

ASSIGNMENT

CS5691 Pattern Recognition and Machine Learning

CS5691 Assignment 1 - Code

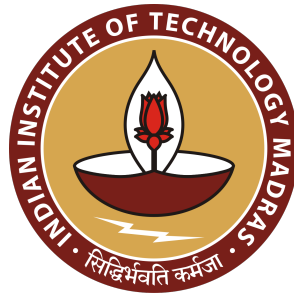
Team Members:

BE17B007 N Sowmya Manojna

PH17B010 Thakkar Riya Anandbhai

PH17B011 Chaithanya Krishna Moorthy

Indian Institute of Technology, Madras



Contents

1	Task 1	2
1.1	Main Code	2
1.2	Polynomial Regression Code	3
1.3	Grid Search Code	4
2	Task 2	5
3	Task 3	19
3.1	No Regularization, L2 Regularization	19
3.1.1	Pre-Processing	21
3.1.2	Gaussian Basis	24
3.2	Tikhonov Regularization	26

1 Task 1

1.1 Main Code

The code for Question 1 is as follows:

```
1 #####
2 # ## CS5691 PRML Assignment 1
3 # **Team 1**
4 # **Team Members:**
5 # N Sowmya Manojna    BE17B007
6 # Thakkar Riya Anandbhai  PH17B010
7 # Chaithanya Krishna Moorthy  PH17B011
8
9 #####
10 # Install required Packages
11 # Uncomment if you are running for the first time
12 # !pip install -r requirements.txt
13 # try:
14 #     !mkdir images/q1
15 # except:
16 #     pass
17
18 #####
19 import numpy as np
20 np.random.seed(0)
21 import pandas as pd
22
23 import warnings
24 warnings.filterwarnings("ignore")
25 import matplotlib.pyplot as plt
26 plt.style.use('science')
27 plt.rcParams['font.size'] = 18
28 plt.rcParams['axes.grid'] = True
29 plt.rcParams["grid.linestyle"] = (5,9)
30 plt.rcParams['figure.figsize'] = 8,6
31
32 from regression import PolynomialRegression
33 from gridsearch import GridSearch
34 #####
35 df = pd.read_csv("../datasets/function1.csv", index_col=0)
36 df.sort_values(by=["x"], inplace=True)
37 df.head()
38
39 #####
40 lambda_list = [0, 0.5, 1, 2, 10, 50, 100]
41 degrees_allowed = [2, 3, 6, 9]
42 datasizes_considered = [10, 200]
43 complete_dataset_size = df.shape[0]
44
45 X = df["x"].to_numpy().reshape(-1,1)
46 y = df["y"].to_numpy().reshape(-1,1)
47
48 results_df_list = []
49 correspondance_list = []
50 for sample_size in datasizes_considered:
51     gridsearch = GridSearch()
52     df_result, correspondance = gridsearch.get_result(df, sample_size=...
53         sample_size, degrees_allowed=degrees_allowed, lambda_list=lambda_list...
```

```

53     results_df_list.append(df_result)
54     correspondance_list.append(correspondance)
55
56     print("\nFor Sample Size of ", sample_size, " - GridSearch Results:")
57     print(df_result)
58     print("="*70)
59     gridsearch.get_plots(X, y, correspondance, sample_size, show=True)
60     #####
61
62     # From the resuts obtained, we see that degree=6, lambda=0.0
63     # best fits the model.
64
65     #####
66
67     best_degree = int(df_result.iloc[0]["degree"])
68     best_lmbda = df_result.iloc[0]["lambda"]
69
70     df_train = df.sample(frac=0.7, random_state=42)
71     df_test = df[~df.index.isin(df_train.index)]
72
73     X_train = df_train["x"].to_numpy().reshape(-1,1)
74     X_test = df_test["x"].to_numpy().reshape(-1,1)
75     y_train = df_train["y"].to_numpy().reshape(-1,1)
76     y_test = df_test["y"].to_numpy().reshape(-1,1)
77
78     regressor = PolynomialRegression()
79     X_train_poly = regressor.fit(X_train, y_train, degree=best_degree, lmbda=...
        best_lmbda)
80     y_train_pred = regressor.transform(X_train)
81     y_test_pred = regressor.transform(X_test)
82     train_error = regressor.error(y_train, y_train_pred)
83     test_error = regressor.error(y_test, y_test_pred)
84
85     print("Training Error:", train_error)
86     print("Testing Error:", test_error)

```

1.2 Polynomial Regression Code

The helper class used to perform PolynomialRegression is as follows:

```

1  import numpy as np
2
3  class PolynomialRegression():
4      def __init__(self):
5          pass
6
7      def fit(self, X, y, degree=2, lmbda=0):
8          self.degree = degree
9          self.lmbda = lmbda
10
11          X_poly = self.get_polynomial_features(X)
12          self.get_weights(X_poly, y)
13          return X_poly
14
15      def transform(self, X_val):
16          X_poly = self.get_polynomial_features(X_val)
17          y_val = X_poly @ self.W
18          return y_val
19

```

```

20     def fit_transform(self, X, y, degree=2, lambda=0):
21         self.fit(X, y, degree, lambda)
22         return self.transform(X)
23
24     def get_polynomial_features(self, X):
25         X_new = np.ones(X.shape)
26         for i in range(1, self.degree+1):
27             X_new = np.append(X_new, X**i, axis=1)
28         return X_new
29
30     def get_weights(self, X_poly, y):
31         d = X_poly.shape[1]
32         self.W = ((np.linalg.inv(X_poly.T @ X_poly + self.lambda*np.identity...
33             (d))) @ X_poly.T) @ y
34
35     def error(self, y_true, y_pred):
36         rmse = np.linalg.norm(y_pred-y_true)/(y_true.size)**0.5
37         return rmse

```

1.3 Grid Search Code

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

```

1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  from regression import PolynomialRegression
5
6  class GridSearch():
7      def __init__(self):
8          pass
9
10     def get_result(self, df, sample_size, degrees_allowed, lambda_list):
11         df_sample = df.sample(n=sample_size, random_state=42)
12
13         df_train = df_sample.sample(frac=0.9, random_state=42)
14         df_val = df_sample[~df_sample.index.isin(df_train.index)]
15
16         X_train = df_train["x"].to_numpy().reshape(-1, 1)
17         X_val = df_val["x"].to_numpy().reshape(-1, 1)
18
19         y_train = df_train["y"].to_numpy().reshape(-1, 1)
20         y_val = df_val["y"].to_numpy().reshape(-1, 1)
21
22         self.result = []
23         self.correspondance = {}
24
25         for degree in degrees_allowed:
26             for lambda in lambda_list:
27                 regressor = PolynomialRegression()
28                 regressor.fit(X_train, y_train, degree=degree, lambda=lambda)
29                 y_train_pred = regressor.transform(X_train)
30                 y_val_pred = regressor.transform(X_val)
31
32                 train_error = regressor.error(y_train, y_train_pred)
33                 val_error = regressor.error(y_val, y_val_pred)
34
35                 self.result.append([degree, lambda, train_error, val_error])

