EECS495

HW5

Liu Cao 3072379

Excercise5.7 Polynomial basis feature regression for scalar valued input

a) The code I add is:

```
# YOUR CODE GOES HERE takes poly features of the input
def poly_features(x,D):
    F = []
    for i in x:
        F.append(1)
        for j in range(1,D+1):
            F.append(i**j)
    F=np.array(F)
    F.shape=(len(x),D+1)
    F=F.T
    return Freturn F
```

b) Two figures show below.

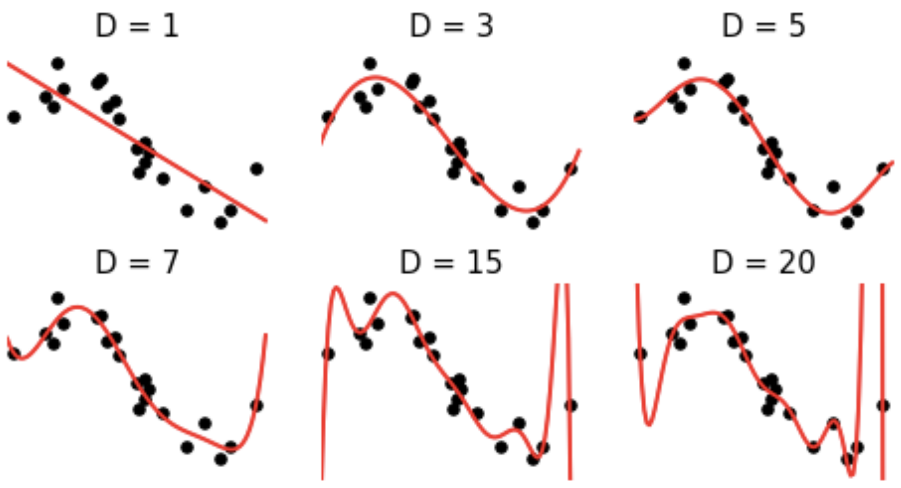

Fig.1 Data along with various degree D polynomial fits

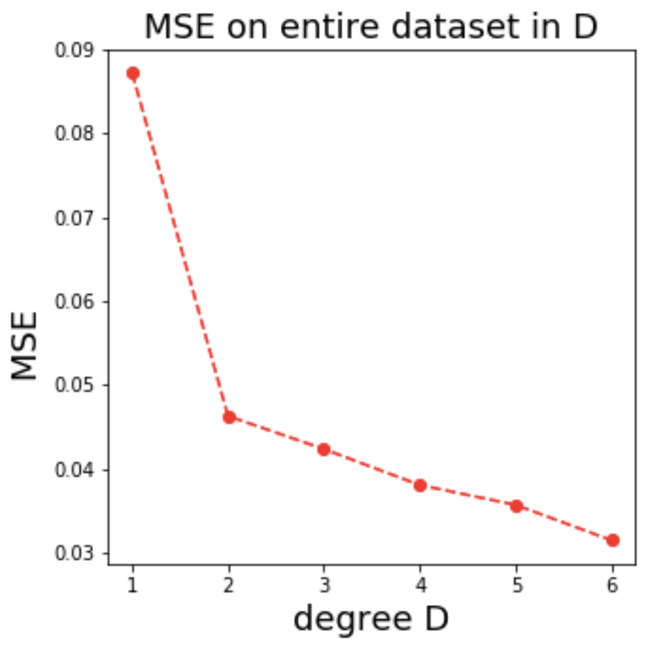

Fig.2 MSE

Description: According to the results in Fig.1 and Fig.2, when D goes up, the MSE will decrease, and the curve will more fit the observed data(function). However, if D is set too large, the curve will change, which no more fits the observed data(function) than before. In this case, the curve becomes over-fitted. Considering above, we need to set the D carefully. We should not only care the MSE is small enough to avoid underfitting the data but also pay attention to the over-fitted curve. In this problem, according to the results in Fig.1 and Fig.2, the best value of D could be around 5.

Exercises5.10 Four guys and four error plots

Eric: According to the Eric's plot, D=5 provides the lowest testing error or the best fit. Then he should use D to train on the entire dataset and observe how well the model fits the data.

Stanley: According to the Stanley's plot, D=3 and D=8 provide the best fit. There are two ways to select. Since the training error is lower when D=8, he can use D=8 to train on the entire dataset and observe how well the model fits the data. However, in this case he may observe the overfitting happens. Thus, a more reliable way which may take a bit longer is k-fold cross-validation. It could avoid the underfitting and overfitting. After obtaining a new D which provide the lowest testing error, he should use D train on the entire dataset and observes how well the model fits the data.

Kyle: According to the Kyle's plot, the plot doesn't show the D which provides the lowest testing error, i.e., the minimum testing error point doesn't show up. He should increase the range of D tried until he can find the minimum testing error point. In that way, he can get the corresponding D and use D train on the entire dataset, observing how well the model fits the data

Kenneth: According to the Kenneth's plot, D=6 provides the lowest testing error or the best fit. Then he should use D to train on the entire dataset and observe how well the model fits the data.

Excercises5.11 Practice the hold out method

Start by randomly splitting the dataset into k = 3 equal sized folds (keeping two folds as training, and one fold as testing data). Use the Fourier basis features M in the range M=2, 4, 6,…,16, we can get the graph showing the training and testing error for each M.

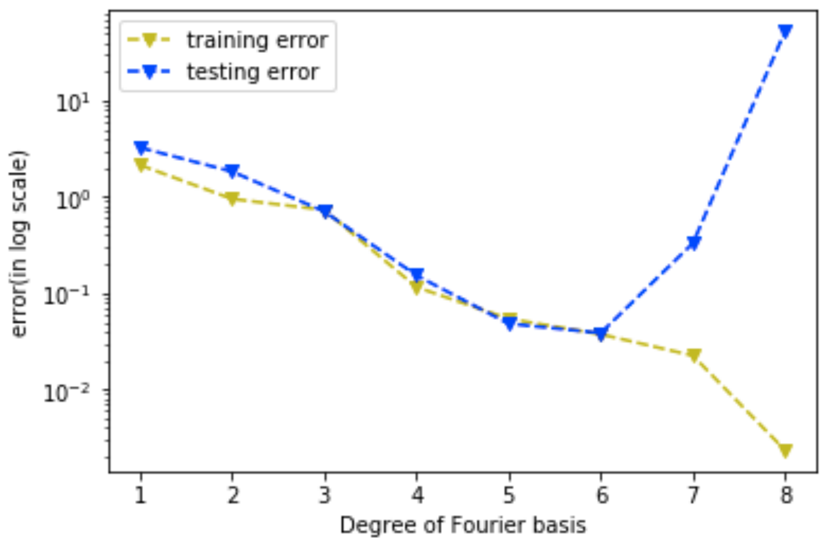

Fig.3 Degree of Fourier basis

According to Fig.3, we can see the best degree of Fourier basis is 12. Then M=12 is used to train on the entire dataset so that we can get the best(i.e., the lowest test error) model(Fig.4).

