Programming Assignment I

CS5691: Pattern Recognition and Machine Learning

Team 17 CCS Section Spring 2021

September 23, 2022

Aanand Krishnan
BE17B001

Manoranjan J
NA17B112

Reneeth Krishna MG
BS17B025

Contents

I Polynomial Curve Fitting

I.1 Python Code

```
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import random
from matplotlib import cm

def process_dataset_1(degree, lamb, batch):

    train = int(0.7*batch)    #70% of data
    valid = int(0.2*batch)    #20 % of data
    test = int(batch-(train+valid))   #10 % of data

    #generating polynomial combinations
    def polybasisfun(vec,deg):
        def degcomb(vec,deg):
            if deg == 0:
                return [1]
            if deg == 1:
                return vec
            if len(vec) == 1:
                return [vec[0]**deg]
            u = []
            for i in range(deg+1):
                u+=([(vec[0]**(i))*x for x in (degcomb(vec[1:],deg-i))])
            return u
        u = np.array([1])
        for i in range(1,deg+1):
            u=np.append(u,degcomb(vec,i))
        return u

    # Read data from csv file
    f0 = pd.read_csv('datasets/function1.csv')
    odata = f0.to_numpy()
    np.random.shuffle(odata)

    # split into train,validate,test
    data = odata[:train,1:2]
    validata = odata[-(valid):,1:2]
    testdata = odata[-(valid+test):-(valid),1:2]
    yd = odata[:train,2]
    yv = odata[-(valid):,2]
    yt = odata[-(valid+test):-(valid),2]

    # formulate X matrix
    X = np.apply_along_axis(polybasisfun,1,data,degree)
```

```python

    # calculate weights
    w = ((np.linalg.inv((lamb*np.eye(np.shape(X)[1]))+((np.transpose(X))@X)))@((np.tr


    # Get predictions
    def output(v,deg,w):
        return (w@(np.transpose(polybasisfun(v,deg))))
    yp = np.apply_along_axis(output,1,data,deg=degree,w=w)
    yvp = np.apply_along_axis(output,1,validata,deg=degree,w=w)
    ytp = np.apply_along_axis(output,1,testdata,deg=degree,w=w)


    print('ERMS_training = %f'%(np.linalg.norm((yp-yd),2)/np.sqrt(train)))
    print('ERMS_test = %f'%(np.linalg.norm((ytp-yt),2)/np.sqrt(test)))
    print('ERMS_valid = %f'%(np.linalg.norm((yvp-yv),2)/np.sqrt(valid)))


    x = np.linspace(min(data),max(data),50)
    x = np.reshape(x,(x.shape[0],1))
    y = np.apply_along_axis(lambda x:output(x,degree,w),1,x)


    fig0 = plt.figure(figsize=(9,9))
    plt.plot(x,y,color='red')
    plt.scatter(data,yd,color='blue')
    plt.title("degree = {} | lambda = {}".format(degree,lamb),fontsize = 20)
    plt.xlabel('X-axis(x)')
    plt.ylabel('Y-axis(y)')


    return np.linalg.norm((yp-yd),2)/np.sqrt(train), np.linalg.norm((ytp-yt),2)/np.sq

################################# Main Code ##################################

# # Call function to experiment dataset 1
Degree = [2,3,6,9]
lamb = [0,10e-2,10e-1,10,10e2]
batch = [10,200]

Erms = []
# Experiment with Degree
for l in lamb:
    erms = process_dataset_1(25, l, 350)
    Erms.append(erms)
```