```

```

36         self.correspondance[(degree, lmbda)] = {"df_sample":...
37             df_sample, "regressor":regressor}
38
39     df_results = pd.DataFrame(self.result, columns=["degree", "lambda",...
40         "Train error", "Validation error"])
41     df_results["Sum Error"] = df_results["Train error"] + df_results["...
42         Validation error"]
43     df_results.sort_values(by="Sum Error", inplace=True)
44     return df_results, self.correspondance
45
46 def get_plots(self, X, y, correspondance, sample_size, show):
47     for key in correspondance:
48         df_sample = correspondance[key]["df_sample"]
49         df_sample.sort_values(by=["x"], inplace=True)
50         X_sample = df_sample["x"].to_numpy().reshape(-1,1)
51         y_sample = df_sample["y"].to_numpy().reshape(-1,1)
52
53         regressor = correspondance[key]["regressor"]
54         y_pred_sample = regressor.transform(X_sample)
55
56         title = "Curve Fitting - Degree: "+str(regressor.degree)\
57             +"; Sample Size: "+str(sample_size)+"; $\lambda$: "\
58             +str(regressor.lmbda)
59         fname = "d_"+str(regressor.degree)+"_size_"+str(sample_size)+"...
60             _l_"+str(regressor.lmbda)+".png"
61
62         plt.figure()
63         plt.plot(X, y, label="True Value")
64         if y_sample.size >= 100:
65             plt.plot(X_sample, y_sample, 'r.', alpha=0.5, label="...
66                 Sampled points")
67         else:
68             plt.plot(X_sample, y_sample, 'ro', alpha=0.75, label="...
69                 Sampled points")
70         plt.plot(X_sample, y_pred_sample, label="Predicted Value")
71         if title:
72             plt.title(title)
73             plt.xlabel("X-values")
74             plt.ylabel("Y-values")
75             plt.legend()
76         plt.savefig("images/"+fname)
77         if show:
78             plt.show()

```

2 Task 2

The code for Question 2 is as follows:

```

1  #####
2  from IPython import get_ipython
3
4  #####
5  import numpy as np
6  import pandas as pd
7  import math as ma
8  import matplotlib.pyplot as plt
9  from mpl_toolkits.mplot3d import Axes3D
10 get_ipython().run_line_magic('matplotlib', 'inline')

```

```

11
12 #####
13 # # Task 2
14
15 #####
16 func2d=pd.read_csv("function1_2d.csv",index_col = 0)
17
18 #####
19 # ## 2.1 Generating the polynomial basis functions of degrees 2, 3 and 6: (...
    Sp to dataset 2)
20
21 #####
22 ### Creating the polynomial basis functions of degree M and number of ...
    examples = n:
23
24 def create_phi(M,n,x1,x2):
25     d=2
26     D = int(ma.factorial(d+M)/(ma.factorial(d)*ma.factorial(M)))
27     phi = np.zeros((n,D))
28
29     if M == 2:
30         exp_ar = [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1]]
31     if M == 3:
32         exp_ar = ...
            [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1],[2,1],[1,2],[3,0],[0,3]]
33     if M == 6:
34         exp_ar = [[0,0],[1,0],[0,1],[2,0],[0,2],[1,1],[2,1],[1,2],\
35                 [3,0],[0,3],[3,1],[1,3],[2,2],[4,0],[0,4],[4,1],\
36                 [1,4],[2,3],[3,2],[5,0],[0,5],[5,1],[1,5],[2,4],\
37                 [4,2],[3,3],[6,0],[0,6]]
38
39     for i in range(D):
40         phi[:,i] = (x1**(exp_ar[i][0]))*(x2**(exp_ar[i][1]))
41
42     return(phi)
43
44 #####
45 # ## 2.2 Solving for optimal parameters using regularization, lambda=0 for...
    unregularized: (versatile)
46 #####
47 # ### The function regularized_pseudo_inv(lamb,X) returns:
48 # 
$$(\lambda I + X^T X)^{-1} X^T$$

49 #
50 # Where lambda is the hyperparameter in the quadratic regularization
51
52 #####
53 def regularized_pseudo_inv(lamb,phi):
54     return(np.matmul(np.linalg.inv(lamb*np.identity(phi.shape[1])+np.matmul...
        (np.transpose(phi),phi)),np.transpose(phi)))
55
56 #####
57 # ### 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.
58
59 #####
60 def opt_regularized_param(lamb,phi,y):
61     return(np.matmul(regularized_pseudo_inv(lamb,phi),y))
62
63 #####

```

```

64 # ## 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)
65 # $$y=Xw$$
66
67 #####
68 def y_pred(phi,w):
69     return(np.matmul(phi,w))
70
71 #####
72 # ## 2.4 Splitting the data into train, cross-validation and test and ...
    helper function for surface plot: (versatile)
73
74 #####
75 def create_datasets(data,train_size,cv_size):
76     data.sample(frac=1).reset_index(drop=True)
77     data_train=data[0:train_size]
78     data_cv=data[train_size:train_size+cv_size]
79     data_test=data[cv_size+train_size:]
80     return(data_train,data_cv,data_test)
81
82
83
84 #####
85 def split_cols(data_train,data_cv,data_test):
86     #x1_train,x2_train,y_train,x1_cv,x2_cv,y_cv,x1_test,x2_test,y_test
87     x1_train=np.array(data_train)[: ,0]
88     x2_train=np.array(data_train)[: ,1]
89     y_train=np.array(data_train)[: ,2]
90     x1_cv=np.array(data_cv)[: ,0]
91     x2_cv=np.array(data_cv)[: ,1]
92     y_cv=np.array(data_cv)[: ,2]
93     x1_test=np.array(data_test)[: ,0]
94     x2_test=np.array(data_test)[: ,1]
95     y_test=np.array(data_test)[: ,2]
96
97     return(x1_train,x2_train,y_train,x1_cv,x2_cv,y_cv,x1_test,x2_test,...
            y_test)
98
99
100
101 #####
102 ### Function to calculate RMSE:
103
104 def RMSE(y_pred, t):
105     n = len(y_pred)
106     return(np.sqrt(np.sum((y_pred - t)**2)/n))
107
108
109 #####
110 #plot the approximated function
111
112
113 def plot_approxY(M,w,y_train,y_pred,train_size,l,a,b):
114
115     # M is the degree of the complexity
116     # w is the array of parameters
117     # y is the actual value of y
118     # l is the value of lambda
119     x1 = np.arange(-16,16,0.5)
120     x2 = np.arange(-16,16,0.5)

```



```

121     x1, x2 = np.meshgrid(x1,x2)
122     Y= np.zeros((64,64))
123     for i in range(64):
124         for j in range(64):
125             Y[i,j] = np.sum(np.matmul(create_phi(M,1,x2[i,j],x1[i,j]),w))
126     fig = plt.figure(figsize=(15,8))
127     ax=fig.gca(projection="3d")
128     ax.plot_wireframe(x1,x2,Y,label="Approximated function")
129     x = b
130     y = a
131     z = y_train
132     ax.scatter(x,y,z,color='red',label="Original train data points")
133     ax.set_ylabel("x1")
134     ax.set_xlabel("x2")
135     ax.set_zlabel("y")
136     ax.view_init(0,45)
137     plt.legend(loc=4)
138     plt.title("Surface plot with degree of complexity = %i, Train data size...
              = %i and regularization parameter, lambda = %.1f"%(M,train_size,1)...
              )
139     plt.show()
140
141     #####
142     # ## 2.5 Predicting for degree 2, train size 50:
143
144     #####
145     data2_train50,data2_cv50,data2_test50=create_datasets(func2d,50,30)
146
147
148     #####
149     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)
150
151
152     #####
153     ### design matrix:
154     phi2_train50=create_phi(2,len(y2_train50),x12_train50,x22_train50)
155     phi2_cv50=create_phi(2,len(y2_cv50),x12_cv50,x22_cv50)
156     phi2_test50=create_phi(2,len(y2_test50),x12_test50,x22_test50)
157
158
159     #####
160     y2_testpred50={}
161     y2_trainpred50={}
162     y2_cvpred50={}
163     lambda_list=[0,0.5,1,2,10,50,100]
164     rmse2_train50=[]
165     rmse2_test50=[]
166     rmse2_cv50=[]
167
168
169     #####
170     for l in lambda_list:
171         w_2_50=opt_regularized_param(l,phi2_train50,y2_train50);
172         y2_trainpred50[l]=y_pred(phi2_train50,w_2_50)
173         y2_testpred50[l]=y_pred(phi2_test50,w_2_50);
174         y2_cvpred50[l]=y_pred(phi2_cv50,w_2_50);
175         rmse2_train50.append(RMSE(y2_trainpred50[l],y2_train50))
176         rmse2_test50.append(RMSE(y2_testpred50[l],y2_test50))
177         rmse2_cv50.append(RMSE(y2_cvpred50[l],y2_cv50))

```

```

178
179
180
181
182 #####
183 data2_50=pd.DataFrame(list(zip(lambda_list,rmse2_train50,rmse2_cv50,...
    rmse2_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
184
185
186 #####
187 data2_50
188
189
190 #####
191 plt.figure()
192 plt.plot(data2_50["Lambda"],data2_50["RMSE Train"],label="RMSE Train")
193 plt.plot(data2_50["Lambda"],data2_50["RMSE CV"],label="RMSE CV")
194 plt.plot(data2_50["Lambda"],data2_50["RMSE test"],label="RMSE test")
195 plt.xlabel("Lambda->")
196 plt.ylabel("RMSE")
197 plt.legend()
198 plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 2"...
    )
199 plt.savefig("d2_50.png")
200 plt.show()
201
202 #####
203 # ### Surface plots for various values of lambda:
204
205 #####
206 for l in lambda_list:
207     plot_approxY(2, w_2_50,y2_train50,y2_trainpred50[l],50,1,x12_train50,...
        x22_train50)
208     plt.savefig("surfaceplotsd2.png")
209
210 #####
211 # ## 2.6 Predicting for degree of complexity = 2 and train data size = 200
212
213 #####
214 data2_train200,data2_cv200,data2_test200=create_datasets(func2d,200,90)
215
216
217 #####
218 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)
219
220
221 #####
222 ### design matrix:
223 phi2_train200=create_phi(2,len(y2_train200),x12_train200,x22_train200)
224 phi2_cv200=create_phi(2,len(y2_cv200),x12_cv200,x22_cv200)
225 phi2_test200=create_phi(2,len(y2_test200),x12_test200,x22_test200)
226
227
228 #####
229 y2_testpred200={}
230 y2_trainpred200={}
231 y2_cvpred200={}
232 lambda_list=[0,0.5,1,2,10,50,100]

```

```

233 rmse2_train200=[]
234 rmse2_test200=[]
235 rmse2_cv200=[]
236
237
238 #####
239 for l in lambda_list:
240     w_2_200=opt_regularized_param(l,phi2_train200,y2_train200);
241     y2_trainpred200[l]=y_pred(phi2_train200,w_2_200)
242     y2_testpred200[l]=y_pred(phi2_test200,w_2_200);
243     y2_cvpred200[l]=y_pred(phi2_cv200,w_2_200);
244     rmse2_train200.append(RMSE(y2_trainpred200[l],y2_train200))
245     rmse2_test200.append(RMSE(y2_testpred200[l],y2_test200))
246     rmse2_cv200.append(RMSE(y2_cvpred200[l],y2_cv200))
247
248
249
250 #####
251 data2_200=pd.DataFrame(list(zip(lambda_list,rmse2_train200,rmse2_cv200,...
    rmse2_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
252
253
254 #####
255 data2_200
256
257
258 #####
259 data2_200.to_csv("RMSE-lambda for complexity = 2 and train size = 200")
260
261
262 #####
263 plt.figure()
264 plt.plot(data2_200["Lambda"],data2_200["RMSE Train"],label="RMSE Train")
265 plt.plot(data2_200["Lambda"],data2_200["RMSE CV"],label="RMSE CV")
266 plt.plot(data2_200["Lambda"],data2_200["RMSE test"],label="RMSE test")
267 plt.xlabel("Lambda->")
268 plt.ylabel("RMSE")
269 plt.legend()
270 plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 2...")
271 plt.savefig("d2_200.png")
272 plt.show()
273
274 #####
275 # ### Surface plots for various values of lambda:
276
277 #####
278 for l in lambda_list:
279     plot_approxY(2, w_2_200,y2_train200,y2_trainpred200[l],200,1,...
        x12_train200,x22_train200)
280
281 #####
282 # ## 2.7 Predicting for degree of complexity = 2 and train data size = 500
283
284 #####
285 data2_train500,data2_cv500,data2_test500=create_datasets(func2d,500,200)
286 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)
287

```

```

288 ### design matrix:
289 phi2_train500=create_phi(2,len(y2_train500),x12_train500,x22_train500)
290 phi2_cv500=create_phi(2,len(y2_cv500),x12_cv500,x22_cv500)
291 phi2_test500=create_phi(2,len(y2_test500),x12_test500,x22_test500)
292
293 y2_testpred500={}
294 y2_trainpred500={}
295 y2_cvpred500={}
296 lambda_list=[0,0.5,1,2,10,50,100]
297 rmse2_train500=[]
298 rmse2_test500=[]
299 rmse2_cv500=[]
300
301 for l in lambda_list:
302     w_2_500=opt_regularized_param(l,phi2_train500,y2_train500);
303     y2_trainpred500[l]=y_pred(phi2_train500,w_2_500)
304     y2_testpred500[l]=y_pred(phi2_test500,w_2_500);
305     y2_cvpred500[l]=y_pred(phi2_cv500,w_2_500);
306     rmse2_train500.append(RMSE(y2_trainpred500[l],y2_train500))
307     rmse2_test500.append(RMSE(y2_testpred500[l],y2_test500))
308     rmse2_cv500.append(RMSE(y2_cvpred500[l],y2_cv500))
309
310
311
312 #####
313 data2_500=pd.DataFrame(list(zip(lambda_list,rmse2_train500,rmse2_cv500,...
314                               rmse2_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
315
316 #####
317 data2_500
318
319
320 #####
321 data2_500.to_csv("RMSE-lambda for complexity =2 train size =500")
322
323
324 #####
325 plt.figure()
326 plt.plot(data2_500["Lambda"],data2_500["RMSE Train"],label="RMSE Train")
327 plt.plot(data2_500["Lambda"],data2_500["RMSE CV"],label="RMSE CV")
328 plt.plot(data2_500["Lambda"],data2_500["RMSE test"],label="RMSE test")
329 plt.xlabel("Lambda->")
330 plt.ylabel("RMSE")
331 plt.legend()
332 plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 2...")
333 plt.savefig("d2_500.png")
334 plt.show()
335
336 #####
337 # ### Surface plots for various values of lambda:
338
339 #####
340 for l in lambda_list:
341     plot_approxY(2, w_2_500,y2_train500,y2_trainpred500[l],500,1,...
342                 x12_train500,x22_train500)
343 #####
344 # ## 2.8 For train size =50 and degree of complexity = 3

```

```

345
346 #####
347 data3_train50,data3_cv50,data3_test50=create_datasets(func2d,50,30)
348 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)
349
350 ### design matrix:
351 phi3_train50=create_phi(3,len(y3_train50),x13_train50,x23_train50)
352 phi3_cv50=create_phi(3,len(y3_cv50),x13_cv50,x23_cv50)
353 phi3_test50=create_phi(3,len(y3_test50),x13_test50,x23_test50)
354
355 y3_testpred50={}
356 y3_trainpred50={}
357 y3_cvpred50={}
358 lambda_list=[0,0.5,1,2,10,50,100]
359 rmse3_train50=[]
360 rmse3_test50=[]
361 rmse3_cv50=[]
362
363 for l in lambda_list:
364     w_3_50=opt_regularized_param(l,phi3_train50,y3_train50);
365     y3_trainpred50[l]=y_pred(phi3_train50,w_3_50)
366     y3_testpred50[l]=y_pred(phi3_test50,w_3_50);
367     y3_cvpred50[l]=y_pred(phi3_cv50,w_3_50);
368     rmse3_train50.append(RMSE(y3_trainpred50[l],y3_train50))
369     rmse3_test50.append(RMSE(y3_testpred50[l],y3_test50))
370     rmse3_cv50.append(RMSE(y3_cvpred50[l],y3_cv50))
371
372
373
374 #####
375 data3_50=pd.DataFrame(list(zip(lambda_list,rmse3_train50,rmse3_cv50,...
      rmse3_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
376
377
378 #####
379 data3_50.to_csv("rmse lambda for complexity = 3 train sie =50")
380
381
382 #####
383 plt.figure()
384 plt.plot(data3_50["Lambda"],data3_50["RMSE Train"],label="RMSE Train")
385 plt.plot(data3_50["Lambda"],data3_50["RMSE CV"],label="RMSE CV")
386 plt.plot(data3_50["Lambda"],data3_50["RMSE test"],label="RMSE test")
387 plt.xlabel("Lambda->")
388 plt.ylabel("RMSE")
389 plt.legend()
390 plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 3"...
      )
391 plt.savefig("d3_50.png")
392 plt.show()
393
394 #####
395 # ### Surface plots for various values of lambda
396
397 #####
398 for l in lambda_list:
399     plot_approxY(3, w_3_50,y3_train50,y3_trainpred50[l],50,1,x13_train50,...
      x23_train50)
400

```

```

401 #####
402 # ## 2.9 For train size = 200 and degree of complexity = 3:
403
404 #####
405 data3_train200,data3_cv200,data3_test200=create_datasets(func2d,200,90)
406 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)
407
408 ### design matrix:
409 phi3_train200=create_phi(3,len(y3_train200),x13_train200,x23_train200)
410 phi3_cv200=create_phi(3,len(y3_cv200),x13_cv200,x23_cv200)
411 phi3_test200=create_phi(3,len(y3_test200),x13_test200,x23_test200)
412
413 y3_testpred200={}
414 y3_trainpred200={}
415 y3_cvpred200={}
416 lambda_list=[0,0.5,1,2,10,50,100]
417 rmse3_train200=[]
418 rmse3_test200=[]
419 rmse3_cv200=[]
420
421 for l in lambda_list:
422     w_3_200=opt_regularized_param(l,phi3_train200,y3_train200);
423     y3_trainpred200[l]=y_pred(phi3_train200,w_3_200)
424     y3_testpred200[l]=y_pred(phi3_test200,w_3_200);
425     y3_cvpred200[l]=y_pred(phi3_cv200,w_3_200);
426     rmse3_train200.append(RMSE(y3_trainpred200[l],y3_train200))
427     rmse3_test200.append(RMSE(y3_testpred200[l],y3_test200))
428     rmse3_cv200.append(RMSE(y3_cvpred200[l],y3_cv200))
429
430
431
432 #####
433 data3_200=pd.DataFrame(list(zip(lambda_list,rmse3_train200,rmse3_cv200,...
    rmse3_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
434
435
436 #####
437 data3_200.to_csv("rmse lambda for complexity = 3, train size = 200")
438
439
440 #####
441 plt.figure()
442 plt.plot(data3_200["Lambda"],data3_200["RMSE Train"],label="RMSE Train")
443 plt.plot(data3_200["Lambda"],data3_200["RMSE CV"],label="RMSE CV")
444 plt.plot(data3_200["Lambda"],data3_200["RMSE test"],label="RMSE test")
445 plt.xlabel("Lambda->")
446 plt.ylabel("RMSE")
447 plt.legend()
448 plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 3...
    ")
449 plt.savefig("d3_200.png")
450 plt.show()
451
452 #####
453 # ### Surface Plots for various values of lambda:
454
455 #####
456 for l in lambda_list:

```

```

457     plot_approxY(3, w_3_200,y3_train200,y3_trainpred200[1],200,1,...
458                 x13_train200,x23_train200)
459 #####
460 # ## 2.10 For train data size = 500 and degree of complexity = 3
461 #####
462 #####
463 data3_train500,data3_cv500,data3_test500=create_datasets(func2d,500,200)
464 x13_train500,x23_train500,y3_train500,x13_cv500,x23_cv500,y3_cv500,...
465     x13_test500,x23_test500,y3_test500=split_cols(data3_train500,...
466     data3_cv500,data3_test500)
467 #####
468 ### design matrix:
469 phi3_train500=create_phi(3,len(y3_train500),x13_train500,x23_train500)
470 phi3_cv500=create_phi(3,len(y3_cv500),x13_cv500,x23_cv500)
471 phi3_test500=create_phi(3,len(y3_test500),x13_test500,x23_test500)
472 #####
473 y3_testpred500={}
474 y3_trainpred500={}
475 y3_cvpred500={}
476 lambda_list=[0,0.5,1,2,10,50,100]
477 rmse3_train500=[]
478 rmse3_test500=[]
479 rmse3_cv500=[]
480 #####
481 for l in lambda_list:
482     w_3_500=opt_regularized_param(l,phi3_train500,y3_train500);
483     y3_trainpred500[l]=y_pred(phi3_train500,w_3_500)
484     y3_testpred500[l]=y_pred(phi3_test500,w_3_500);
485     y3_cvpred500[l]=y_pred(phi3_cv500,w_3_500);
486     rmse3_train500.append(RMSE(y3_trainpred500[l],y3_train500))
487     rmse3_test500.append(RMSE(y3_testpred500[l],y3_test500))
488     rmse3_cv500.append(RMSE(y3_cvpred500[l],y3_cv500))
489 #####
490 #####
491 data3_500=pd.DataFrame(list(zip(lambda_list,rmse3_train500,rmse3_cv500,...
492     rmse3_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
493 #####
494 #####
495 data3_500.to_csv("rmse lambda for complexity = 3 train size = 500")
496 #####
497 #####
498 #####
499 plt.figure()
500 plt.plot(data3_500["Lambda"],data3_500["RMSE Train"],label="RMSE Train")
501 plt.plot(data3_500["Lambda"],data3_500["RMSE CV"],label="RMSE CV")
502 plt.plot(data3_500["Lambda"],data3_500["RMSE test"],label="RMSE test")
503 plt.xlabel("Lambda->")
504 plt.ylabel("RMSE")
505 plt.legend()
506 plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 3...")
507 plt.savefig("d3_500.png")
508 plt.show()
509 #####
510 #####
511 # ### Surface Plots for various values of lambda:

```

```

512
513 #####
514 for l in lambda_list:
515     plot_approxY(3, w_3_500,y3_train500,y3_trainpred500[l],500,1,...
516                 x13_train500,x23_train500)
517 #####
518 # ## 2.11 Degree of complexity = 6 and train data size = 50
519
520 #####
521 data6_train50,data6_cv50,data6_test50=create_datasets(func2d,50,30)
522 x16_train50,x26_train50,y6_train50,x16_cv50,x26_cv50,y6_cv50,x16_test50,...
523     x26_test50,y6_test50=split_cols(data6_train50,data6_cv50,data6_test50)
524
525 ### design matrix:
526 phi6_train50=create_phi(6,len(y6_train50),x16_train50,x26_train50)
527 phi6_cv50=create_phi(6,len(y6_cv50),x16_cv50,x26_cv50)
528 phi6_test50=create_phi(6,len(y6_test50),x16_test50,x26_test50)
529
530 y6_testpred50={}
531 y6_trainpred50={}
532 y6_cvpred50={}
533 lambda_list=[0,0.5,1,2,10,50,100]
534 rmse6_train50=[]
535 rmse6_test50=[]
536 rmse6_cv50=[]
537
538 for l in lambda_list:
539     w_6_50=opt_regularized_param(l,phi6_train50,y6_train50);
540     y6_trainpred50[l]=y_pred(phi6_train50,w_6_50)
541     y6_testpred50[l]=y_pred(phi6_test50,w_6_50);
542     y6_cvpred50[l]=y_pred(phi6_cv50,w_6_50);
543     rmse6_train50.append(RMSE(y6_trainpred50[l],y6_train50))
544     rmse6_test50.append(RMSE(y6_testpred50[l],y6_test50))
545     rmse6_cv50.append(RMSE(y6_cvpred50[l],y6_cv50))
546
547
548 #####
549 data6_50=pd.DataFrame(list(zip(lambda_list,rmse6_train50,rmse6_cv50,...
550                               rmse6_test50)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
551
552 #####
553 data6_50.to_csv("rmse_lambda complexity= 6, train = 50")
554
555
556 #####
557 plt.figure()
558 plt.plot(data6_50["Lambda"],data6_50["RMSE Train"],label="RMSE Train")
559 plt.plot(data6_50["Lambda"],data6_50["RMSE CV"],label="RMSE CV")
560 plt.plot(data6_50["Lambda"],data6_50["RMSE test"],label="RMSE test")
561 plt.xlabel("Lambda->")
562 plt.ylabel("RMSE")
563 plt.legend()
564 plt.title("RMSE vs lambda for train size = 50 and degree of complexity = 6"...
565           )
566 plt.savefig("d6_50.png")
567 plt.show()

```



```

568 #####
569 # ### Surface plots for various values of lambda:
570
571 #####
572 for l in lambda_list:
573     plot_approxY(6, w_6_50,y6_train50,y6_trainpred50[l],50,l,x16_train50,...
574                 x26_train50)
575
576 #####
577 # ## 2.12 For degree of complexity = 6 and train data size = 200
578 #####
579 data6_train200,data6_cv200,data6_test200=create_datasets(func2d,200,90)
580 x16_train200,x26_train200,y6_train200,x16_cv200,x26_cv200,y6_cv200,...
581     x16_test200,x26_test200,y6_test200=split_cols(data6_train200,...
582     data6_cv200,data6_test200)
583
584 ##### design matrix:
585 phi6_train200=create_phi(6,len(y6_train200),x16_train200,x26_train200)
586 phi6_cv200=create_phi(6,len(y6_cv200),x16_cv200,x26_cv200)
587 phi6_test200=create_phi(6,len(y6_test200),x16_test200,x26_test200)
588
589 y6_testpred200={}
590 y6_trainpred200={}
591 y6_cvpred200={}
592 lambda_list=[0,0.5,1,2,10,50,100]
593 rmse6_train200=[]
594 rmse6_test200=[]
595 rmse6_cv200=[]
596
597 for l in lambda_list:
598     w_6_200=opt_regularized_param(l,phi6_train200,y6_train200);
599     y6_trainpred200[l]=y_pred(phi6_train200,w_6_200)
600     y6_testpred200[l]=y_pred(phi6_test200,w_6_200);
601     y6_cvpred200[l]=y_pred(phi6_cv200,w_6_200);
602     rmse6_train200.append(RMSE(y6_trainpred200[l],y6_train200))
603     rmse6_test200.append(RMSE(y6_testpred200[l],y6_test200))
604     rmse6_cv200.append(RMSE(y6_cvpred200[l],y6_cv200))
605
606 #####
607 data6_200=pd.DataFrame(list(zip(lambda_list,rmse6_train200,rmse6_cv200,...
608     rmse6_test200)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
609
610 #####
611 data6_200.to_csv("rmse lambda complexity = 6 train = 200")
612
613 #####
614 plt.figure()
615 plt.plot(data6_200["Lambda"],data6_200["RMSE Train"],label="RMSE Train")
616 plt.plot(data6_200["Lambda"],data6_200["RMSE CV"],label="RMSE CV")
617 plt.plot(data6_200["Lambda"],data6_200["RMSE test"],label="RMSE test")
618 plt.xlabel("Lambda->")
619 plt.ylabel("RMSE")
620 plt.legend()
621 plt.title("RMSE vs lambda for train size = 200 and degree of complexity = 6...
622 ")

```

```

623 plt.savefig("d6_200.png")
624 plt.show()
625
626 #####
627 # ### Surface plots for various values of lambda:
628
629 #####
630 for l in lambda_list:
631     plot_approxY(6, w_6_200,y6_train200,y6_trainpred200[l],200,1,...
632                 x16_train200,x26_train200)
633
634 #####
635 # ## 2.13 For degree of complexity = 6 and train data size = 500
636
637 #####
638 data6_train500,data6_cv500,data6_test500=create_datasets(func2d,500,200)
639 x16_train500,x26_train500,y6_train500,x16_cv500,x26_cv500,y6_cv500,...
640 x16_test500,x26_test500,y6_test500=split_cols(data6_train500,...
641 data6_cv500,data6_test500)
642
643 ##### design matrix:
644 phi6_train500=create_phi(6,len(y6_train500),x16_train500,x26_train500)
645 phi6_cv500=create_phi(6,len(y6_cv500),x16_cv500,x26_cv500)
646 phi6_test500=create_phi(6,len(y6_test500),x16_test500,x26_test500)
647
648 y6_testpred500={}
649 y6_trainpred500={}
650 y6_cvpred500={}
651 lambda_list=[0,0.5,1,2,10,50,100]
652 rmse6_train500=[]
653 rmse6_test500=[]
654 rmse6_cv500=[]
655
656 for l in lambda_list:
657     w_6_500=opt_regularized_param(l,phi6_train500,y6_train500);
658     y6_trainpred500[l]=y_pred(phi6_train500,w_6_500)
659     y6_testpred500[l]=y_pred(phi6_test500,w_6_500);
660     y6_cvpred500[l]=y_pred(phi6_cv500,w_6_500);
661     rmse6_train500.append(RMSE(y6_trainpred500[l],y6_train500))
662     rmse6_test500.append(RMSE(y6_testpred500[l],y6_test500))
663     rmse6_cv500.append(RMSE(y6_cvpred500[l],y6_cv500))
664
665 #####
666 data6_500=pd.DataFrame(list(zip(lambda_list,rmse6_train500,rmse6_cv500,...
667 rmse6_test500)),columns=["Lambda", "RMSE Train","RMSE CV","RMSE test"])
668
669 #####
670 data6_500.to_csv("rmse lambda complexity = 6 train = 500")
671
672 #####
673 plt.figure()
674 plt.plot(data6_500["Lambda"],data6_500["RMSE Train"],label="RMSE Train")
675 plt.plot(data6_500["Lambda"],data6_500["RMSE CV"],label="RMSE CV")
676 plt.plot(data6_500["Lambda"],data6_500["RMSE test"],label="RMSE test")
677 plt.xlabel("Lambda->")
678 plt.ylabel("RMSE")

```

```

679 plt.legend()
680 plt.title("RMSE vs lambda for train size = 500 and degree of complexity = 6...")
681 plt.savefig("d6_500.png")
682 plt.show()
683
684
685 #####
686 data6_500
687
688 #####
689 # ### Surface plots for various values of lambda:
690
691 #####
692 for l in lambda_list:
693     plot_approxY(6, w_6_500,y6_train500,y6_trainpred500[l],500,1,...
694                 x16_train500,x26_train500)
695
696 #####
697 # # Conclusion:
698 #
699 # The model with best RMSE values over all three datasets: train, cross-...
700 # validation and test is with degree of complexity 6, train data size = ...
701 # 500 and regularization parameter lambda = 0
702
703 #####
704 plt.figure()
705 plt.scatter(y6_train500,y6_trainpred500[0],label="predicted y")
706 plt.legend()
707 plt.xlabel("Actual y")
708 plt.ylabel("Predicted y")
709 plt.title("Predicted output vs actual output for train data set")
710 plt.savefig("predgoodtrain.png")
711 plt.show()
712
713
714 #####
715 plt.figure()
716 plt.scatter(y6_cv500,y6_cvpred500[0],label="predicted y")
717 plt.legend()
718 plt.xlabel("Actual y")
719 plt.ylabel("Predicted y")
720 plt.title("Predicted output vs actual output for cross-validation data set"...
721 )
722 plt.savefig("predcv.png")
723 plt.show()
724
725
726 #####
727 plt.figure()
728 plt.scatter(y6_test500,y6_testpred500[0],label="predicted y")
729 plt.legend()
730 plt.xlabel("Actual y")
731 plt.ylabel("Predicted y")
732 plt.title("Predicted output vs actual output for test data set")
733 plt.savefig("predtest.png")
734 plt.show()
735
736
737 #####

```

```

734 ### rmse train:
735 rmse6_train500[0]
736
737
738 #####
739 ### rmse cross validation:
740 rmse6_cv500[0]
741
742
743 #####
744 ### rmse test data:
745 rmse6_test500[0]
746
747
748 #####
749 plt.figure()
750 plt.scatter(x16_train500,y6_train500)
751
752
753 #####
754 plt.figure()
755 plt.scatter(x26_train500,y6_train500)
756
757
758 #####
759 plt.figure()
760 plt.scatter(y6_trainpred500[0]-y6_train500)
761
762
763 #####

```

3 Task 3

3.1 No Regularization, L2 Regularization

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

```

1 #####
2 # ## CS5691 PRML Assignment 1
3 # **Team 1**
4 # **Team Members:**
5 # N Sowmya Manojna BE17B007
6 # Thakkar Riya Anandbhai PH17B010
7 # Chaithanya Krishna Moorthy PH17B011
8
9 #####
10 # Install required Packages
11 # Uncomment if you are running for the first time
12 # !pip install -r requirements.txt
13 # try:
14 #     !mkdir images/q3
15 # except:
16 #     pass
17
18 #####
19 import os
20 import missingno
21 import numpy as np

```

```

22 import pandas as pd
23 import seaborn as sns
24 import matplotlib.pyplot as plt
25 from preprocess import PreProcess
26 from gaussianbasis import GaussianBasis
27 from sklearn.cluster import KMeans
28 from mpl_toolkits.mplot3d import Axes3D
29 from statsmodels.stats.outliers_influence import variance_inflation_factor
30
31 #####
32 print("Reading Dataset... ", end="")
33 df = pd.read_csv("../datasets/1_bias_clean.csv")
34 print("Done!")
35
36 print("Starting Preprocessing... ")
37 preprocess = PreProcess()
38 preprocess.clean(df, verbose=True)
39 print("="*60)
40 print("="*60)
41 print("Preprocessing Done!!")
42
43 df_save = pd.read_csv("../datasets/processed.csv", index_col=0)
44 df_new = df_save.copy()
45 df_new = df_new.drop(["Next_Tmax", "Next_Tmin"], axis=1)
46
47 num_clusters = [1]
48 num_clusters.extend(range(2,10))
49 num_clusters.extend(range(15, 31, 5))
50 num_clusters.extend(range(40, 101, 10))
51
52 print("Starting Regularization... ", end="")
53 lambda_list = [0, 0.5, 1, 2, 10, 50, 100]
54 regressor = GaussianBasis()
55 output = regressor.fit_grid(df_new, df_save, num_clusters, regularization="...
    L2", lambda_list=lambda_list, verbose=True, show=False)
56 df_result, sse_dict, num_clusters = output
57
58
59 df_result["Sum Error"] = df_result["Error 1"] + df_result["Error 2"]
60 df_result.sort_values(by=["Sum Error"], inplace=True)
61 print(df_result)
62
63 sse_list = [sse_dict[i] for i in sse_dict]
64 plt.figure(figsize=[12,8])
65 plt.plot(num_clusters,sse_list )
66 plt.xlabel("Number of Clusters")
67 plt.ylabel("SSE")
68 plt.title("Knee Plot for determining the number of clusters")
69 plt.grid()
70 plt.show()
71
72 plt.figure(figsize=[12,8])
73 plt.plot(num_clusters[:5], sse_list[:5])
74 plt.xlabel("Number of Clusters")
75 plt.ylabel("SSE")
76 plt.title("Knee Plot for determining the number of clusters")
77 plt.grid()
78 plt.show()

