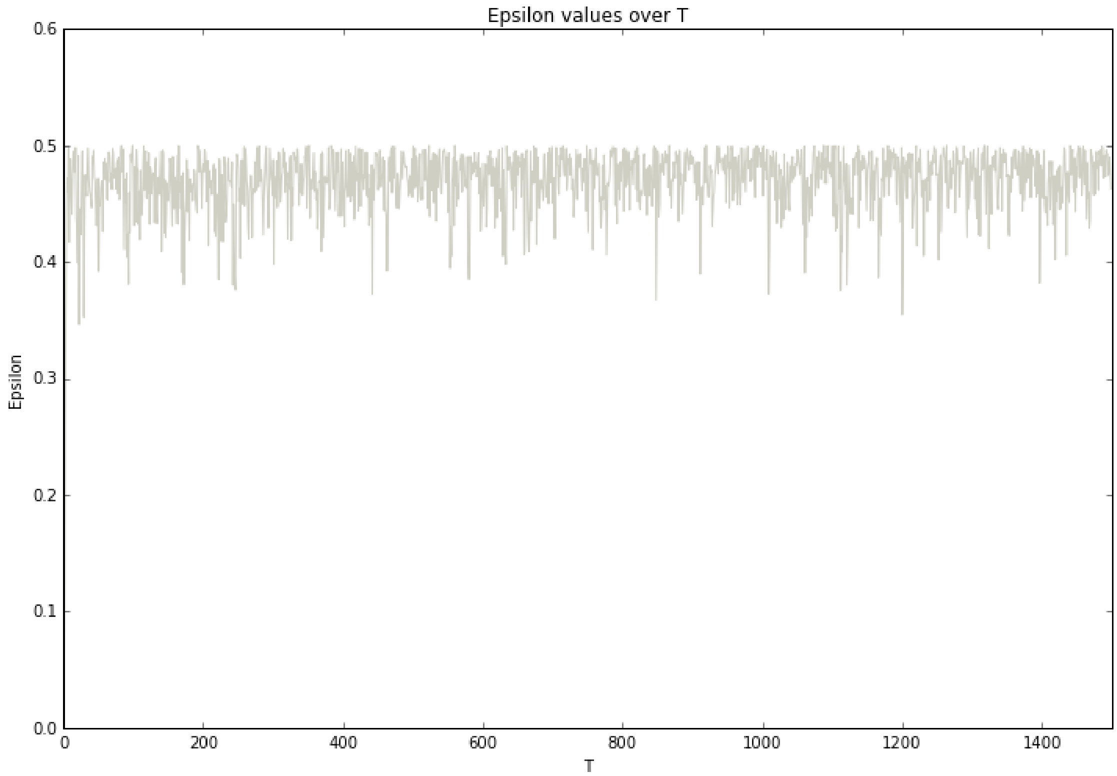
1.2 2-C

```
In [14]: flattened_B = [val for sublist in B_table for val in sublist]

         plt.figure(figsize=(12, 8))

         plt.hist(flattened_B, bins='auto', color='#AEC6CF')
         plt.xlim(-.05, 1000.05)
         plt.title('number of times each training data point was selected by the bo

Out[14]: <matplotlib.text.Text at 0x10c8412b0>
```

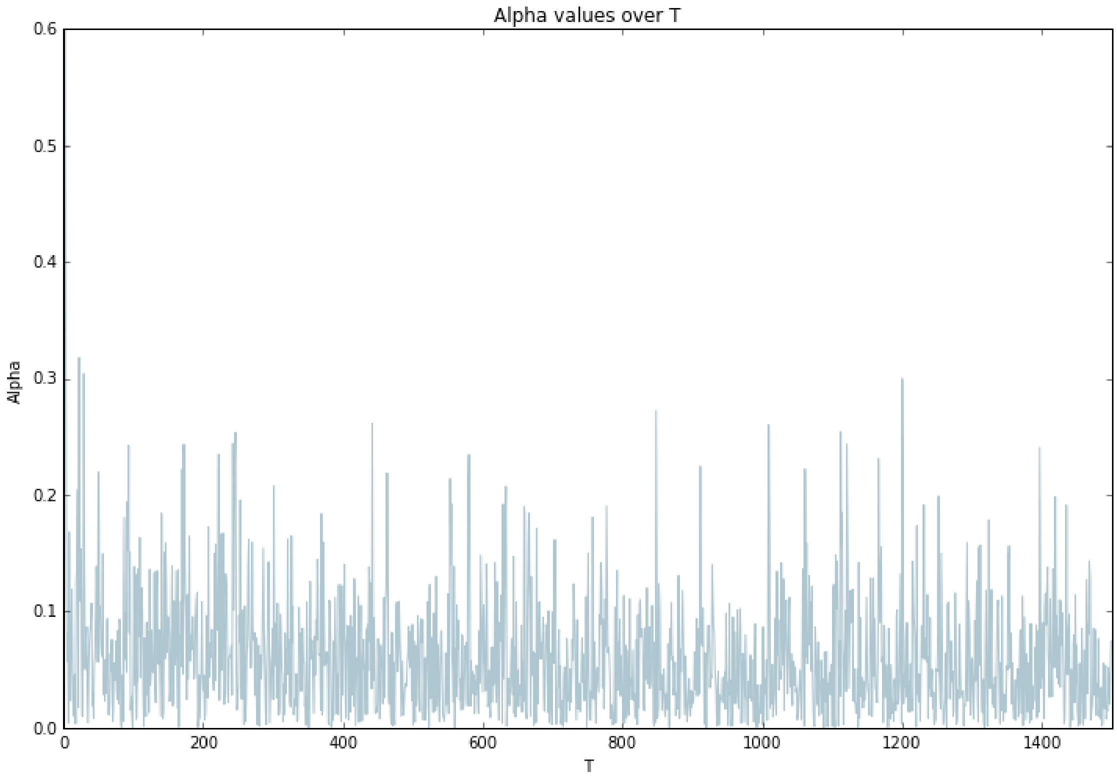


1.3 2-D

```
In [15]: plt.figure(figsize=(12, 8))

         plt.plot(epsilon_list, '#CFCFC4')
         plt.title('Epsilon values over T')
         plt.ylabel('Epsilon')
         plt.xlabel('T')
         plt.xlim(-0.05, 1500.05)
         plt.ylim(0, 0.6)

Out[15]: (0, 0.6)
```




```
In [16]: plt.figure(figsize=(12, 8))

         plt.plot(alpha_list, '#AEC6CF')
         plt.title('Alpha values over T')
         plt.ylabel('Alpha')
         plt.xlabel('T')
         plt.xlim(-0.05, 1500.05)
         plt.ylim(0, 0.6)

Out[16]: (0, 0.6)
```


Alpha values over T
Alpha
T
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000
1200
1400