II Linear Model for Regression using Polynomial Basis Functions

II.1 Python Code

```
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import random
from matplotlib import cm
#polynomial basis function regression
degree = 6
lamb = 0
batch = 500
train = int(0.7*batch)    #70% of data
test = int(0.2*batch)    #20 % of data
valid = batch-(train+test)   #10 % of data


#generating polynomial combinations
def polybasisfun(vec,deg):
    def degcomb(vec,deg):
        if deg == 0:
            return [1]
        if deg == 1:
            return vec
        if len(vec) == 1:
            return [vec[0]**deg]
        u = []
        for i in range(deg+1):
            u+=([(vec[0]**(i))*x for x in (degcomb(vec[1:],deg-i))])
        return u
    u = np.array([1])
    for i in range(1,deg+1):
        u=np.append(u,degcomb(vec,i))
    return u
f0 = pd.read_csv('function1_2d.csv')
odata = f0.to_numpy()
np.random.shuffle(odata)
data = odata[:train,1:3]
validata = odata[-(valid):,1:3]
testdata = odata[-(valid+test):-(valid),1:3]
yv = odata[-(valid):,3]
yt = odata[-(valid+test):-(valid),3]
X = np.apply_along_axis(polybasisfun,1,data,degree)
yd = odata[:train,3]
w = ((np.linalg.inv((lamb*np.eye(np.shape(X)[1]))+((np.transpose(X))@X)))
                                                            @((np.transpose(X))@yd))

```

```python
def output(v,deg,w):
    return (w@(np.transpose(polybasisfun(v,deg))))

yp = np.apply_along_axis(output,1,data,deg=degree,w=w)
yvp = np.apply_along_axis(output,1,validata,deg=degree,w=w)
ytp = np.apply_along_axis(output,1,testdata,deg=degree,w=w)
print('ERMS_training = %f'%(np.linalg.norm((yp-yd),2)/np.sqrt(train)))
print('ERMS_test = %f'%(np.linalg.norm((ytp-yt),2)/np.sqrt(test)))
print('ERMS_valid = %f'%(np.linalg.norm((yvp-yv),2)/np.sqrt(valid)))
x = np.linspace(min(data[:,0]),max(data[:,0]),200)
y = np.linspace(min(data[:,1]),max(data[:,1]),200)
X,Y=np.meshgrid(x,y)
Z = (np.array([output([i,j],degree,w) for i,j in
     zip(np.ravel(X),np.ravel(Y))])).reshape(Y.shape)
fig = plt.figure(figsize=(16,12))
ax = plt.axes(projection='3d')
ax.plot_surface(X,Y,Z,color='red', alpha=0.2,)
ax.scatter(data[:,0],data[:,1],yd,alpha=1,color='black',linewidths=1)
ax.set_xlabel('X-axis(x1)',fontsize = 15)
ax.set_ylabel('Y-axis(x2)',fontsize = 15)
ax.set_zlabel('Z-axis(y)',fontsize = 15)
ax.set_title("batch size = {} | degree = {} | lambda = {}".format(batch,degree,
             lamb),fontsize = 20)
plt.show()

fig1 = plt.figure(figsize=(9,9))
plt.scatter(yd,yp,marker='x')
plt.title('training',fontsize=20)
plt.xlabel('target',fontsize=20)
plt.ylabel('predicted',fontsize=20)
fig2 = plt.figure(figsize=(9,9))
plt.scatter(yt,ytp,marker='x')
plt.title('test',fontsize=20)
plt.xlabel('target',fontsize=20)
plt.ylabel('predicted',fontsize=20)
fig3 = plt.figure(figsize=(9,9))
plt.scatter(yv,yvp,marker='x')
plt.title('validation',fontsize=20)
plt.xlabel('target',fontsize=20)
plt.ylabel('predicted',fontsize=20)
```

III Linear Model for Regression using Gaussian Basis Functions

III.1 Python Code

```
# -*- coding: utf-8 -*-


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

################################ Dataset 3 ##################################

def process_dataset_3(D, var, r):

    """
    D: No of gauss functions for fit
    var: variance for fit
    r: regularisation term
    """
    # Read txt file

    f = open('datasets/2_music.txt', 'r')

    length = 0
    for line in f:

        d = [float(i) for i in line.split(',')]

        if length == 0:
            data = np.array(d)
            data = data[np.newaxis,:]
        else:
            d = np.array(d)
            d = d[np.newaxis,:]
            data = np.append(data,d,axis=0)
        length = length+1

    f.close()

    data = pd.DataFrame(data)


    # shuffle dataset
    data = data.sample(frac=1)

    # get dependent and independent varaible
    data = np.array(data)


```

```
    X = data[:,0:-2]
    y = data[:,-2:]

    # get unique values of y
    y_unique = np.unique(y,axis=0)

    # length of data for fit
    train_len = int(np.shape(X)[0]*0.7)
    val_len = int(np.shape(X)[0]*0.2)
    test_len = int(np.shape(X)[0]) - val_len - train_len

    # test train split
    X_train = X[0:train_len]
    X_test = X[train_len:train_len+test_len]
    X_val = X[train_len+test_len:train_len+test_len+val_len]

    y_train = y[0:train_len]
    y_test = y[train_len:train_len+test_len]
    y_val = y[train_len+test_len:train_len+test_len+val_len]

    # K-means clustering
    loop = 1
    prev_zni = np.zeros((train_len,D-1))

    while(loop>0):
        zni = np.zeros((train_len,D-1))

        if loop == 1:
            # randomly choose k points
            random_index = np.random.randint(0,train_len,D-1)

            # Initialize MUi
            MUi = X_train[random_index,:]

        # Determine points belonging to clusters
        for j in range(0,train_len):
            i = np.argmin(np.linalg.norm(X_train[j,:]-MUi,axis=1),axis=0)
            zni[j,i] = 1

        # Determine number of datapoints in the clusters
        Ni = np.sum(zni,axis=0)

        # Update MUi
        for j in range(0,D-1):
            if Ni[j] == 0:
                continue

            z = zni[:,j]
            MUi[j,:] = (np.sum(X_train*z[:,np.newaxis],axis=0))/Ni[j]

```

```
        # compare previous and current Zni
        comp = prev_zni - zni

        prev_zni = zni
        loop = loop+1

        # if no change break the loop
        if np.max(comp) == 0 and np.min(comp) == 0:
            break

    # Assign Mui
    mu = MUi

    # gaussian function
    phii = lambda x,mu : np.e**(-(np.linalg.norm(x-mu))/var**2)

    # Formulate phi matrix for training data
    phin = np.zeros((train_len,D))

    for i in range(0,train_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
                #phin[i,j] = phii(X_train[i,:],mu[j,:])
            else:
                phin[i,j] = phii(X_train[i,:],mu[j,:])

    # weights with quadratic regularisation
    W = np.linalg.inv(phin.T@phin + r*np.identity(D))@phin.T@y_train

    # weights with tikhanov regularisation
    # phin_bar = np.zeros((D,D))
    # for i in range(0,D):
    #     for j in range(0,D):
    #         phin_bar[i,j] = np.exp(np.linalg.norm(mu[i,:]-mu[j,:])/var**2)

    # W = np.linalg.inv(phin.T@phin + r*phin_bar)@phin.T@y_train

    ##############################################################################

    # Test on training dataset
    y_pred = np.zeros((train_len,2))

    # Formulate phi matrix for training data
    phin = np.zeros((train_len,D))

    for i in range(0,train_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
```

```
                # phin[i,j] = phii(X_train[i,:],mu[j,:])
            else:
                phin[i,j] = phii(X_train[i,:],mu[j])

    for i in range(0,train_len):
        pred = (phin[i,:])@W
        y_pred[i,:] = pred

        # term1 = np.deg2rad((pred[0] - y_unique[:,0])/2)
        # term2 = np.deg2rad((pred[1] - y_unique[:,1])/2)
        # a = (np.sin(term1)**2) + # np.cos(np.deg2rad(pred[0]))*
        #       np.cos(np.deg2rad(y_unique[:,0]))*(np.sin(term2# )**2)
        # dist = 2*6373*np.arctan2(np.sqrt(a),np.sqrt(1-a))

        # index = np.argmin(dist)
        # y_pred[i,:] = y_unique[index,:]

    Erms_train1 = ((np.linalg.norm(y_pred[:,0]-y_train[:,0])))/np.sqrt(train_len)
    Erms_train2 = ((np.linalg.norm(y_pred[:,1]-y_train[:,1])))/np.sqrt(train_len)

    # # scatter plot
    # fig1 = plt.figure(figsize=(9,9))
    # plt.scatter(y_train[:,0],y_pred[:,0],marker='x')
    # plt.scatter(y_train[:,1],y_pred[:,1],marker='x')
    # plt.xlabel('target',fontsize=20)
    # plt.ylabel('predicted',fontsize=20)
    # plt.title("Gaussian Basis Function (For Dataset 3 of # best performing
    #               model with training data) | degree = {} | # lambda = {}".format(D
    #               fontsize = 20)
    # plt.legend(["y0","y1"])
    ############################################################################

    # Test on validation dataset
    y_pred = np.zeros((val_len,2))

    # Formulate phi matrix for validation data
    phin = np.zeros((val_len,D))

    for i in range(0,val_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
                # phin[i,j] = phii(X_val[i,:],mu[j,:])
            else:
                phin[i,j] = phii(X_val[i,:],mu[j])

    for i in range(0,val_len):
        pred = (phin[i,:])@W
        y_pred[i,:] = pred

```

```python

    Erms_val1 = ((np.linalg.norm(y_pred[:,0]-y_val[:,0])))/np.sqrt(val_len)
    Erms_val2 = ((np.linalg.norm(y_pred[:,1]-y_val[:,1])))/np.sqrt(val_len)

    # scatter plot for validation data
    # fig1 = plt.figure(figsize=(9,9))
    # plt.scatter(y_val,y_pred,marker='x')
    # plt.xlabel('target',fontsize=20)
    # plt.ylabel('predicted',fontsize=20)
    # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
    #             with validation data) | degree = {} | lambda # = {}".format(D,r),
    #              fontsize = 20)

    ################################################################################

    # Test on test dataset
    y_pred = np.zeros((test_len,2))

    # Formulate phi matrix for test data
    phin = np.zeros((test_len,D))

    for i in range(0,test_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
                #phin[i,j] = phii(X_test[i,:],mu[j,:])
            else:
                phin[i,j] = phii(X_test[i,:],mu[j])

    for i in range(0,test_len):
        pred = (phin[i,:])@W
        y_pred[i,:] = pred

    Erms_test1 = ((np.linalg.norm(y_pred[:,0]-y_test[:,0])))/np.sqrt(test_len)
    Erms_test2 = ((np.linalg.norm(y_pred[:,1]-y_test[:,1])))/np.sqrt(test_len)

    # scatter plot for test data
    fig1 = plt.figure(figsize=(9,9))
    plt.scatter(y_test[:,0],y_pred[:,0],marker='x')
    plt.scatter(y_test[:,1],y_pred[:,1],marker='x')
    plt.xlabel('target',fontsize=20)
    plt.ylabel('predicted',fontsize=20)
    plt.title("Gaussian Basis Function (For Dataset 3 of best performing model
              with test data) | degree = {} | lambda = {}".format(D,r),
               fontsize = 20)
    plt.legend(["y0","y1"])
    ############################################################################
    return Erms_train1, Erms_train2, Erms_val1, Erms_val2, Erms_test1, Erms_test2

################################ Dataset 2 ####################################
```

```python

def process_dataset_2(D, var, r):

    """
    D: No of gauss functions for fit
    var: variance for fit
    r: regularisation term
    """

    # Read csv file
    data = pd.read_csv('datasets/function1_2d.csv')

    # shuffle dataset
    # data = data.sample(frac=1)

    # get dependent and independent varaible
    X = data[['x1','x2']]
    y = data['y']

    # convert into numpy
    X = np.array(X)
    y = np.array(y)

    # length of data for fit
    train_len = int(np.shape(X)[0]*0.7)
    test_len = int(np.shape(X)[0]*0.2)
    val_len = int(np.shape(X)[0]*0.1)

    # test train split
    X_train = X[0:train_len]
    X_test = X[train_len:train_len+test_len]
    X_val = X[train_len+test_len:train_len+test_len+val_len]

    y_train = y[0:train_len]
    y_test = y[train_len:train_len+test_len]
    y_val = y[train_len+test_len: train_len+test_len+val_len]

    # K-means clustering
    loop = 1
    prev_zni = np.zeros((train_len,D-1))

    while(loop>0):
        zni = np.zeros((train_len,D-1))

        if loop == 1:
            # randomly choose k points
            random_index = np.random.randint(0,train_len,D-1)

            # Initialize MUi
            MUi = X_train[random_index,:]
```

```python

        # Determine points belonging to clusters
        for j in range(0,train_len):
            i = np.argmin(np.linalg.norm(X_train[j,:]-MUi,axis=1),axis=0)
            zni[j,i] = 1

        # Determine number of datapoints in the clusters
        Ni = np.sum(zni,axis=0)

        # Update MUi
        for j in range(0,D-1):
            if Ni[j] == 0:
                continue

            z = zni[:,j]
            MUi[j,:] = (np.sum(X_train*z[:,np.newaxis],axis=0))/Ni[j]

        # compare previous and current Zni
        comp = prev_zni - zni

        prev_zni = zni
        loop = loop+1

        # if no change break the loop
        if np.max(comp) == 0 and np.min(comp) == 0:
            break

    # Assign Mui
    mu = MUi

    # gaussian function
    phii = lambda x,mu : np.e**(-(np.linalg.norm(x-mu))/var**2)

    # Formulate phi matrix for training data
    phin = np.zeros((train_len,D))

    for i in range(0,train_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
            else:
                phin[i,j] = phii(X_train[i,:],mu[j-1,:])


    # weights with quadratic regularisation
    W = np.linalg.inv(phin.T@phin + r*np.identity(D))@phin.T@y_train

    ##########################################################################

    # Test on training dataset
```

```
    y_pred = np.zeros((train_len,1))

    # Formulate phi matrix for training data
    phin = np.zeros((train_len,D))

    for i in range(0,train_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
            else:
                phin[i,j] = phii(X_train[i,:],mu[j-1])

    for i in range(0,train_len):
        y_pred[i,:] = (phin[i,:])@W

    Erms_train = ((np.linalg.norm(y_pred-y_train[:,np.newaxis])))/
                   np.sqrt(train_len)


    # # scatter plot for training data
    # fig1 = plt.figure(figsize=(9,9))
    # plt.scatter(y_train,y_pred,marker='x')
    # plt.xlabel('target',fontsize=20)
    # plt.ylabel('predicted',fontsize=20)
    # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
    #              with training data) | degree = {} | lambda = # {}".format(D,r),
    #               fontsize = 20)


    ############################################################################

    # Test on validation dataset
    y_pred = np.zeros((val_len,1))

    # Formulate phi matrix for validation data
    phin = np.zeros((val_len,D))

    for i in range(0,val_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
            else:
                phin[i,j] = phii(X_val[i,:],mu[j-1])

    for i in range(0,val_len):
        y_pred[i,:] = (phin[i,:])@W

    Erms_val = ((np.linalg.norm(y_pred-y_val[:,np.newaxis])))/np.sqrt(val_len)

    # scatter plot for validation data
```

```python
    # fig1 = plt.figure(figsize=(9,9))
    # plt.scatter(y_val,y_pred,marker='x')
    # plt.xlabel('target',fontsize=20)
    # plt.ylabel('predicted',fontsize=20)
    # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
    #               with validation data) | degree = {} | lambda #= {}".format(D,r),
    #                fontsize = 20)

    ############################################################################

    # Test on test dataset
    y_pred = np.zeros((test_len,1))