```

3.1.1 Pre-Processing

The helper class used to perform PreProcessing is as follows:

```
1 import missingno
2 import numpy as np
3 import pandas as pd
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 from mpl_toolkits.mplot3d import Axes3D
7 from sklearn.cluster import KMeans
8 from statsmodels.stats.outliers_influence import variance_inflation_factor
9
10 #####
11 class PreProcess():
12     def __init__(self):
13         pass
14
15     def clean(self, df, verbose=False):
16         if verbose==True:
17             print("="*60)
18         if verbose==True:
19             print("Sample of Dataset")
20         if verbose==True:
21             print(df.head())
22
23         if verbose==True:
24             print("="*60)
25         if verbose==True:
26             print("Information of Dataset")
27         df.info()
28
29         if verbose==True:
30             print("="*60)
31         if verbose==True:
32             print("'Nan' Distribution:")
33         if verbose==True:
34             print(df.isnull().sum())
35
36         if verbose==True:
37             print("="*60)
38         if verbose==True:
39             print("Description of the dataset:")
40         if verbose==True:
41             print(df.describe())
42
43         plt.figure()
44         missingno.matrix(df)
45         plt.title("Missing Values Visualization")
46         plt.show()
47
48         if verbose==True:
49             print("Removing the Rows with NaNs")
50         df_clean = df.dropna(axis=0)
51         plt.figure()
52         missingno.matrix(df_clean)
53         plt.title("Missing Values Visualization - After NaN removal")
54
55         df = df_clean
56         desc = df.describe()
57         sum(desc.loc["std"] == 0)
```

```

58
59 # sns.pairplot(df)
60
61 plt.figure(figsize=[15,15])
62 sns.heatmap(df.corr().round(2), linewidths=.5, annot=True)
63 plt.title("Heatmap of the data")
64 plt.show()
65
66 if verbose==True:
67     print("="*60)
68 if verbose==True:
69     print("Identifying highly correlated features... ")
70 # Use the upper triangle to mask the correlation matrix
71 df_new = df.copy()
72 df_new = df_new.drop(["Next_Tmin", "Next_Tmax"], axis=1)
73 upper_triangle = np.triu(np.ones(df_new.corr().shape)).astype(bool)
74
75 # Get the correlation pairs
76 correlation_pairs = df_new.corr().mask(upper_triangle).abs().\
77     .unstack().sort_values(ascending=False)
78 correlation_pairs = pd.DataFrame(correlation_pairs)
79 if verbose==True:
80     print("Correlation between Features")
81 if verbose==True:
82     print(correlation_pairs.head(25))
83 if verbose==True:
84     print("="*50)
85
86 # if verbose==True:
87 #     print the highly correlated features
88 highly_correlated = correlation_pairs[correlation_pairs[0]>0.75]
89 if verbose==True:
90     print("Highly Correlated Features")
91 if verbose==True:
92     print(highly_correlated)
93 if verbose==True:
94     print("="*50)
95
96 # Remove each of the highly correlated features and check
97 highly_correlated.reset_index(inplace=True)
98 hc_features = highly_correlated["level_0"]
99
100 df_new.head()
101 if verbose==True:
102     print("Done!")
103
104 # Remove highly correlated features
105 for feature in hc_features:
106     if verbose==True:
107         print("Removing:", feature, "...")
108     df_new = df_new.drop([feature], axis=1)
109     upper_triangle = np.triu(np.ones(df_new.corr().shape)).astype(
110         bool)
111     c = df_new.corr().mask(upper_triangle).abs().unstack().\
112         sort_values(ascending=False)
113     c = pd.DataFrame(c)
114     if verbose==True:
115         print(c[c[0]>0.75])
116     if verbose==True:
117         print("="*50, "\n")

```

```

115         if c[c[0]>0.75].size == 0:
116             if verbose==True:
117                 print("The number of highly correlated fetaures has ...
                    become zero!")
118             if verbose==True:
119                 print("Preventing all further removals :)")
120             break
121
122     df_new.head()
123
124     if verbose==True:
125         print("="*60)
126     if verbose==True:
127         print("Checking for Variance Inflation Factor... ")
128     # Variance Inflation Factor - directly correlated
129     X_df = df_new.copy()
130
131     vif = pd.DataFrame()
132     vif["Features"] = X_df.columns
133     vif["VIF Factor"]=[variance_inflation_factor(X_df.values, i) for i ...
        in range(X_df.shape[1])]
134     vif.sort_values(by=["VIF Factor"], ascending=False, inplace=True)
135
136     max_col = vif[vif["VIF Factor"]>1000]
137     for feature in max_col["Features"]:
138         max_col = vif[vif["VIF Factor"]>1000]
139         if verbose==True:
140             print("Features with high VIF:")
141         if verbose==True:
142             print(max_col)
143         if verbose==True:
144             print("Dropping", feature, "...")
145         X_df.drop([feature], axis=1, inplace=True)
146
147     vif = pd.DataFrame()
148     vif["Features"] = X_df.columns
149     vif["VIF Factor"]=[variance_inflation_factor(X_df.values, i) ...
        for i in range(X_df.shape[1])]
150     vif.sort_values(by=["VIF Factor"], ascending=False, inplace=...
        True)
151
152     if vif[vif["VIF Factor"]>1000].size==0:
153         if verbose==True:
154             print("The number of fetaures that have high VIF has ...
                    become zero!")
155         if verbose==True:
156             print("Preventing all further removals :)")
157         break
158     if verbose==True:
159         print("Done!")
160
161     df_new = X_df
162     df_new.head()
163
164     df_save = df_new.copy()
165     df_save["Next_Tmin"] = df["Next_Tmin"]
166     df_save["Next_Tmax"] = df["Next_Tmax"]
167     df_save.to_csv("../datasets/processed.csv")
168     # df_save.to_csv("processed.csv")
169     if verbose==True:

```



```

170         print("Saved the data as processed.csv")
171
172     return 1

```

3.1.2 Gaussian Basis

The helper class used to perform GaussianBasis are as follows:

```

1  import os
2  import missingno
3  import numpy as np
4  import pandas as pd
5  import seaborn as sns
6  from tqdm import tqdm
7  import matplotlib.pyplot as plt
8  from preprocess import PreProcess
9  from sklearn.cluster import KMeans
10 from mpl_toolkits.mplot3d import Axes3D
11 from statsmodels.stats.outliers_influence import variance_inflation_factor
12 from sklearn.cluster import KMeans
13
14 class GaussianBasis():
15     def __init__(self):
16         pass
17
18     def fit_grid(self, df_new, df_save, num_clusters, regularization=None, ...
19                 lambda_list=[0], verbose=False, show=False):
20         n_clu_hist = []
21         lambda_hist = []
22         error1_hist = []
23         error2_hist = []
24         sse_dict = {}
25
26         if regularization == None or regularization=="L2":
27             print("Running K-Means...", end="")
28             for n_clu in tqdm(num_clusters):
29                 kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(...
30                                df_new.to_numpy())
31                 sse_dict[n_clu] = kmeans.inertia_
32
33                 mean_centers = kmeans.cluster_centers_
34                 corresponding_center = mean_centers[kmeans.labels_,:]
35
36                 X = df_new.to_numpy()
37                 distance = np.linalg.norm(X-corresponding_center, axis=1)
38                 var = np.var(distance)*distance.size
39
40                 phi = np.ones((X.shape[0], 1))
41                 for i in range(n_clu):
42                     phi = np.append(phi, np.exp(-np.linalg.norm(X-...
43                                mean_centers[i,:], axis=1)**2/var).reshape(-1,1), ...
44                                axis=1)
45
46                 for lambda in lambda_list:
47                     n_clu_hist.append(n_clu)
48                     lambda_hist.append(lambda)
49
50                 W1 = (np.linalg.inv(phi.T @ phi + lambda*np.identity(phi...
51                        .shape[1])) @ phi.T) @ df_save["Next_Tmin"]

