

Exercise 5.7

a)

Code:

```
1. # YOUR CODE GOES HERE takes poly features of the input
2. def poly_features(x,D):
3.     F = np.ones((len(x),D+1))
4.     for i in range(len(x)):
5.         for j in range(D+1):
6.             F[i,j] = x[i]**j
7.
8.     F = F.T
9.
10.    return F
```

Figure:

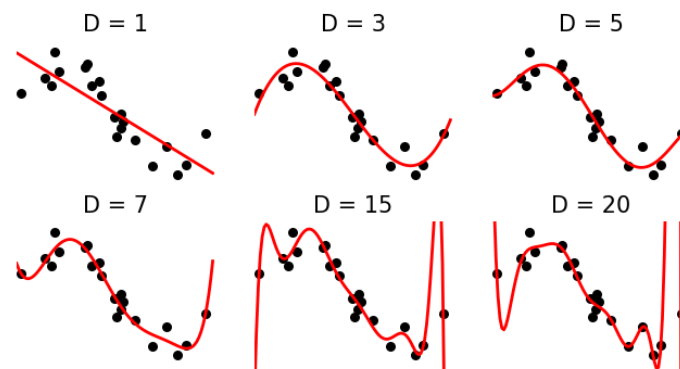


Figure1

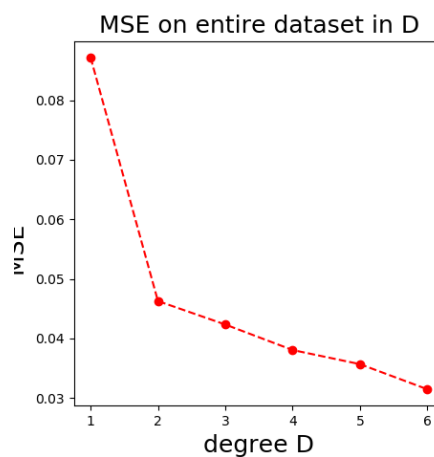


Figure2

b)

In the first figure, with the increasing of D, the indicator function fit the data better.

In the second figure, with the increasing of D, the mean squared error (MSE) decrease, which means the fitness becomes better and more stable.

Exercise 5.10

- 1) For Eric's plot, I recommend 5 as the degree of polynomial, since at that degree, the testing error is at the least level and the training error is small.
- 2) For Stanley's plot, I recommend choosing 8 as the degree of polynomial, since at that degree, the testing error is at the least level and the training error is small.
- 3) For Kyle's plot, I recommend keeping increasing the number of D , since the errors keep decreasing, there may be some other degree which fits the datasets better.
- 4) For Kenneth's plot, I recommend choosing 6 as the degree of polynomial, since at that degree, the testing error is at the least level and the training error is small.

Exercise 5.11

Code:

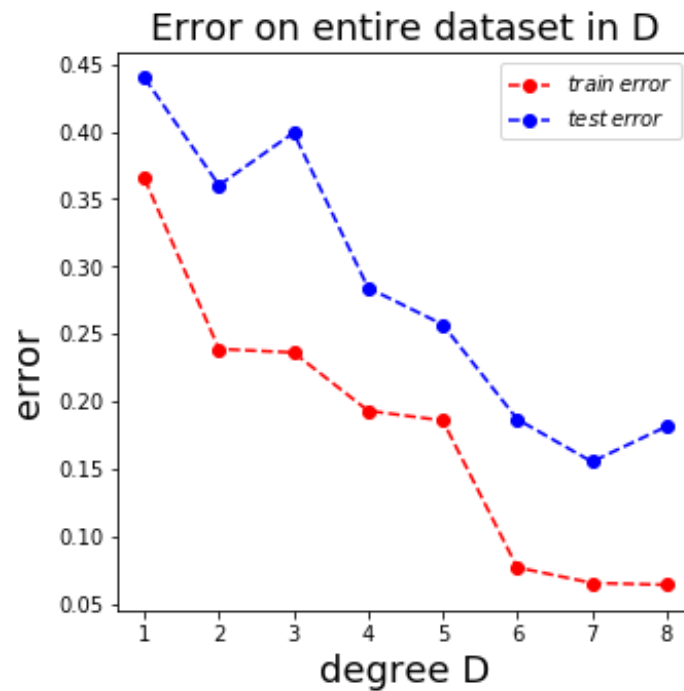
```
1. from __future__ import division
2. import numpy as np
3. import numpy.matlib
4. import matplotlib.pyplot as plt
5. import pylab
6. from sklearn.model_selection import train_test_split
7.
8. # load data
9. def load_data():
10.     data = np.array(np.genfromtxt('C:/Users/10448/Desktop/wavy_data.csv', de
11.     limiter=','))
12.     x = np.reshape(data[:,0],(np.size(data[:,0]),1))
13.     y = np.reshape(data[:,1],(np.size(data[:,1]),1))
14.     return x,y
15.
16. # Fourier features
17. def four_features(x,D):
18.     F = np.zeros((len(x),D+1))
19.     for i in range(len(x)):
20.         for j in range(D+1):
21.             if j%2 ==0:
22.                 F[i,j] = np.cos(2*np.pi*((j+2)/2)*x[i])
23.             else:
24.                 F[i,j] = np.sin(2*np.pi*((j+1)/2)*x[i])
25.     F = F.T
26.     return F
27.
28. # plot train error and test error over all D tested
29. def plot_error(train,test,deg):
30.     plt.plot(np.arange(1,np.size(train)+1),train,'ro--')
31.     plt.plot(np.arange(1,np.size(test)+1),test,'bo--')
32.     plt.title('Error on entire dataset in D', fontsize=18)
33.     plt.xlabel('degree D', fontsize=18)
34.     plt.ylabel('error', fontsize=18)
35.     plt.legend([r'$train$error$',r'$test$error$'])
36.
37. # run over all the degrees and calculate errors
38. def try_all_degs(x,y,deg_range):
39.
40.     # split data
41.     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=1/3,
42.     random_state=19)
43.
44.     # generate train and test error features
45.     train_error = []
46.     test_error = []
47.
48.     # calculate errors
49.     for D in np.arange(0,np.size(deg_range)):
50.         # generate fourier feature transformation
51.         F_train = four_features(x_train,deg_range[D])
52.         F_test = four_features(x_test,deg_range[D])
53.
54.         # concatenate ones for F
55.         tr_o = np.ones((np.shape(F_train)[1],1))
56.         F_train_new = np.concatenate((tr_o,F_train.T),axis = 1)
57.         F_train_new = F_train_new.T
58.         te_o = np.ones((np.shape(F_test)[1],1))
59.         F_test_new = np.concatenate((te_o,F_test.T),axis = 1)
60.         F_test_new = F_test_new.T
```

```

60.
61.     # get error
62.     temp_train = np.linalg.pinv(np.dot(F_train_new,F_train_new.T))
63.     w_train = np.dot(np.dot(temp_train,F_train_new),y_train)
64.     tr_error = np.linalg.norm(np.dot(F_train_new.T,w_train)-
y_train)/np.size(y_train)
65.     temp_test = np.linalg.pinv(np.dot(F_test_new,F_test_new.T))
66.     w_test = w_train
67.     te_error = np.linalg.norm(np.dot(F_test_new.T,w_test)-
y_test)/np.size(y_test)
68.     train_error.append(tr_error)
69.     test_error.append(te_error)
70.
71.     # make plot of train and test errors
72.     fig = plt.figure(figsize = (5,5))
73.     plot_error(train_error,test_error,deg_range)
74.     plt.show()
75.
76. # load data and defined degree range
77. x, y = load_data()
78. deg_range = [1,2,3,4,5,6,7,8]           # degree fourier to try
79.
80. # run all over degree range
81. try_all_degs(x,y,deg_range)