    # Formulate phi matrix for test data
    phin = np.zeros((test_len,D))

    for i in range(0,test_len):
        for j in range(0,D):
            if j==0:
                phin[i,j] = 1
            else:
                phin[i,j] = phii(X_test[i,:],mu[j-1])

    for i in range(0,test_len):
        y_pred[i,:] = (phin[i,:])@W

    Erms_test = ((np.linalg.norm(y_pred-y_test[:,np.newaxis])))/np.sqrt(test_len)

    # # scatter plot for validation data
    # fig1 = plt.figure(figsize=(9,9))
    # plt.scatter(y_test,y_pred,marker='x')
    # plt.xlabel('target',fontsize=20)
    # plt.ylabel('predicted',fontsize=20)
    # plt.title("Gaussian Basis Function (For Dataset 2 of best performing
    #               model with test data) | degree = {} | lambda # = {}".format(D,r),
    #                fontsize = 20)

    ############################################################################
    return Erms_train, Erms_val, Erms_test


############################ Main Code #########################################

# Call function to experiment dataset 2/3
D = [2,3,6,9,20,33,40,100]
var = [1,3,5,10,50,100]
reg = [0,10e-5,10e-4,10e-3,10e-2,10e-1,10,10e2,10e3,10e4,10e5]

Erms = []
# Experiment with variance
```

```python
for v in var:
    erms = process_dataset_3(100, v, 0)
     Erms.append(erms)

# Plot table
fig = plt.figure(figsize=(9,9))
row_labels=["var = 1", "var = 3", "var = 5", "var = 10", "var = 50", "var = 100"]
column_labels=["Erms_train", "Erms_val", "Erms_test"]
column_labels=["Erms_train_y0","Erms_train_y1", "Erms_val0","Erms_val1",
                 "Erms_test0", "Erms_test1"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=Erms,rowLabels=row_labels,colLabels=column_labels, loc="center")

Erms = []
# Experiment with Dimensions
for d in D:
    erms = process_dataset_3(d, 3, 0)
    Erms.append(erms)

# Plot table
fig = plt.figure(figsize=(9,9))
row_labels=["D = 2", "D = 3", "D = 6", "D = 9", "D = 20", "D = 30",
              "D = 40", "D = 100"]
column_labels=["Erms_train", "Erms_val", "Erms_test"]
column_labels=["Erms_train_y0","Erms_train_y1", "Erms_val0","Erms_val1",
                 "Erms_test0", "Erms_test1"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=Erms,rowLabels=row_labels,colLabels=column_labels,
            loc="center")

Erms = []
# Experiment with regularisation
for r in reg:
    erms = process_dataset_3(100, 3, r)
    Erms.append(erms)

# Plot table
fig = plt.figure(figsize=(9,9))
row_labels=["lambda = 0", "lambda = 10e-5", "lambda = 10e-4", "lambda = 10e-3",
              "lambda = 10e-2", "lambda = 10e-1", "lambda = 10", "lambda = 10e2",
              "lambda = 10e3", "lambda = 10e4", "lambda = 10e5"]
column_labels=["Erms_train", "Erms_val", "Erms_test"]
column_labels=["Erms_train_y0","Erms_train_y1", "Erms_val0","Erms_val1",
                 "Erms_test0", "Erms_test1"]
plt.axis('tight')
plt.axis('off')
plt.table(cellText=Erms,rowLabels=row_labels,colLabels=column_labels,
            loc="center")
```

IV Direct links For Python Codes

- Polynomial Curve Fitting
- Linear Model for Regression using Polynomial Basis Functions
- Linear Model for Regression using Gaussian Basis Functions