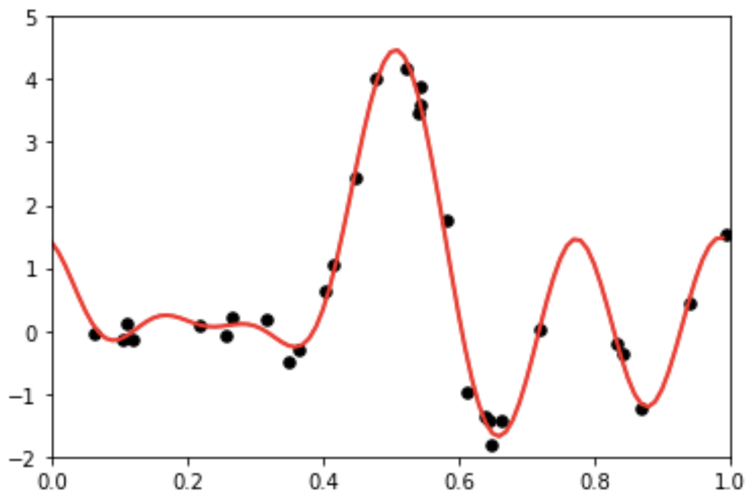

Fig.4 The lowest test error model

Complete code:

```
from __future__ import division

import numpy as np

import numpy.matlib

import matplotlib.pyplot as plt

import pylab

import math


def read_data(filename):

    data = np.array(np.genfromtxt(filename, delimiter=','))

    X = np.reshape(data[:, 0], (1,np.size(data[:, 0])))

    y = np.reshape(data[:, 1], (np.size(data[:, 1]), 1))

    return X, y


def fourier_basis(X, D):

    F = np.zeros((2 * D, len(X[0])))

    one = np.ones(len(X[0]))

    one = np.reshape(one, (1, len(X[0])))

    for i in range(0, D):

        for j in range(0, len(X[0])):

            F[2 * i][j] = np.cos(2 * np.pi * (i + 1) * X[0][j])

            F[2 * i + 1][j] = np.sin(2 * np.pi * (i + 1) * X[0][j])
    F = np.concatenate((one, F), axis=0)
    return F


def least_square_sol(F, y):
    '''
    Refer to eq. 5.19 in the text
    '''
    w = np.dot(np.dot(np.linalg.pinv(np.dot(F, F.T)), F), y)
    return w


def mean_square_error(w, F, y):
    res = np.dot(F.T, w) - y
    res = res * res
    mse = np.mean(res, axis=0)
    return mse


def random_split(P, K):
    temp = np.random.permutation(P)
    folds = np.split(temp, K)
    return folds


def train_val_split(X, y, folds, fold_id):


    X_val = np.zeros((1, len(folds[fold_id])))
    y_val = np.zeros((len(folds[fold_id]), 1))
    X_train = np.zeros((1, (len(folds) - 1) * len(folds[fold_id])))
    y_train = np.zeros((((len(folds) - 1) * len(folds[fold_id]), 1))
    j = 0
    for i in folds[fold_id]:
        X_val[0][j] = X[0][i]
        y_val[j][0] = y[i][0]
        j = j + 1
    count = 0
    for k in range(0, len(folds)):
        if k != fold_id:
            for l in folds[k]:
                X_train[0][count] = X[0][l]
                y_train[count][0] = y[l][0]
                count = count + 1
    return X_train, y_train, X_val, y_val


def make_plot(D, MSE_train, MSE_val):
    plt.figure()
    train, = plt.plot(D, MSE_train, 'yv--')
    val, = plt.plot(D, MSE_val, 'bv--')
    plt.legend(handles=[train, val], labels=['training error', 'testing error'])
    plt.xlabel('Degree of Fourier basis')
    plt.ylabel('error(in log scale)')
    plt.yscale('log')
    plt.show()


def load_data():
    data = np.array(np.genfromtxt('wavy_data.csv', delimiter=','))
    x = np.reshape(data[:,0],(np.size(data[:,0]),1))
    y = np.reshape(data[:,1],(np.size(data[:,1]),1))
    return x,y


# YOUR CODE GOES HERE takes poly features of the input
def fourier_features(x,D):
    F = []
    for i in x:
        F.append(1)
        for j in range(1,D+1):
            F.append(math.cos(2*math.pi*j*i))
            F.append(math.sin(2*math.pi*j*i))
    F=np.array(F)
    F.shape=(len(x),2*D+1)
    F=F.T
    return F

# plot the polynomial
def plot_model(w,D):
    # plot determined surface in 3d space
    s = np.arange(0,1,.01)
    f = fourier_features(s,D)
    z = np.dot(f.T,w)

    # plot contour in original space
    plt.plot(s,z, color = 'r', linewidth = 2)
    plt.ylim([-2,5])
    plt.xlim([0,1])

# plot data
def plot_data(x,y,deg):

        plt.scatter(x,y,s = 30, color = 'k')

# run over all the degrees, fit each models, and calculate errors
def try_all_degs(x,y,deg_range):
    # plot datapoints - one panel for each deg in deg_range
    plot_data(x,y,deg_range)

    # generate nonlinear features
    mses = []

    for D in np.arange(0,np.size(deg_range)):
        # generate poly feature transformation
        F = fourier_features(x,deg_range[D])
        # get weights for current model
        temp = np.linalg.pinv(np.dot(F,F.T))
        w = np.dot(np.dot(temp,F),y)
        MSE = np.linalg.norm(np.dot(F.T,w)-y)/np.size(y)
        mses.append(MSE)

        # plot fit to data
        plot_model(w,deg_range[D])

X, y = read_data('wavy_data.csv')
num_fold, num_degree = 3, 8
folds = random_split(P=y.size, K=num_fold)

X_train, y_train, X_val, y_val = train_val_split(X, y, folds, fold_id=0)

MSE_train, MSE_val = [], []
D = np.arange(1, num_degree+1)
for d in D:
    F_train = fourier_basis(X_train, D=d)
    F_val = fourier_basis(X_val, D=d)
    w = least_square_sol(F_train, y_train)
    MSE_train.append(mean_square_error(w, F_train, y_train))
    MSE_val.append(mean_square_error(w, F_val, y_val))

print ('The best degree of Fourier basis:%d' % (MSE_val.index(min(MSE_val))+1))
make_plot(D, MSE_train, MSE_val)

# load data and defined degree range
x, y = load_data()
deg_range =[(MSE_val.index(min(MSE_val))+1)]                # degree polys to try
# run all over degree range
try_all_degs(x,y,deg_range)
```

Excercises5.12 Code up k-fold cross-validation

Start by randomly splitting the dataset into k = 6 equal sized folds (keeping five folds as training, and one fold as testing data during each round of cross-validation). Use the polynomial basis features M in the range M=1,2,…,6, we can get the graph showing the average training and testing error for each M.

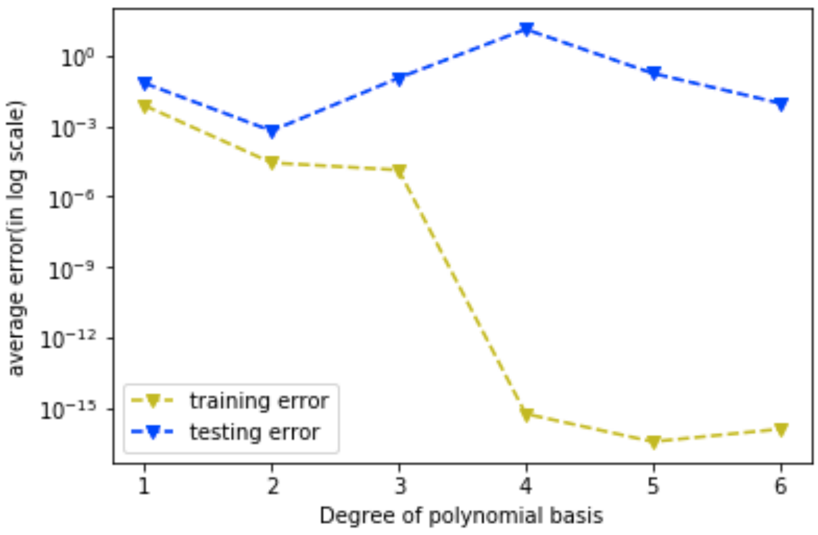

Fig.5 Degree of polynomial basis

According to Fig.5, we can see the best degree of polynomial basis is 2. Then M=2 is used to train on the entire dataset so that we can get the best(i.e., the lowest average test error) model(Fig.6).

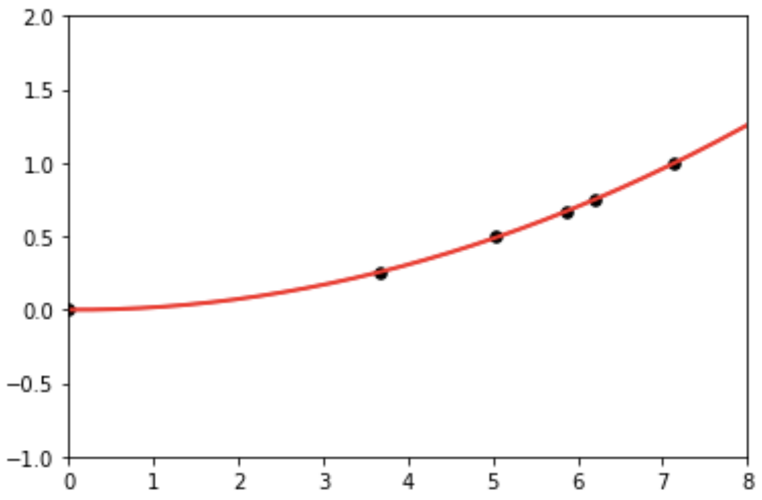

Fig.6 The lowest average test error model

Complete code:

```
from __future__ import division
```

```
import numpy as np

import numpy.matlib

import matplotlib.pyplot as plt

import pylab

import math


def read_data(filename):

    data = np.array(np.genfromtxt(filename, delimiter=','))

    X = np.reshape(data[:, 0], (1,np.size(data[:, 0])))

    y = np.reshape(data[:, 1], (np.size(data[:, 1]), 1))

    return X, y


def poly_basis(X, D):

    temp = np.ones((1, np.shape(X)[1]))

    F = X

    for i in range(2, D + 1):

        F = np.row_stack((F, X ** i))

    F = np.row_stack((temp, F))

    return F


def least_square_sol(F, y):

    '''

    Refer to eq. 5.19 in the text

    '''

    w = np.dot(np.dot(np.linalg.pinv(np.dot(F, F.T)), F), y)

    return w


def mean_square_error(w, F, y):

    res = np.dot(F.T, w) - y

    res = res * res

    mse = np.mean(res, axis=0)

    return mse
```

```
def random_split(P, K):
    temp = np.random.permutation(P)
    folds = np.split(temp, K)
    return folds


def train_val_split(X, y, folds, fold_id):
    X_val = np.zeros((1, len(folds[fold_id])))
    y_val = np.zeros((len(folds[fold_id]), 1))
    X_train = np.zeros((1, (len(folds) - 1) * len(folds[fold_id])))
    y_train = np.zeros(((len(folds) - 1) * len(folds[fold_id]), 1))
    j = 0
    for i in folds[fold_id]:
        X_val[0][j] = X[0][i]
        y_val[j][0] = y[i][0]
        j = j + 1
    count = 0
    for k in range(0, len(folds)):
        if k != fold_id:
            for l in folds[k]:
                X_train[0][count] = X[0][l]
                y_train[count][0] = y[l][0]
                count = count + 1
    return X_train, y_train, X_val, y_val


def make_plot(D, MSE_train, MSE_val):
    plt.figure()
    train, = plt.plot(D, MSE_train, 'yv--')
    val, = plt.plot(D, MSE_val, 'bv--')
    plt.legend(handles=[train, val], labels=['training error', 'testing error'])
    plt.xlabel('Degree of polynomial basis')
    plt.ylabel('average error(in log scale)')
    plt.yscale('log')
    plt.show()
```

```
def load_data():
    data = np.array(np.genfromtxt('galileo_ramp_data.csv', delimiter=','))
    x = np.reshape(data[:,0],(np.size(data[:,0]),1))
    y = np.reshape(data[:,1],(np.size(data[:,1]),1))
    return x,y

# YOUR CODE GOES HERE takes poly features of the input
def poly_features(x,D):
    F = []
    for i in x:
        F.append(1)
        for j in range(1,D+1):
            F.append(i**j)
    F=np.array(F)
    F.shape=(len(x),D+1)
    F=F.T
    return F

# plot the polynomial
def plot_model(w,D):
    # plot determined surface in 3d space
    s = np.arange(0,8,.01)
    f = poly_features(s,D)
    z = np.dot(f.T,w)

    # plot contour in original space
    plt.plot(s,z, color = 'r', linewidth = 2)
    plt.ylim([-1,2])
    plt.xlim([0,8])

# plot data
def plot_data(x,y,deg):
        plt.scatter(x,y,s = 30, color = 'k')

# run over all the degrees, fit each models, and calculate errors
def try_all_degs(x,y,deg_range):
    # plot datapoints - one panel for each deg in deg_range
    plot_data(x,y,deg_range)

    # generate nonlinear features
    mses = []

    for D in np.arange(0,np.size(deg_range)):
        # generate poly feature transformation
        F = poly_features(x,deg_range[D])

        # get weights for current model
        temp = np.linalg.pinv(np.dot(F,F.T))
        w = np.dot(np.dot(temp,F),y)
        MSE = np.linalg.norm(np.dot(F.T,w)-y)/np.size(y)
        mses.append(MSE)

        # plot fit to data
        plot_model(w,deg_range[D])

# load data and defined degree range
X, y = read_data('galileo_ramp_data.csv')
num_fold, num_degree = 6, 6
folds = random_split(P=y.size, K=num_fold)

MSE_train, MSE_val = [0]*num_degree, [0]*num_degree
D = np.arange(1, num_degree+1)
for f in range(num_fold):
     X_train, y_train, X_val, y_val = train_val_split(X, y, folds, fold_id=f)
     for i, d in enumerate(D):
            F_train = poly_basis(X_train, D=d)
            F_val = poly_basis(X_val, D=d)
            w = least_square_sol(F_train, y_train)
            MSE_train[i]=MSE_train[i]+mean_square_error(w, F_train, y_train)/num_fold
            MSE_val[i]=MSE_val[i]+ mean_square_error(w, F_val, y_val)/num_fold
print ('The best degree of polynomial basis is %d' % (MSE_val.index(min(MSE_val))+1))
make_plot(D, MSE_train, MSE_val)
x, y = load_data()
deg_range =[(MSE_val.index(min(MSE_val))+1)]                # degree polys to try
# run all over degree range
try_all_degs(x,y,deg_range)
```