```

Figure:



Exercise 5.12

Code:

```
1. from __future__ import division
2. import numpy as np
3. import numpy.matlib
4. import matplotlib.pyplot as plt
5. import pylab
6. from sklearn.model_selection import train_test_split
7. from sklearn.model_selection import KFold
8.
9. # load data
10. def load_data():
11.     data = np.array(np.genfromtxt('C:/Users/10448/Desktop/galileo_ramp_data.
12.     csv', delimiter=','))
13.     x = np.reshape(data[:,0],(np.size(data[:,0]),1))
14.     y = np.reshape(data[:,1],(np.size(data[:,1]),1))
15.     return x,y
16.
17. # Fourier features
18. def poly_features(x,D):
19.     F = np.ones((len(x),D+1))
20.     for i in range(len(x)):
21.         for j in range(D+1):
22.             F[i,j] = x[i]**j
23.     F = F.T
24.     return F
25.
26. # plot train error and test error over all D tested
27. def plot_error(train,test,deg):
28.     plt.plot(np.arange(1,np.size(train)+1),train,'ro--')
29.     plt.plot(np.arange(1,np.size(test)+1),test,'bo--')
30.     plt.title('Error on entire dataset in D', fontsize=18)
31.     plt.xlabel('degree D', fontsize=18)
32.     plt.ylabel('error', fontsize=18)
33.     plt.legend([r'$train$error$',r'$test$error$'])
34.
35. # run over all the degrees and calculate errors
36. def try_all_degs(x,y,deg_range):
37.
38.     # split data
39.     kf = KFold(n_splits = 6,shuffle = False)
40.     x_train = []
41.     x_test = []
42.     y_train = []
43.     y_test = []
44.     for train_index, test_index in kf.split(x):
45.         x_train.append(x[train_index])
46.         x_test.append(x[test_index])
47.         y_train.append(y[train_index])
48.         y_test.append(y[test_index])
49.
50.     # generate train and test error features
51.     train_error = []
52.     test_error = []
53.     train_error_ave = 0
54.     test_error_ave = 0
55.
56.     # calculate errors
57.     for D in np.arange(0,np.size(deg_range)):
58.         # generate poly feature transformation
59.         for i in range(np.size(x_train,0)):
60.             F_train = poly_features(x_train[i],deg_range[D])
```

```

61.         F_test = poly_features(x_test[i],deg_range[D])
62.         # get error
63.         temp_train = np.linalg.pinv(np.dot(F_train,F_train.T))
64.         w_train = np.dot(np.dot(temp_train,F_train),y_train[i])
65.         tr_error = np.linalg.norm(np.dot(F_train.T,w_train)-
y_train[i])/np.size(y_train[i])
66.         temp_test = np.linalg.pinv(np.dot(F_test,F_test.T))
67.         w_test = w_train
68.         te_error = np.linalg.norm(np.dot(F_test.T,w_test)-
y_test[i])/np.size(y_test[i])
69.         train_error_ave = train_error_ave + tr_error/(np.size(x_train,0)
)
70.         test_error_ave = test_error_ave + te_error/(np.size(x_train,0))

71.
72.     train_error.append(tr_error)
73.     test_error.append(te_error)
74.     train_error_ave = 0
75.     test_error_ave = 0
76.
77.     # make plot of train and test errors
78.     fig = plt.figure(figsize = (5,5))
79.     plot_error(train_error,test_error,deg_range)
80.     plt.show()
81.
82. # load data and defined degree range
83. x, y = load_data()
84. deg_range = [1,2,3,4,5,6]           # degree polynomial to try
85.
86. # run all over degree range
87. try_all_degs(x,y,deg_range)

```

Figure:

