ASSIGNMENT

CS5691 Pattern Recognition and Machine Learning

CS5691 Assignment 1 - Code

Team Members:

BE17B007	N Sowmya Manojna
PH17B010	Thakkar Riya Anandbhai
PH17B011	Chaithanya Krishna Moorthy

Indian Institute of Technology, Madras

Contents

1 Task 1

1.1 Main Code

The code for Question 1 is as follows:

```
#####################################################################
# ## CS5691 PRML Assignment 1
# **Team 1**
# **Team Members:**
# N Sowmya Manojna   BE17B007
# Thakkar Riya Anandbhai  PH17B010
# Chaithanya Krishna Moorthy  PH17B011

#####################################################################
# Install required Packages
# Uncomment if you are running for the first time
# !pip install -r requirements.txt
# try:
#     !mkdir images/q1
# except:
#     pass

#####################################################################
import numpy as np
np.random.seed(0)
import pandas as pd

import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
plt.style.use('science')
plt.rcParams['font.size'] = 18
plt.rcParams['axes.grid'] = True
plt.rcParams["grid.linestyle"] = (5,9)
plt.rcParams['figure.figsize'] = 8,6

from regression import PolynomialRegression
from gridsearch import GridSearch
#####################################################################
df = pd.read_csv("../datasets/function1.csv", index_col=0)
df.sort_values(by=["x"], inplace=True)
df.head()

#####################################################################
lmbda_list = [0, 0.5, 1, 2, 10, 50, 100]
degrees_allowed = [2, 3, 6, 9]
datasizes_considered = [10, 200]
complete_dataset_size = df.shape[0]

X = df["x"].to_numpy().reshape(-1,1)
y = df["y"].to_numpy().reshape(-1,1)

results_df_list = []
correspondance_list = []
for sample_size in datasizes_considered:
    gridsearch = GridSearch()
    df_result, correspondance = gridsearch.get_result(df, sample_size=...
        sample_size, degrees_allowed=degrees_allowed, lmbda_list=lmbda_list...
        )
```

```
    results_df_list.append(df_result)
    correspondance_list.append(correspondance)

    print("\nFor Sample Size of ", sample_size, " - GridSearch Results:")
    print(df_result)
    print("="*70)
    gridsearch.get_plots(X, y, correspondance, sample_size, show=True)
####################################################################

# From the resuts obtained, we see that degree=6, lambda=0.0
# best fits the model.

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

best_degree = int(df_result.iloc[0]["degree"])
best_lmbda = df_result.iloc[0]["lambda"]

df_train = df.sample(frac=0.7, random_state=42)
df_test = df[~df.index.isin(df_train.index)]

X_train = df_train["x"].to_numpy().reshape(-1,1)
X_test = df_test["x"].to_numpy().reshape(-1,1)
y_train = df_train["y"].to_numpy().reshape(-1,1)
y_test = df_test["y"].to_numpy().reshape(-1,1)

regressor = PolynomialRegression()
X_train_poly = regressor.fit(X_train, y_train, degree=best_degree, lmbda=...
    best_lmbda)
y_train_pred = regressor.transform(X_train)
y_test_pred = regressor.transform(X_test)
train_error = regressor.error(y_train, y_train_pred)
test_error = regressor.error(y_test, y_test_pred)

print("Training Error:", train_error)
print("Testing Error:", test_error)
```

1.2 Polynomial Regression Code

The helper class used to perform `PolynomialRegression` is as follows:

```
import numpy as np

class PolynomialRegression():
    def __init__(self):
        pass

    def fit(self, X, y, degree=2, lmbda=0):
        self.degree = degree
        self.lmbda = lmbda

        X_poly = self.get_polynomial_features(X)
        self.get_weights(X_poly, y)
        return X_poly

    def transform(self, X_val):
        X_poly = self.get_polynomial_features(X_val)
        y_val = X_poly @ self.W
        return y_val

```

```
    def fit_transform(self, X, y, degree=2, lmbda=0):
        self.fit(X, y, degree, lmbda)
        return self.transform(X)

    def get_polynomial_features(self, X):
        X_new = np.ones(X.shape)
        for i in range(1, self.degree+1):
            X_new = np.append(X_new, X**i, axis=1)
        return X_new

    def get_weights(self, X_poly, y):
        d = X_poly.shape[1]
        self.W = ((np.linalg.inv(X_poly.T @ X_poly + self.lmbda*np.identity...
            (d)) @ X_poly.T) @ y

    def error(self, y_true, y_pred):
        rmse = np.linalg.norm(y_pred-y_true)/(y_true.size)**0.5
        return rmse
```

1.3 Grid Search Code

The helper class used to perform `Grid Search` are as follows:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from regression import PolynomialRegression

class GridSearch():
    def __init__(self):
        pass

    def get_result(self, df, sample_size, degrees_allowed, lmbda_list):
        df_sample = df.sample(n=sample_size, random_state=42)

        df_train = df_sample.sample(frac=0.9, random_state=42)
        df_val = df_sample[~df_sample.index.isin(df_train.index)]

        X_train = df_train["x"].to_numpy().reshape(-1, 1)
        X_val = df_val["x"].to_numpy().reshape(-1, 1)

        y_train = df_train["y"].to_numpy().reshape(-1, 1)
        y_val = df_val["y"].to_numpy().reshape(-1, 1)

        self.result = []
        self.correspondance = {}

        for degree in degrees_allowed:
            for lmbda in lmbda_list:
                regressor = PolynomialRegression()
                regressor.fit(X_train, y_train, degree=degree, lmbda=lmbda)
                y_train_pred = regressor.transform(X_train)
                y_val_pred = regressor.transform(X_val)

                train_error = regressor.error(y_train, y_train_pred)
                val_error = regressor.error(y_val, y_val_pred)

                self.result.append([degree, lmbda, train_error, val_error])
```

```
                self.correspondance[(degree, lmbda)] = {"df_sample":...
                    df_sample, "regressor":regressor}

        df_results = pd.DataFrame(self.result, columns=["degree", "lambda",...
             "Train error", "Validation error"])
        df_results["Sum Error"] = df_results["Train error"] + df_results["...
            Validation error"]
        df_results.sort_values(by="Sum Error", inplace=True)
        return df_results, self.correspondance

    def get_plots(self, X, y, correspondance, sample_size, show):
        for key in correspondance:
            df_sample = correspondance[key]["df_sample"]
            df_sample.sort_values(by=["x"], inplace=True)
            X_sample = df_sample["x"].to_numpy().reshape(-1,1)
            y_sample = df_sample["y"].to_numpy().reshape(-1,1)

            regressor = correspondance[key]["regressor"]
            y_pred_sample = regressor.transform(X_sample)

            title = "Curve Fitting - Degree: "+str(regressor.degree)\
            +"; Sample Size: "+str(sample_size)+"; $\lambda$: "\
            +str(regressor.lmbda)
            fname = "d_"+str(regressor.degree)+"_size_"+str(sample_size)+"...
                _l_"+str(regressor.lmbda)+".png"

            plt.figure()
            plt.plot(X, y, label="True Value")
            if y_sample.size >= 100:
                plt.plot(X_sample, y_sample, 'r.', alpha=0.5, label="...
                    Sampled points")
            else:
                plt.plot(X_sample, y_sample, 'ro', alpha=0.75, label="...
                    Sampled points")
            plt.plot(X_sample, y_pred_sample, label="Predicted Value")
            if title:
                plt.title(title)
            plt.xlabel("X-values")
            plt.ylabel("Y-values")
            plt.legend()
            plt.savefig("images/"+fname)
            if show:
                plt.show()
```

2 Task 2

The code for Question 2 is as follows:

```
####################################################################
from IPython import get_ipython

####################################################################
import numpy as np
import pandas as pd
import math as ma
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
get_ipython().run_line_magic('matplotlib', 'inline')
```

```

##################################################################
# # Task 2

##################################################################
func2d=pd.read_csv("function1_2d.csv",index_col = 0)

##################################################################
# ## 2.1 Generating the polynomial basis functions of degrees 2, 3 and 6: (...
    Sp to dataset 2)

##################################################################
### Creating the polynomial basis functions of degree M and number of ...
    examples = n:

def create_phi(M,n,x1,x2):
    d=2
    D = int(ma.factorial(d+M)/(ma.factorial(d)*ma.factorial(M)))
    phi = np.zeros((n,D))

    if M == 2:
        exp_ar = [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1]]
    if M == 3:
        exp_ar = ...
            [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1],[2,1],[1,2],[3,0],[0,3]]
    if M == 6:
        exp_ar = [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1],[2,1],[1,2],\
                  [3,0],[0,3],[3,1],[1,3],[2,2],[4,0],[0,4],[4,1],\
                  [1,4],[2,3],[3,2],[5,0],[0,5],[5,1],[1,5],[2,4],\
                  [4,2],[3,3],[6,0],[0,6]]

    for i in range(D):
        phi[:,i] = (x1**(exp_ar[i][0]))*(x2**(exp_ar[i][1]))

    return(phi)

##################################################################
# ## 2.2 Solving for optimal parameters using regularization, lambda=0  for...
    unregularized: (versatile)
##################################################################
# ### The function regularized_pseudo_inv(lamb,X) returns:
# $$(\lambda I+X^TX)^{-1}X^T$$
#
# Where lambda is the hyperparameter in the quadratic regularization

##################################################################
def regularized_pseudo_inv(lamb,phi):
    return(np.matmul(np.linalg.inv(lamb*np.identity(phi.shape[1])+np.matmul...
        (np.transpose(phi),phi)),np.transpose(phi)))

##################################################################
# ### The function opt_regularized_param(lamb,phi,y) returns an array of ...
    optimal parameter values calculated using regularized cost function for...
    an input design matrix phi, hyperparameter lambda and output values y.

##################################################################
def opt_regularized_param(lamb,phi,y):
    return(np.matmul(regularized_pseudo_inv(lamb,phi),y))

##################################################################
```

```
# ## 2.3 The function y_pred(X,w) returns predicted function values for ...
    input matrix X and set of chosen parameter values w a: (versatile)
# $$y=Xw$$

###################################################################
def y_pred(phi,w):
    return(np.matmul(phi,w))

###################################################################
# ## 2.4 Splitting the data into train, cross-validation and test and ...
    helper function for surface plot: (versatile)

###################################################################
def create_datasets(data,train_size,cv_size):
    data.sample(frac=1).reset_index(drop=True)
    data_train=data[0:train_size]
    data_cv=data[train_size:train_size+cv_size]
    data_test=data[cv_size+train_size:]
    return(data_train,data_cv,data_test)



###################################################################
def split_cols(data_train,data_cv,data_test):
    #x1_train,x2_train,y_train,x1_cv,x2_cv,y_cv,x1_test,x2_test,y_test
    x1_train=np.array(data_train)[:,0]
    x2_train=np.array(data_train)[:,1]
    y_train=np.array(data_train)[:,2]
    x1_cv=np.array(data_cv)[:,0]
    x2_cv=np.array(data_cv)[:,1]
    y_cv=np.array(data_cv)[:,2]
    x1_test=np.array(data_test)[:,0]
    x2_test=np.array(data_test)[:,1]
    y_test=np.array(data_test)[:,2]

    return(x1_train,x2_train,y_train,x1_cv,x2_cv,y_cv,x1_test,x2_test,...
        y_test)



###################################################################
### Function to calculate RMSE:

def RMSE(y_pred, t):
    n = len(y_pred)
    return(np.sqrt(np.sum((y_pred - t)**2)/n))


###################################################################
#plot the approximated function


def plot_approxY(M,w,y_train,y_pred,train_size,l,a,b):

    # M is the degree of the complexity
    # w is the array of parameters
    # y is the actual value of y
    # l is the value of lambda
    x1 = np.arange(-16,16,0.5)
    x2 = np.arange(-16,16,0.5)
```

```python
    x1, x2 = np.meshgrid(x1,x2)
    Y= np.zeros((64,64))
    for i in range(64):
        for j in range(64):
            Y[i,j] = np.sum(np.matmul(create_phi(M,1,x2[i,j],x1[i,j]),w))
    fig = plt.figure(figsize=(15,8))
    ax=fig.gca(projection="3d")
    ax.plot_wireframe(x1,x2,Y,label="Approximated function")
    x = b
    y = a
    z = y_train
    ax.scatter(x,y,z,color='red',label="Original train data points")
    ax.set_ylabel("x1")
    ax.set_xlabel("x2")
    ax.set_zlabel("y")
    ax.view_init(0,45)
    plt.legend(loc=4)
    plt.title("Surface plot with degree of complexity = %i, Train data size...
        = %i and regularization parameter, lambda = %.1f"%(M,train_size,l)...
      )
    plt.show()

##################################################################
# ## 2.5 Predicting for degree 2, train size 50:

##################################################################
data2_train50,data2_cv50,data2_test50=create_datasets(func2d,50,30)


##################################################################
x12_train50,x22_train50,y2_train50,x12_cv50,x22_cv50,y2_cv50,x12_test50,...
   x22_test50,y2_test50=split_cols(data2_train50,data2_cv50,data2_test50)


##################################################################
### design matrix:
phi2_train50=create_phi(2,len(y2_train50),x12_train50,x22_train50)
phi2_cv50=create_phi(2,len(y2_cv50),x12_cv50,x22_cv50)
phi2_test50=create_phi(2,len(y2_test50),x12_test50,x22_test50)


##################################################################
y2_testpred50={}
y2_trainpred50={}
y2_cvpred50={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse2_train50=[]
rmse2_test50=[]
rmse2_cv50=[]


##################################################################
for l in lambda_list:
    w_2_50=opt_regularized_param(l,phi2_train50,y2_train50);
    y2_trainpred50[l]=y_pred(phi2_train50,w_2_50)
    y2_testpred50[l]=y_pred(phi2_test50,w_2_50);
    y2_cvpred50[l]=y_pred(phi2_cv50,w_2_50);
    rmse2_train50.append(RMSE(y2_trainpred50[l],y2_train50))
    rmse2_test50.append(RMSE(y2_testpred50[l],y2_test50))
    rmse2_cv50.append(RMSE(y2_cvpred50[l],y2_cv50))
```

```
#####################################################################
data2_50=pd.DataFrame(list(zip(lambda_list,rmse2_train50,rmse2_cv50,...
    rmse2_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


#####################################################################
data2_50


#####################################################################
plt.figure()
plt.plot(data2_50["Lambda"],data2_50["RMSE Train"],label="RMSE Train")
plt.plot(data2_50["Lambda"],data2_50["RMSE CV"],label="RMSE CV")
plt.plot(data2_50["Lambda"],data2_50["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 2"...
    )
plt.savefig("d2_50.png")
plt.show()

#####################################################################
# ### Surface plots for various values of lambda:

#####################################################################
for l in lambda_list:
    plot_approxY(2, w_2_50,y2_train50,y2_trainpred50[l],50,l,x12_train50,...
        x22_train50)
    plt.savefig("surfaceplotsd2.png")

#####################################################################
# ## 2.6 Predicting for degree of complexity = 2 and train data size = 200

#####################################################################
data2_train200,data2_cv200,data2_test200=create_datasets(func2d,200,90)


#####################################################################
x12_train200,x22_train200,y2_train200,x12_cv200,x22_cv200,y2_cv200,...
    x12_test200,x22_test200,y2_test200=split_cols(data2_train200,...
    data2_cv200,data2_test200)


#####################################################################
### design matrix:
phi2_train200=create_phi(2,len(y2_train200),x12_train200,x22_train200)
phi2_cv200=create_phi(2,len(y2_cv200),x12_cv200,x22_cv200)
phi2_test200=create_phi(2,len(y2_test200),x12_test200,x22_test200)


#####################################################################
y2_testpred200={}
y2_trainpred200={}
y2_cvpred200={}
lambda_list=[0,0.5,1,2,10,50,100]
```

```
rmse2_train200=[]
rmse2_test200=[]
rmse2_cv200=[]


#################################################################
for l in lambda_list:
    w_2_200=opt_regularized_param(l,phi2_train200,y2_train200);
    y2_trainpred200[l]=y_pred(phi2_train200,w_2_200)
    y2_testpred200[l]=y_pred(phi2_test200,w_2_200);
    y2_cvpred200[l]=y_pred(phi2_cv200,w_2_200);
    rmse2_train200.append(RMSE(y2_trainpred200[l],y2_train200))
    rmse2_test200.append(RMSE(y2_testpred200[l],y2_test200))
    rmse2_cv200.append(RMSE(y2_cvpred200[l],y2_cv200))



#################################################################
data2_200=pd.DataFrame(list(zip(lambda_list,rmse2_train200,rmse2_cv200,...
    rmse2_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


#################################################################
data2_200


#################################################################
data2_200.to_csv("RMSE-lambda for complexity = 2 and train size = 200")


#################################################################
plt.figure()
plt.plot(data2_200["Lambda"],data2_200["RMSE Train"],label="RMSE Train")
plt.plot(data2_200["Lambda"],data2_200["RMSE CV"],label="RMSE CV")
plt.plot(data2_200["Lambda"],data2_200["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 2...
    ")
plt.savefig("d2_200.png")
plt.show()

#################################################################
# ### Surface plots for various values of lambda:

#################################################################
for l in lambda_list:
    plot_approxY(2, w_2_200,y2_train200,y2_trainpred200[l],200,l,...
        x12_train200,x22_train200)

#################################################################
# ## 2.7 Predicting for degree of complexity = 2 and train data size = 500

#################################################################
data2_train500,data2_cv500,data2_test500=create_datasets(func2d,500,200)
x12_train500,x22_train500,y2_train500,x12_cv500,x22_cv500,y2_cv500,...
    x12_test500,x22_test500,y2_test500=split_cols(data2_train500,...
    data2_cv500,data2_test500)

```

```
### design matrix:
phi2_train500=create_phi(2,len(y2_train500),x12_train500,x22_train500)
phi2_cv500=create_phi(2,len(y2_cv500),x12_cv500,x22_cv500)
phi2_test500=create_phi(2,len(y2_test500),x12_test500,x22_test500)

y2_testpred500={}
y2_trainpred500={}
y2_cvpred500={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse2_train500=[]
rmse2_test500=[]
rmse2_cv500=[]

for l in lambda_list:
    w_2_500=opt_regularized_param(l,phi2_train500,y2_train500);
    y2_trainpred500[l]=y_pred(phi2_train500,w_2_500)
    y2_testpred500[l]=y_pred(phi2_test500,w_2_500);
    y2_cvpred500[l]=y_pred(phi2_cv500,w_2_500);
    rmse2_train500.append(RMSE(y2_trainpred500[l],y2_train500))
    rmse2_test500.append(RMSE(y2_testpred500[l],y2_test500))
    rmse2_cv500.append(RMSE(y2_cvpred500[l],y2_cv500))



###################################################################
data2_500=pd.DataFrame(list(zip(lambda_list,rmse2_train500,rmse2_cv500,...
    rmse2_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


###################################################################
data2_500


###################################################################
data2_500.to_csv("RMSE-lambda for complexity =2 train size =500")


###################################################################
plt.figure()
plt.plot(data2_500["Lambda"],data2_500["RMSE Train"],label="RMSE Train")
plt.plot(data2_500["Lambda"],data2_500["RMSE CV"],label="RMSE CV")
plt.plot(data2_500["Lambda"],data2_500["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 2...
    ")
plt.savefig("d2_500.png")
plt.show()

###################################################################
# ### Surface plots for various values of lambda:

###################################################################
for l in lambda_list:
    plot_approxY(2, w_2_500,y2_train500,y2_trainpred500[l],500,l,...
        x12_train500,x22_train500)

###################################################################
# ## 2.8 For train size =50 and degree of complexity = 3
```

```
####################################################################
data3_train50,data3_cv50,data3_test50=create_datasets(func2d,50,30)
x13_train50,x23_train50,y3_train50,x13_cv50,x23_cv50,y3_cv50,x13_test50,...
    x23_test50,y3_test50=split_cols(data3_train50,data3_cv50,data3_test50)

### design matrix:
phi3_train50=create_phi(3,len(y3_train50),x13_train50,x23_train50)
phi3_cv50=create_phi(3,len(y3_cv50),x13_cv50,x23_cv50)
phi3_test50=create_phi(3,len(y3_test50),x13_test50,x23_test50)

y3_testpred50={}
y3_trainpred50={}
y3_cvpred50={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse3_train50=[]
rmse3_test50=[]
rmse3_cv50=[]

for l in lambda_list:
    w_3_50=opt_regularized_param(l,phi3_train50,y3_train50);
    y3_trainpred50[l]=y_pred(phi3_train50,w_3_50)
    y3_testpred50[l]=y_pred(phi3_test50,w_3_50);
    y3_cvpred50[l]=y_pred(phi3_cv50,w_3_50);
    rmse3_train50.append(RMSE(y3_trainpred50[l],y3_train50))
    rmse3_test50.append(RMSE(y3_testpred50[l],y3_test50))
    rmse3_cv50.append(RMSE(y3_cvpred50[l],y3_cv50))



####################################################################
data3_50=pd.DataFrame(list(zip(lambda_list,rmse3_train50,rmse3_cv50,...
    rmse3_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


####################################################################
data3_50.to_csv("rmse lambda for complexity = 3 train sie =50")


####################################################################
plt.figure()
plt.plot(data3_50["Lambda"],data3_50["RMSE Train"],label="RMSE Train")
plt.plot(data3_50["Lambda"],data3_50["RMSE CV"],label="RMSE CV")
plt.plot(data3_50["Lambda"],data3_50["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 3"...
    )
plt.savefig("d3_50.png")
plt.show()

####################################################################
# ### Surface plots for various values of lambda

####################################################################
for l in lambda_list:
    plot_approxY(3, w_3_50,y3_train50,y3_trainpred50[l],50,l,x13_train50,...
        x23_train50)

```

```
################################################################
# ## 2.9 For train size = 200 and degree of complexity = 3:

################################################################
data3_train200,data3_cv200,data3_test200=create_datasets(func2d,200,90)
x13_train200,x23_train200,y3_train200,x13_cv200,x23_cv200,y3_cv200,...
    x13_test200,x23_test200,y3_test200=split_cols(data3_train200,...
    data3_cv200,data3_test200)

### design matrix:
phi3_train200=create_phi(3,len(y3_train200),x13_train200,x23_train200)
phi3_cv200=create_phi(3,len(y3_cv200),x13_cv200,x23_cv200)
phi3_test200=create_phi(3,len(y3_test200),x13_test200,x23_test200)

y3_testpred200={}
y3_trainpred200={}
y3_cvpred200={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse3_train200=[]
rmse3_test200=[]
rmse3_cv200=[]

for l in lambda_list:
    w_3_200=opt_regularized_param(l,phi3_train200,y3_train200);
    y3_trainpred200[l]=y_pred(phi3_train200,w_3_200)
    y3_testpred200[l]=y_pred(phi3_test200,w_3_200);
    y3_cvpred200[l]=y_pred(phi3_cv200,w_3_200);
    rmse3_train200.append(RMSE(y3_trainpred200[l],y3_train200))
    rmse3_test200.append(RMSE(y3_testpred200[l],y3_test200))
    rmse3_cv200.append(RMSE(y3_cvpred200[l],y3_cv200))



################################################################
data3_200=pd.DataFrame(list(zip(lambda_list,rmse3_train200,rmse3_cv200,...
    rmse3_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


################################################################
data3_200.to_csv("rmse lambda for complexity = 3, train size = 200")


################################################################
plt.figure()
plt.plot(data3_200["Lambda"],data3_200["RMSE Train"],label="RMSE Train")
plt.plot(data3_200["Lambda"],data3_200["RMSE CV"],label="RMSE CV")
plt.plot(data3_200["Lambda"],data3_200["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 3...
    ")
plt.savefig("d3_200.png")
plt.show()

################################################################
# ### Surface Plots for various values of lambda:

################################################################
for l in lambda_list:
```

```
    plot_approxY(3, w_3_200,y3_train200,y3_trainpred200[l],200,l,...
        x13_train200,x23_train200)

##################################################################
# ## 2.10 For train data size = 500 and degree of complexity = 3

##################################################################
data3_train500,data3_cv500,data3_test500=create_datasets(func2d,500,200)
x13_train500,x23_train500,y3_train500,x13_cv500,x23_cv500,y3_cv500,...
    x13_test500,x23_test500,y3_test500=split_cols(data3_train500,...
    data3_cv500,data3_test500)

### design matrix:
phi3_train500=create_phi(3,len(y3_train500),x13_train500,x23_train500)
phi3_cv500=create_phi(3,len(y3_cv500),x13_cv500,x23_cv500)
phi3_test500=create_phi(3,len(y3_test500),x13_test500,x23_test500)

y3_testpred500={}
y3_trainpred500={}
y3_cvpred500={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse3_train500=[]
rmse3_test500=[]
rmse3_cv500=[]

for l in lambda_list:
    w_3_500=opt_regularized_param(l,phi3_train500,y3_train500);
    y3_trainpred500[l]=y_pred(phi3_train500,w_3_500)
    y3_testpred500[l]=y_pred(phi3_test500,w_3_500);
    y3_cvpred500[l]=y_pred(phi3_cv500,w_3_500);
    rmse3_train500.append(RMSE(y3_trainpred500[l],y3_train500))
    rmse3_test500.append(RMSE(y3_testpred500[l],y3_test500))
    rmse3_cv500.append(RMSE(y3_cvpred500[l],y3_cv500))



##################################################################
data3_500=pd.DataFrame(list(zip(lambda_list,rmse3_train500,rmse3_cv500,...
    rmse3_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


##################################################################
data3_500.to_csv("rmse lambda for complexity = 3 train size = 500")


##################################################################
plt.figure()
plt.plot(data3_500["Lambda"],data3_500["RMSE Train"],label="RMSE Train")
plt.plot(data3_500["Lambda"],data3_500["RMSE CV"],label="RMSE CV")
plt.plot(data3_500["Lambda"],data3_500["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 3...
    ")
plt.savefig("d3_500.png")
plt.show()

##################################################################
# ### Surface Plots for various values of lambda:
```

```
########################################################################
for l in lambda_list:
    plot_approxY(3, w_3_500,y3_train500,y3_trainpred500[l],500,l,...
        x13_train500,x23_train500)

########################################################################
# ## 2.11 Degree of complexity = 6 and train data size = 50

########################################################################
data6_train50,data6_cv50,data6_test50=create_datasets(func2d,50,30)
x16_train50,x26_train50,y6_train50,x16_cv50,x26_cv50,y6_cv50,x16_test50,...
    x26_test50,y6_test50=split_cols(data6_train50,data6_cv50,data6_test50)

### design matrix:
phi6_train50=create_phi(6,len(y6_train50),x16_train50,x26_train50)
phi6_cv50=create_phi(6,len(y6_cv50),x16_cv50,x26_cv50)
phi6_test50=create_phi(6,len(y6_test50),x16_test50,x26_test50)

y6_testpred50={}
y6_trainpred50={}
y6_cvpred50={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse6_train50=[]
rmse6_test50=[]
rmse6_cv50=[]

for l in lambda_list:
    w_6_50=opt_regularized_param(l,phi6_train50,y6_train50);
    y6_trainpred50[l]=y_pred(phi6_train50,w_6_50)
    y6_testpred50[l]=y_pred(phi6_test50,w_6_50);
    y6_cvpred50[l]=y_pred(phi6_cv50,w_6_50);
    rmse6_train50.append(RMSE(y6_trainpred50[l],y6_train50))
    rmse6_test50.append(RMSE(y6_testpred50[l],y6_test50))
    rmse6_cv50.append(RMSE(y6_cvpred50[l],y6_cv50))



########################################################################
data6_50=pd.DataFrame(list(zip(lambda_list,rmse6_train50,rmse6_cv50,...
    rmse6_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


########################################################################
data6_50.to_csv("rmse lambda complexity= 6, train = 50")


########################################################################
plt.figure()
plt.plot(data6_50["Lambda"],data6_50["RMSE Train"],label="RMSE Train")
plt.plot(data6_50["Lambda"],data6_50["RMSE CV"],label="RMSE CV")
plt.plot(data6_50["Lambda"],data6_50["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 6"...
    )
plt.savefig("d6_50.png")
plt.show()

```

```
##################################################################
# ### Surface plots for various values of lambda:

##################################################################
for l in lambda_list:
    plot_approxY(6, w_6_50,y6_train50,y6_trainpred50[l],50,l,x16_train50,...
        x26_train50)

##################################################################
# ## 2.12 For degree of complexity = 6 and train data size = 200

##################################################################
data6_train200,data6_cv200,data6_test200=create_datasets(func2d,200,90)
x16_train200,x26_train200,y6_train200,x16_cv200,x26_cv200,y6_cv200,...
    x16_test200,x26_test200,y6_test200=split_cols(data6_train200,...
    data6_cv200,data6_test200)

### design matrix:
phi6_train200=create_phi(6,len(y6_train200),x16_train200,x26_train200)
phi6_cv200=create_phi(6,len(y6_cv200),x16_cv200,x26_cv200)
phi6_test200=create_phi(6,len(y6_test200),x16_test200,x26_test200)

y6_testpred200={}
y6_trainpred200={}
y6_cvpred200={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse6_train200=[]
rmse6_test200=[]
rmse6_cv200=[]

for l in lambda_list:
    w_6_200=opt_regularized_param(l,phi6_train200,y6_train200);
    y6_trainpred200[l]=y_pred(phi6_train200,w_6_200)
    y6_testpred200[l]=y_pred(phi6_test200,w_6_200);
    y6_cvpred200[l]=y_pred(phi6_cv200,w_6_200);
    rmse6_train200.append(RMSE(y6_trainpred200[l],y6_train200))
    rmse6_test200.append(RMSE(y6_testpred200[l],y6_test200))
    rmse6_cv200.append(RMSE(y6_cvpred200[l],y6_cv200))



##################################################################
data6_200=pd.DataFrame(list(zip(lambda_list,rmse6_train200,rmse6_cv200,...
    rmse6_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


##################################################################
data6_200.to_csv("rmse lambda complexity = 6 train = 200")


##################################################################
plt.figure()
plt.plot(data6_200["Lambda"],data6_200["RMSE Train"],label="RMSE Train")
plt.plot(data6_200["Lambda"],data6_200["RMSE CV"],label="RMSE CV")
plt.plot(data6_200["Lambda"],data6_200["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
plt.legend()
plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 6...
    ")
```

```python
plt.savefig("d6_200.png")
plt.show()

##################################################################
# ### Surface plots for various values of lambda:

##################################################################
for l in lambda_list:
    plot_approxY(6, w_6_200,y6_train200,y6_trainpred200[l],200,l,...
        x16_train200,x26_train200)

##################################################################
# ## 2.13 For degree of complexity = 6 and train data size = 500

##################################################################
data6_train500,data6_cv500,data6_test500=create_datasets(func2d,500,200)
x16_train500,x26_train500,y6_train500,x16_cv500,x26_cv500,y6_cv500,...
    x16_test500,x26_test500,y6_test500=split_cols(data6_train500,...
    data6_cv500,data6_test500)

### design matrix:
phi6_train500=create_phi(6,len(y6_train500),x16_train500,x26_train500)
phi6_cv500=create_phi(6,len(y6_cv500),x16_cv500,x26_cv500)
phi6_test500=create_phi(6,len(y6_test500),x16_test500,x26_test500)

y6_testpred500={}
y6_trainpred500={}
y6_cvpred500={}
lambda_list=[0,0.5,1,2,10,50,100]
rmse6_train500=[]
rmse6_test500=[]
rmse6_cv500=[]

for l in lambda_list:
    w_6_500=opt_regularized_param(l,phi6_train500,y6_train500);
    y6_trainpred500[l]=y_pred(phi6_train500,w_6_500)
    y6_testpred500[l]=y_pred(phi6_test500,w_6_500);
    y6_cvpred500[l]=y_pred(phi6_cv500,w_6_500);
    rmse6_train500.append(RMSE(y6_trainpred500[l],y6_train500))
    rmse6_test500.append(RMSE(y6_testpred500[l],y6_test500))
    rmse6_cv500.append(RMSE(y6_cvpred500[l],y6_cv500))



##################################################################
data6_500=pd.DataFrame(list(zip(lambda_list,rmse6_train500,rmse6_cv500,...
    rmse6_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])


##################################################################
data6_500.to_csv("rmse lambda complexity = 6 train = 500")


##################################################################
plt.figure()
plt.plot(data6_500["Lambda"],data6_500["RMSE Train"],label="RMSE Train")
plt.plot(data6_500["Lambda"],data6_500["RMSE CV"],label="RMSE CV")
plt.plot(data6_500["Lambda"],data6_500["RMSE test"],label="RMSE test")
plt.xlabel("Lambda->")
plt.ylabel("RMSE")
```

```
plt.legend()
plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 6...
    ")
plt.savefig("d6_500.png")
plt.show()


###################################################################
data6_500

###################################################################
# ### Surface plots for various values of lambda:

###################################################################
for l in lambda_list:
    plot_approxY(6, w_6_500,y6_train500,y6_trainpred500[l],500,l,...
        x16_train500,x26_train500)

###################################################################
# # Conclusion:
#
# The model with best RMSE values over all three datasets: train, cross-...
    validation and test is with degree of complexity 6, train data size = ...
    500 and regularization parameter lambda = 0

###################################################################
plt.figure()
plt.scatter(y6_train500,y6_trainpred500[0],label="predicted y")
plt.legend()
plt.xlabel("Actual y")
plt.ylabel("Predicted y")
plt.title("Predicted output vs actual output for train data set")
plt.savefig("predgoodtrain.png")
plt.show()


###################################################################
plt.figure()
plt.scatter(y6_cv500,y6_cvpred500[0],label="predicted y")
plt.legend()
plt.xlabel("Actual y")
plt.ylabel("Predicted y")
plt.title("Predicted output vs actual output for cross-validation data set"...
    )
plt.savefig("predcv.png")
plt.show()


###################################################################
plt.figure()
plt.scatter(y6_test500,y6_testpred500[0],label="predicted y")
plt.legend()
plt.xlabel("Actual y")
plt.ylabel("Predicted y")
plt.title("Predicted output vs actual output for test data set")
plt.savefig("predtest.png")
plt.show()


###################################################################
```

```
### rmse train:
rmse6_train500[0]


#################################################################
### rmse cross validation:
rmse6_cv500[0]


#################################################################
### rmse test data:
rmse6_test500[0]


#################################################################
plt.figure()
plt.scatter(x16_train500,y6_train500)


#################################################################
plt.figure()
plt.scatter(x26_train500,y6_train500)


#################################################################
plt.figure()
plt.scatter(y6_trainpred500[0]-y6_train500)


#################################################################
```

3 Task 3

3.1 No Regularization, L2 Regularization

The code for Question 3, No Regularization and L2 Regularization is as follows:

```
#################################################################
# ## CS5691 PRML Assignment 1
# **Team 1**
# **Team Members:**
# N Sowmya Manojna   BE17B007
# Thakkar Riya Anandbhai  PH17B010
# Chaithanya Krishna Moorthy  PH17B011

#################################################################
# Install required Packages
# Uncomment if you are running for the first time
# !pip install -r requirements.txt
# try:
#     !mkdir images/q3
# except:
#     pass

#################################################################
import os
import missingno
import numpy as np
```

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from preprocess import PreProcess
from gaussianbasis import GaussianBasis
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from statsmodels.stats.outliers_influence import variance_inflation_factor

####################################################################
print("Reading Dataset... ", end="")
df = pd.read_csv("../datasets/1_bias_clean.csv")
print("Done!")

print("Starting Preprocessing... ")
preprocess = PreProcess()
preprocess.clean(df, verbose=True)
print("="*60)
print("="*60)
print("Preprocessing Done!!")

df_save = pd.read_csv("../datasets/processed.csv", index_col=0)
df_new = df_save.copy()
df_new = df_new.drop(["Next_Tmax", "Next_Tmin"], axis=1)

num_clusters = [1]
num_clusters.extend(range(2,10))
num_clusters.extend(range(15, 31, 5))
num_clusters.extend(range(40, 101, 10))

print("Starting Regularization... ", end="")
lmbda_list = [0, 0.5, 1, 2, 10, 50, 100]
regressor = GaussianBasis()
output = regressor.fit_grid(df_new, df_save, num_clusters, regularization="L2", lmbda_list=lmbda_list, verbose=True, show=False)
df_result, sse_dict, num_clusters = output


df_result["Sum Error"] = df_result["Error 1"] + df_result["Error 2"]
df_result.sort_values(by=["Sum Error"], inplace=True)
print(df_result)

sse_list = [sse_dict[i] for i in sse_dict]
plt.figure(figsize=[12,8])
plt.plot(num_clusters,sse_list )
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Knee Plot for determining the number of clusters")
plt.grid()
plt.show()

plt.figure(figsize=[12,8])
plt.plot(num_clusters[:5], sse_list[:5])
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Knee Plot for determining the number of clusters")
plt.grid()
plt.show()
```

3.1.1 Pre-Processing

The helper class used to perform `PreProcessing` is as follows:

```
import missingno
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from statsmodels.stats.outliers_influence import variance_inflation_factor

##################################################################
class PreProcess():
    def __init__(self):
        pass

    def clean(self, df, verbose=False):
        if verbose==True:
            print("="*60)
        if verbose==True:
            print("Sample of Dataset")
        if verbose==True:
            print(df.head())

        if verbose==True:
            print("="*60)
        if verbose==True:
            print("Information of Dataset")
        df.info()

        if verbose==True:
            print("="*60)
        if verbose==True:
            print("'Nan' Distribution:")
        if verbose==True:
            print(df.isnull().sum())

        if verbose==True:
            print("="*60)
        if verbose==True:
            print("Description of the dataset:")
        if verbose==True:
            print(df.describe())

        plt.figure()
        missingno.matrix(df)
        plt.title("Missing Values Visualization")
        plt.show()

        if verbose==True:
            print("Removing the Rows with NaNs")
        df_clean = df.dropna(axis=0)
        plt.figure()
        missingno.matrix(df_clean)
        plt.title("Missing Values Visualization - After NaN removal")

        df = df_clean
        desc = df.describe()
        sum(desc.loc["std"] == 0)
```

```
    # sns.pairplot(df)

    plt.figure(figsize=[15,15])
    sns.heatmap(df.corr().round(2), linewidths=.5, annot=True)
    plt.title("Heatmap of the data")
    plt.show()

    if verbose==True:
        print("="*60)
    if verbose==True:
        print("Identifying highly correlated features... ")
    # Use the upper triangle to mask the correlation matrix
    df_new = df.copy()
    df_new = df_new.drop(["Next_Tmin", "Next_Tmax"], axis=1)
    upper_triangle = np.triu(np.ones(df_new.corr().shape)).astype(bool)

    # Get the correlation pairs
    correlation_pairs = df_new.corr().mask(upper_triangle).abs()....
        unstack().sort_values(ascending=False)
    correlation_pairs = pd.DataFrame(correlation_pairs)
    if verbose==True:
        print("Correlation between Features")
    if verbose==True:
        print(correlation_pairs.head(25))
    if verbose==True:
        print("="*50)

    # if verbose==True:
    #   print the highly correlated features
    highly_correlated = correlation_pairs[correlation_pairs[0]>0.75]
    if verbose==True:
        print("Highly Correlated Features")
    if verbose==True:
        print(highly_correlated)
    if verbose==True:
        print("="*50)

    # Remove each of the highly correlated features and check
    highly_correlated.reset_index(inplace=True)
    hc_features = highly_correlated["level_0"]

    df_new.head()
    if verbose==True:
        print("Done!")

    # Remove highly correlated features
    for feature in hc_features:
        if verbose==True:
            print("Removing:", feature, "...")
        df_new = df_new.drop([feature], axis=1)
        upper_triangle = np.triu(np.ones(df_new.corr().shape)).astype(...
            bool)
        c = df_new.corr().mask(upper_triangle).abs().unstack()....
            sort_values(ascending=False)
        c = pd.DataFrame(c)
        if verbose==True:
            print(c[c[0]>0.75])
        if verbose==True:
            print("="*50, "\n")
```

```
            if c[c[0]>0.75].size == 0:
                if verbose==True:
                    print("The number of highly  correlated fetaures has ...
                        become zero!")
                if verbose==True:
                    print("Preventing all further removals :)")
                break

        df_new.head()

        if verbose==True:
            print("="*60)
        if verbose==True:
            print("Checking for Variance Inflation Factor... ")
        # Variance Inflation Factor - directly correlated
        X_df = df_new.copy()

        vif = pd.DataFrame()
        vif["Features"] = X_df.columns
        vif["VIF Factor"]=[variance_inflation_factor(X_df.values, i) for i ...
            in range(X_df.shape[1])]
        vif.sort_values(by=["VIF Factor"], ascending=False, inplace=True)

        max_col = vif[vif["VIF Factor"]>1000]
        for feature in max_col["Features"]:
            max_col = vif[vif["VIF Factor"]>1000]
            if verbose==True:
                print("Features with high VIF:")
            if verbose==True:
                print(max_col)
            if verbose==True:
                print("Dropping", feature, "...")
            X_df.drop([feature], axis=1, inplace=True)

            vif = pd.DataFrame()
            vif["Features"] = X_df.columns
            vif["VIF Factor"]=[variance_inflation_factor(X_df.values, i) ...
                for i in range(X_df.shape[1])]
            vif.sort_values(by=["VIF Factor"], ascending=False, inplace=...
                True)

            if vif[vif["VIF Factor"]>1000].size==0:
                if verbose==True:
                    print("The number of fetaures that have high VIF has ...
                        become zero!")
                if verbose==True:
                    print("Preventing all further removals :)")
                break
        if verbose==True:
            print("Done!")

        df_new = X_df
        df_new.head()

        df_save = df_new.copy()
        df_save["Next_Tmin"] = df["Next_Tmin"]
        df_save["Next_Tmax"] = df["Next_Tmax"]
        df_save.to_csv("../datasets/processed.csv")
        # df_save.to_csv("processed.csv")
        if verbose==True:
```

```
            print("Saved the data as processed.csv")

        return 1
```

3.1.2 Gaussian Basis

The helper class used to perform `GaussianBasis` are as follows:

```
import os
import missingno
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
from preprocess import PreProcess
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.cluster import KMeans

class GaussianBasis():
    def __init__(self):
        pass

    def fit_grid(self, df_new, df_save, num_clusters, regularization=None, ...
       lmbda_list=[0], verbose=False, show=False):
        n_clu_hist = []
        lmbda_hist = []
        error1_hist = []
        error2_hist = []
        sse_dict = {}

        if regularization == None or regularization=="L2":
            print("Running K-Means...", end="")
            for n_clu in tqdm(num_clusters):
                kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(...
                    df_new.to_numpy())
                sse_dict[n_clu] = kmeans.inertia_

                mean_centers = kmeans.cluster_centers_
                corresponding_center = mean_centers[kmeans.labels_,:]

                X = df_new.to_numpy()
                distance = np.linalg.norm(X-corresponding_center, axis=1)
                var = np.var(distance)*distance.size

                phi = np.ones((X.shape[0], 1))
                for i in range(n_clu):
                    phi = np.append(phi, np.exp(-np.linalg.norm(X-...
                        mean_centers[i,:], axis=1)**2/var).reshape(-1,1), ...
                        axis=1)

                for lmbda in lmbda_list:
                    n_clu_hist.append(n_clu)
                    lmbda_hist.append(lmbda)

                    W1 = (np.linalg.inv(phi.T @ phi + lmbda*np.identity(phi...
                        .shape[1])) @ phi.T) @ df_save["Next_Tmin"]
```

```
                    W2 = (np.linalg.inv(phi.T @ phi + lmbda*np.identity(phi.shape[1])) @ phi.T) @ df_save["Next_Tmax"]
                    W1 = W1.reshape(-1,1)
                    W1 = W2.reshape(-1,1)
                    pred1 = phi @ W1
                    pred2 = phi @ W2

                    plt.figure(figsize=[16,9])
                    plt.title("Clusters: "+str(n_clu))
                    plt.subplot(1, 2, 1)
                    plt.hist(pred1, alpha=0.5)
                    plt.hist(df_save["Next_Tmin"], alpha=0.5)
                    plt.title("Next_Tmin; Clusters: "+str(n_clu))
                    plt.grid()
                    plt.subplot(1, 2, 2)
                    plt.plot(df_save["Next_Tmin"], pred1, ".")
                    plt.plot(df_save["Next_Tmin"], df_save["Next_Tmin"], '.')
                    plt.title("Next_Tmin; Clusters: "+str(n_clu))
                    plt.grid()
                    if show:
                        plt.show()
                    fname = "fit_1_k_"+str(n_clu)+"_lmbda_"+str(lmbda)+".png"
                    plt.savefig("images/q3/"+fname)

                    plt.figure(figsize=[16,9])
                    plt.title("Clusters: "+str(n_clu))
                    plt.subplot(1, 2, 1)
                    plt.hist(pred2, alpha=0.5, label="Predicted")
                    plt.hist(df_save["Next_Tmax"], alpha=0.5, label="True")
                    plt.title("Next_Tmax; Clusters: "+str(n_clu))
                    plt.grid()
                    plt.subplot(1, 2, 2)
                    plt.plot(df_save["Next_Tmax"], pred2, ".")
                    plt.plot(df_save["Next_Tmax"], df_save["Next_Tmax"], '.')
                    plt.title("Clusters: "+str(n_clu))
                    plt.grid()
                    if show:
                        plt.show()
                    fname = "fit_2_k_"+str(num_clusters)+"_lmbda_"+str(lmbda)+".png"
                    plt.savefig("images/q3/"+fname)

                    error1 = np.linalg.norm(df_save["Next_Tmin"].to_numpy().reshape(-1,1)-pred1)
                    error2 = np.linalg.norm(df_save["Next_Tmin"].to_numpy().reshape(-1,1)-pred2)

                    error1_hist.append(error1)
                    error2_hist.append(error2)
                print("Done!")

        df_result = pd.DataFrame()
        df_result["Cluster"] = n_clu_hist
        df_result["Lambda"] = lmbda_hist
        df_result["Error 1"] = error1_hist
        df_result["Error 2"] = error2_hist
        df_result.to_csv("../datasets/q3_gridsearch.csv")
```

```
        return df_result, sse_dict
```

3.2 Tikhonov Regularization

The code for Question 3, Tikhonov Regularization is as follows:

```
############################################################################
from IPython import get_ipython
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')


############################################################################
dataset2 = pd.read_csv('function1_2d.csv',index_col = 0)


############################################################################
dataset2['y']


############################################################################
num_clusters = [1]
num_clusters.extend(range(2,100,10))
#num_clusters.extend(range(15, 31, 5))
#num_clusters.extend(range(40, 101, 10))

sse_list = []
label_list = []
cluster_centers_list = []
error_list = []

for n_clu in num_clusters:
    kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(dataset2....
        to_numpy())
    sse_list.append(kmeans.inertia_)
    label_list.append(kmeans.labels_)
    cluster_centers_list.append(kmeans.cluster_centers_)

    mean_centers = cluster_centers_list[-1]
  # print("Mean shape:", mean_centers.shape)
    corresponding_center = mean_centers[label_list[-1],:]

    X = dataset2.to_numpy()
    distance = np.linalg.norm(X-corresponding_center, axis=1)
    var = np.var(distance)*distance.size

    phi = np.ones((X.shape[0], 1))
    for i in range(n_clu):
        A = X-mean_centers[i,:]
    # print("A shape:", A.shape)
        A = np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var)
    # print("A shape:", A.shape)
```

```
        phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,:], ...
            axis=1)**2/var).reshape(-1,1), axis=1)

    lmbda = 0
    W1 = (np.linalg.inv(phi.T @ phi + lmbda*np.identity(phi.shape[1])) @ ...
        phi.T) @ dataset2["y"]
    W1 = W1.reshape(-1,1)
    pred = phi @ W1

    plt.figure(figsize=[12,8])
    plt.title("Clusters: "+str(n_clu))
    plt.subplot(1, 2, 1)
    plt.hist(pred, alpha=0.5)
    plt.hist(dataset2["y"], alpha=0.5)
    plt.title("Clusters: "+str(n_clu))
    plt.grid()
    plt.subplot(1, 2, 2)
    plt.plot(dataset2["y"], pred, ".")
    plt.plot(dataset2["y"], dataset2["y"], '.')
    plt.title("Clusters: "+str(n_clu))
    plt.grid()
    plt.savefig('')
    plt.show()
    error = np.linalg.norm(dataset2["y"].to_numpy().reshape(-1,1)-pred)
    error_list.append(error)


###########################################################################
plt.figure(figsize=[12,8])
plt.subplot(1,2,1)
plt.plot(num_clusters, sse_list)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Knee Plot for determining the number of clusters")
plt.grid()
plt.subplot(1,2,2)
plt.plot(num_clusters, error_list)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title(("L2 Error for fit"))
plt.grid()
plt.show()


###########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"Clusters":num_clusters, "Error":error_list})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


###########################################################################
def create_datasets(data,train_size,cv_size):
    data.sample(frac=1).reset_index(drop=True)
    data_train=data[0:train_size]
    data_cv=data[train_size:train_size+cv_size]
    data_test=data[cv_size+train_size:]
    return(data_train,data_cv,data_test)


```

```

###########################################################################
lambda_list = [0.01,0.1,1,5,10,50,100]
trDS2, cvDS2, tDS2 = create_datasets(dataset2,1400,400)


###########################################################################
def f(ds,l,n_clu):
    sse_list = []
    label_list = []
    cluster_centers_list = []

    kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(ds.to_numpy())
    sse_list.append(kmeans.inertia_)
    label_list.append(kmeans.labels_)
    cluster_centers_list.append(kmeans.cluster_centers_)

    mean_centers = cluster_centers_list[-1]
  # print("Mean shape:", mean_centers.shape)
    corresponding_center = mean_centers[label_list[-1],:]

    X = ds.to_numpy()
    distance = np.linalg.norm(X-corresponding_center, axis=1)
    var = np.var(distance)*distance.size

    phi = np.ones((X.shape[0], 1))
    for i in range(n_clu):
        A = X-mean_centers[i,:]
    # print("A shape:", A.shape)
        A = np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var)
    # print("A shape:", A.shape)
        phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var).reshape(-1,1), axis=1)
    #lmbda = 1
    return(phi,mean_centers,var)

########################################################################### [markdown]
#     W1 = (np.linalg.inv(phi.T @ phi + l*np.identity(phi.shape[1])) @ phi.T) @ dataset2["y"]
#     W1 = W1.reshape(-1,1)
#     pred = phi @ W1
#     return(W1,)
#     error = np.linalg.norm(dataset2["y"].to_numpy().reshape(-1,1)-pred)
#     error_list.append(error)

###########################################################################
optClds2 = 10


###########################################################################
error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
    phi_tr = f(trDS2,l,optClds2)[0]
    phi_cv = f(cvDS2,l,optClds2)[0]
    phi_t = f(tDS2,l,optClds2)[0]
    w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ phi_tr.T) @ trDS2["y"]
```

```
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS2["y"].to_numpy().reshape(-1,1)-...
        pred_cv))
    error_t.append(np.linalg.norm(tDS2["y"].to_numpy().reshape(-1,1)-pred_t...
        ))
    error_tr.append(np.linalg.norm(trDS2["y"].to_numpy().reshape(-1,1)-...
        pred_tr))


###########################################################################
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


###########################################################################
plt.plot(lambda_list,error_cv)


###########################################################################
phi_tr = f(trDS2,l,optClds2)[0]
phi_t = f(tDS2,l,optClds2)[0]
w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.01*np.identity(phi_tr.shape[1])) @...
     phi_tr.T) @ trDS2["y"]
pred_t = phi_t @ w
pred_tr = phi_tr @ w


###########################################################################
plt.scatter(trDS2.iloc[:,2],pred_tr)
plt.xlabel("Target output,training data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output for linear regression in ...
    gaussian basis and quadratic regularization")
plt.savefig("scatter_ds2quad.png")
plt.show()


###########################################################################
plt.scatter(cvDS2.iloc[:,2],pred_cv)


###########################################################################
plt.scatter(tDS2.iloc[:,2],pred_t)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output for linear regression in ...
    gaussian basis and quadratic regularization")
plt.savefig("scatter_ds2quadtest.png")
plt.show()


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



###########################################################################
def tikhanov_reg(phi,mu,sigma,l):
    K = len(mu)
```

```
    phiT = np.zeros((K+1,K+1))
    phiT[0,0] = 1
    for i in range(1,K+1):
        for j in range(1,K+1):
            phiT[i,j] = np.exp(-(np.linalg.norm(mu[i-1]-mu[j-1])**2)/sigma...
                **2)
    #l = 300
    #print(phiT.shape)
    pinv = np.linalg.inv(phi.T @ phi+l*phiT) @ phi.T
    return(pinv)


###########################################################################
lambda_list = [0,0.01,0.1,1,5]#,10,50,100]


###########################################################################
error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
#for l in [1]:
    phi_tr,mu_list,sig = f(trDS2,l,optClds2)
    phi_cv = f(cvDS2,l,optClds2)[0]
    phi_t = f(tDS2,l,optClds2)[0]
    #print(phi_tr.shape)
    #print(mu_list.shape)
    tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
    #print("tikh shape ", tikh.shape)
    #print("phi_tr shape ", phi_tr.shape)
    #print("trDS2_y shape ", trDS2["y"].shape)

    w = tikh @ trDS2["y"]
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS2["y"].to_numpy().reshape(-1,1)-...
        pred_cv))
    error_t.append(np.linalg.norm(tDS2["y"].to_numpy().reshape(-1,1)-pred_t...
        ))
    error_tr.append(np.linalg.norm(trDS2["y"].to_numpy().reshape(-1,1)-...
        pred_tr))


###########################################################################
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


###########################################################################
plt.plot(lambda_list,error_cv)
plt.title('Erms on cross-validation data vs regularization factor')
plt.xlabel('lambda')
plt.ylabel('rmse error')


###########################################################################
plt.hist(pred_tr, alpha=0.5)
plt.hist(trDS2["y"], alpha=0.5)

```

```
##########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


##########################################################################
l = 0.01
phi_tr = f(trDS2,l,optClds2)[0]
phi_t = f(tDS2,l,optClds2)[0]
tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
w = tikh @ trDS2["y"]
w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS2["y"]
pred_t = phi_t @ w
pred_tr = phi_tr @ w


##########################################################################
plt.scatter(trDS2.iloc[:,2],pred_tr)
plt.xlabel("Target output,training data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output for linear regression in ...
    gaussian basis and Tikhonov regularization")
plt.savefig("scatter_ds2tikhtr.png")
plt.show()


##########################################################################
plt.scatter(tDS2.iloc[:,2],pred_t)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output for linear regression in ...
    gaussian basis and Tikhonov regularization")
plt.savefig("scatter_ds2tikhtest.png")
plt.show()


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



##########################################################################
df_new = pd.read_csv("processed_biasclean.csv",index_col = 0)


##########################################################################
df_new.head()


##########################################################################
trDS3, cvDS3, tDS3 = create_datasets(df_new,1400,400)


##########################################################################
num_clusters = [1]
num_clusters.extend(range(2,10))
num_clusters.extend(range(15, 31, 5))
```

```
num_clusters.extend(range(40, 101, 10))

sse_list = []
label_list = []
cluster_centers_list = []
error_list = []

for n_clu in num_clusters:
    kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(df_new.to_numpy...
        ())
    sse_list.append(kmeans.inertia_)
    label_list.append(kmeans.labels_)
    cluster_centers_list.append(kmeans.cluster_centers_)

    mean_centers = cluster_centers_list[-1]
  # print("Mean shape:", mean_centers.shape)
    corresponding_center = mean_centers[label_list[-1],:]

    X = df_new.to_numpy()
    distance = np.linalg.norm(X-corresponding_center, axis=1)
    var = np.var(distance)*distance.size

    phi = np.ones((X.shape[0], 1))
    for i in range(n_clu):
        A = X-mean_centers[i,:]
    # print("A shape:", A.shape)
        A = np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var)
    # print("A shape:", A.shape)
        phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,:], ...
            axis=1)**2/var).reshape(-1,1), axis=1)

    lmbda = 0
    W1 = (np.linalg.inv(phi.T @ phi + lmbda*np.identity(phi.shape[1])) @ ...
        phi.T) @ df_new["Next_Tmin"]
    W1 = W1.reshape(-1,1)
    pred = phi @ W1

    plt.figure(figsize=[12,8])
    plt.title("Clusters: "+str(n_clu))
    plt.subplot(1, 2, 1)
    plt.hist(pred, alpha=0.5)
    plt.hist(df_new["Next_Tmin"], alpha=0.5)
    plt.title("Clusters: "+str(n_clu))
    plt.grid()
    plt.subplot(1, 2, 2)
    plt.plot(df_new["Next_Tmin"], pred, ".")
    plt.plot(df_new["Next_Tmin"], df_new["Next_Tmin"], '.')
    plt.title("Clusters: "+str(n_clu))
    plt.grid()
    plt.show()
    error = np.linalg.norm(df_new["Next_Tmin"].to_numpy().reshape(-1,1)-...
        pred)
    error_list.append(error)


##########################################################################
plt.figure(figsize=[12,8])
plt.subplot(1,2,1)
plt.plot(num_clusters, sse_list)
plt.xlabel("Number of Clusters")
```

```
plt.ylabel("SSE")
plt.title("Knee Plot for determining the number of clusters")
plt.grid()
plt.subplot(1,2,2)
plt.plot(num_clusters, error_list)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title(("L2 Error for fit"))
plt.grid()
plt.show()


########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"Clusters":num_clusters, "Error":error_list})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


########################################################################
optClds3 = 9


########################################################################
lambda_list = [0.01,0.1,1,5,10]


########################################################################
error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
    phi_tr = f(trDS3,l,optClds3)[0]
    phi_cv = f(cvDS3,l,optClds3)[0]
    phi_t = f(tDS3,l,optClds3)[0]
    w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) ...
        @ phi_tr.T) @ trDS3['Next_Tmin']
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_cv))
    error_t.append(np.linalg.norm(tDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_t))
    error_tr.append(np.linalg.norm(trDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_tr))


########################################################################
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


########################################################################
plt.plot(lambda_list,error_cv)
plt.title('Error on cross-validation data vs regularization factor, ...
    Quadratic regularization')
plt.xlabel('lambda')
plt.ylabel('rmse error')

```

```
##########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


##########################################################################
l = .1
phi_tr = f(trDS3,l,optClds3)[0]
phi_t = f(tDS3,l,optClds3)[0]
w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS3['Next_Tmin']
pred_t = phi_t @ w
pred_tr = phi_tr @ w


##########################################################################
plt.scatter(trDS3.iloc[:,17],pred_tr)
plt.xlabel("Target output,train data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model with quadratic regularization, ...
    for Next_Tmin")
plt.savefig("scatter_ds3quadtrainT_min.png")
plt.show()


##########################################################################
plt.scatter(tDS3.iloc[:,17],pred_t)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output with quadratic ...
    regularization, for Next_Tmin")
plt.savefig("scatter_ds3quadtestT_min.png")
plt.show()


##########################################################################
error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
    phi_tr = f(trDS3,l,optClds3)[0]
    phi_cv = f(cvDS3,l,optClds3)[0]
    phi_t = f(tDS3,l,optClds3)[0]
    w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) ...
        @ phi_tr.T) @ trDS3['Next_Tmax']
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_cv))
    error_t.append(np.linalg.norm(tDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_t))
    error_tr.append(np.linalg.norm(trDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_tr))


##########################################################################
```

```
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


##########################################################################
plt.plot(lambda_list,error_cv)
plt.title('Error on cross-validation data vs regularization factor, ...
    Quadratic regularization')
plt.xlabel('lambda')
plt.ylabel('rmse error')


##########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


##########################################################################
l = 1
phi_tr = f(trDS3,l,optClds3)[0]
phi_t = f(tDS3,l,optClds3)[0]
w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS3['Next_Tmax']
pred_t = phi_t @ w
pred_tr = phi_tr @ w


##########################################################################
plt.scatter(trDS3.iloc[:,18],pred_tr)
plt.xlabel("Target output,train data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model with quadratic regularization, ...
    for Next_Tmax")
plt.savefig("scatter_ds3quadtrainT_max.png")
plt.show()


##########################################################################
plt.scatter(tDS3.iloc[:,18],pred_t)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model with quadratic regularization, ...
    for Next_Tmax")
plt.savefig("scatter_ds3quadtestT_max.png")
plt.show()


##########################################################################
#Tikhonov reg for "Next_Tmin"

error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
#for l in [1]:
    phi_tr,mu_list,sig = f(trDS3,l,optClds3)
    phi_cv = f(cvDS2,l,optClds3)[0]
    phi_t = f(tDS2,l,optClds3)[0]
```

```
    #print(phi_tr.shape)
    #print(mu_list.shape)
    tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
    #print("tikh shape ", tikh.shape)
    #print("phi_tr shape ", phi_tr.shape)
    #print("trDS2_y shape ", trDS2["y"].shape)

    w = tikh @ trDS3['Next_Tmin']
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_cv))
    error_t.append(np.linalg.norm(tDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_t))
    error_tr.append(np.linalg.norm(trDS3['Next_Tmin'].to_numpy().reshape...
        (-1,1)-pred_tr))


############################################################################
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


############################################################################
plt.plot(lambda_list,error_cv)
plt.title('Error on cross-validation data vs regularization factor, ...
    tikhonov regularization for Dataset 3')
plt.xlabel('lambda')
plt.ylabel('rmse error')


############################################################################
plt.hist(pred_tr, alpha=0.5)
plt.hist(trDS3["Next_Tmin"], alpha=0.5)


############################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


############################################################################
l = .1
phi_tr = f(trDS3,l,optClds3)[0]
phi_t = f(tDS3,l,optClds3)[0]
tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
w = tikh @ trDS3["Next_Tmin"]
w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS3["Next_Tmin"]
pred_t = phi_t @ w
pred_tr = phi_tr @ w


############################################################################
plt.scatter(trDS3.iloc[:,17],pred_tr)
plt.xlabel("Target output,train data")
plt.ylabel("Model output")
```

```
plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
    for Next_Tmin")
plt.savefig("scatter_ds3tikhtrainT_min.png")
plt.show()


########################################################################
plt.scatter(tDS3.iloc[:,17],pred_t)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
    for Next_Tmin")
plt.savefig("scatter_ds3tikhtrainT_min.png")
plt.show()


########################################################################
#Tikhonov reg for "Next_Tmax"
error_tr = []
error_cv = []
error_t = []
for l in range(len(lambda_list)):
#for l in [1]:
    phi_tr,mu_list,sig = f(trDS3,l,optClds3)
    phi_cv = f(cvDS2,l,optClds3)[0]
    phi_t = f(tDS2,l,optClds3)[0]
    #print(phi_tr.shape)
    #print(mu_list.shape)
    n = len(trDS3)
    tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
    #print("tikh shape ", tikh.shape)
    #print("phi_tr shape ", phi_tr.shape)
    #print("trDS2_y shape ", trDS2["y"].shape)

    w = tikh @ trDS3['Next_Tmax']
    pred_cv = phi_cv @ w
    pred_t = phi_t @ w
    pred_tr = phi_tr @ w
    error_cv.append(np.linalg.norm(cvDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_cv))
    error_t.append(np.linalg.norm(tDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_t))
    error_tr.append(np.linalg.norm(trDS3['Next_Tmax'].to_numpy().reshape...
        (-1,1)-pred_tr))


########################################################################
pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])


########################################################################
plt.plot(lambda_list,error_cv)
plt.title('Error on cross-validation data vs regularization factor, ...
    tikhonov regularization for Dataset 3, for T_max')
plt.xlabel('lambda')
plt.ylabel('rmse error')


########################################################################
```

```
plt.hist(pred_tr, alpha=0.5)
plt.hist(trDS3["Next_Tmax"], alpha=0.5)


###########################################################################
error_list = np.array(error_list)
df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
df_error.sort_values(by=["Error"], ascending=True, inplace=True)
df_error


###########################################################################
l = .01
phi_tr = f(trDS3,l,optClds3)[0]
phi_t = f(tDS3,l,optClds3)[0]
tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
w = tikh @ trDS3["Next_Tmax"]
w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS3["Next_Tmax"]
pred_t = phi_t @ w
pred_tr = phi_tr @ w


###########################################################################
plt.scatter(trDS3.iloc[:,18],pred_tr)
plt.xlabel("Target output,train data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model output with Tikhonov ...
    regularization, for Next_Tmax")
plt.savefig("scatter_ds3tikhtrainT_max.png")
plt.show()


###########################################################################
plt.scatter(trDS3.iloc[:,18],pred_tr)
plt.xlabel("Target output,test data")
plt.ylabel("Model output")
plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
    for Next_Tmax")
plt.savefig("scatter_ds3tikhtestT_min.png")
plt.show()


###########################################################################
```