```

```

47     W2 = (np.linalg.inv(phi.T @ phi + lambda*np.identity(phi...
         .shape[1])) @ phi.T) @ df_save["Next_Tmax"]
48     W1 = W2.reshape(-1,1)
49     W1 = W2.reshape(-1,1)
50     pred1 = phi @ W1
51     pred2 = phi @ W2
52
53     plt.figure(figsize=[16,9])
54     plt.title("Clusters: "+str(n_clu))
55     plt.subplot(1, 2, 1)
56     plt.hist(pred1, alpha=0.5)
57     plt.hist(df_save["Next_Tmin"], alpha=0.5)
58     plt.title("Next_Tmin; Clusters: "+str(n_clu))
59     plt.grid()
60     plt.subplot(1, 2, 2)
61     plt.plot(df_save["Next_Tmin"], pred1, ".")
62     plt.plot(df_save["Next_Tmin"], df_save["Next_Tmin"], '....')
63
64     plt.title("Next_Tmin; Clusters: "+str(n_clu))
65     plt.grid()
66     if show:
67         plt.show()
68     fname = "fit_1_k_"+str(n_clu)+"_lambda_"+str(lambda)+"....
        png"
69     plt.savefig("images/q3/"+fname)
70
71     plt.figure(figsize=[16,9])
72     plt.title("Clusters: "+str(n_clu))
73     plt.subplot(1, 2, 1)
74     plt.hist(pred2, alpha=0.5, label="Predicted")
75     plt.hist(df_save["Next_Tmax"], alpha=0.5, label="True")
76     plt.title("Next_Tmax; Clusters: "+str(n_clu))
77     plt.grid()
78     plt.subplot(1, 2, 2)
79     plt.plot(df_save["Next_Tmax"], pred2, ".")
80     plt.plot(df_save["Next_Tmax"], df_save["Next_Tmax"], '....')
81
82     plt.title("Clusters: "+str(n_clu))
83     plt.grid()
84     if show:
85         plt.show()
86     fname = "fit_2_k_"+str(num_clusters)+"_lambda_"+str(...
        lambda)+".png"
87     plt.savefig("images/q3/"+fname)
88
89     error1 = np.linalg.norm(df_save["Next_Tmin"].to_numpy()...
        .reshape(-1,1)-pred1)
90     error2 = np.linalg.norm(df_save["Next_Tmin"].to_numpy()...
        .reshape(-1,1)-pred2)
91
92     error1_hist.append(error1)
93     error2_hist.append(error2)
94     print("Done!")
95
96     df_result = pd.DataFrame()
97     df_result["Cluster"] = n_clu_hist
98     df_result["Lambda"] = lambda_hist
99     df_result["Error 1"] = error1_hist
100    df_result["Error 2"] = error2_hist
101    df_result.to_csv("../datasets/q3_gridsearch.csv")

```

```

100
101         return df_result, sse_dict

```

3.2 Tikhonov Regularization

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

```

1 #####
2 from IPython import get_ipython
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from mpl_toolkits.mplot3d import Axes3D
7 from sklearn.cluster import KMeans
8 import seaborn as sns
9 get_ipython().run_line_magic('matplotlib', 'inline')
10
11
12 #####
13 dataset2 = pd.read_csv('function1_2d.csv', index_col = 0)
14
15
16 #####
17 dataset2['y']
18
19
20 #####
21 num_clusters = [1]
22 num_clusters.extend(range(2,100,10))
23 #num_clusters.extend(range(15, 31, 5))
24 #num_clusters.extend(range(40, 101, 10))
25
26 sse_list = []
27 label_list = []
28 cluster_centers_list = []
29 error_list = []
30
31 for n_clu in num_clusters:
32     kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(dataset2....
33         to_numpy())
34     sse_list.append(kmeans.inertia_)
35     label_list.append(kmeans.labels_)
36     cluster_centers_list.append(kmeans.cluster_centers_)
37
38     mean_centers = cluster_centers_list[-1]
39     # print("Mean shape:", mean_centers.shape)
40     corresponding_center = mean_centers[label_list[-1],:]
41
42     X = dataset2.to_numpy()
43     distance = np.linalg.norm(X-corresponding_center, axis=1)
44     var = np.var(distance)*distance.size
45
46     phi = np.ones((X.shape[0], 1))
47     for i in range(n_clu):
48         A = X-mean_centers[i,:]
49         # print("A shape:", A.shape)
50         A = np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var)
51         # print("A shape:", A.shape)

```

```

51     phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,:], ...
52                     axis=1)**2/var).reshape(-1,1), axis=1)
53
54     lambda = 0
55     W1 = (np.linalg.inv(phi.T @ phi + lambda*np.identity(phi.shape[1])) @ ...
56           phi.T) @ dataset2["y"]
57     W1 = W1.reshape(-1,1)
58     pred = phi @ W1
59
60     plt.figure(figsize=[12,8])
61     plt.title("Clusters: "+str(n_clu))
62     plt.subplot(1, 2, 1)
63     plt.hist(pred, alpha=0.5)
64     plt.hist(dataset2["y"], alpha=0.5)
65     plt.title("Clusters: "+str(n_clu))
66     plt.grid()
67     plt.subplot(1, 2, 2)
68     plt.plot(dataset2["y"], pred, ".")
69     plt.plot(dataset2["y"], dataset2["y"], '.')
70     plt.title("Clusters: "+str(n_clu))
71     plt.grid()
72     plt.savefig('')
73     plt.show()
74     error = np.linalg.norm(dataset2["y"].to_numpy().reshape(-1,1)-pred)
75     error_list.append(error)
76
77     #####
78     plt.figure(figsize=[12,8])
79     plt.subplot(1,2,1)
80     plt.plot(num_clusters, sse_list)
81     plt.xlabel("Number of Clusters")
82     plt.ylabel("SSE")
83     plt.title("Knee Plot for determining the number of clusters")
84     plt.grid()
85     plt.subplot(1,2,2)
86     plt.plot(num_clusters, error_list)
87     plt.xlabel("Number of Clusters")
88     plt.ylabel("SSE")
89     plt.title(("L2 Error for fit"))
90     plt.grid()
91     plt.show()
92
93     #####
94     error_list = np.array(error_list)
95     df_error = pd.DataFrame({"Clusters":num_clusters, "Error":error_list})
96     df_error.sort_values(by=["Error"], ascending=True, inplace=True)
97     df_error
98
99
100    #####
101    def create_datasets(data,train_size,cv_size):
102        data.sample(frac=1).reset_index(drop=True)
103        data_train=data[0:train_size]
104        data_cv=data[train_size:train_size+cv_size]
105        data_test=data[cv_size+train_size:]
106        return(data_train,data_cv,data_test)
107
108

```

```

109
110 #####
111 lambda_list = [0.01,0.1,1,5,10,50,100]
112 trDS2, cvDS2, tDS2 = create_datasets(dataset2,1400,400)
113
114
115 #####
116 def f(ds,l,n_clu):
117     sse_list = []
118     label_list = []
119     cluster_centers_list = []
120
121     kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(ds.to_numpy())
122     sse_list.append(kmeans.inertia_)
123     label_list.append(kmeans.labels_)
124     cluster_centers_list.append(kmeans.cluster_centers_)
125
126     mean_centers = cluster_centers_list[-1]
127     # print("Mean shape:", mean_centers.shape)
128     corresponding_center = mean_centers[label_list[-1],:]
129
130     X = ds.to_numpy()
131     distance = np.linalg.norm(X-corresponding_center, axis=1)
132     var = np.var(distance)*distance.size
133
134     phi = np.ones((X.shape[0], 1))
135     for i in range(n_clu):
136         A = X-mean_centers[i,: ]
137         # print("A shape:", A.shape)
138         A = np.exp(-np.linalg.norm(X-mean_centers[i,: ], axis=1)**2/var)
139         # print("A shape:", A.shape)
140         phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,: ], ...
141                             axis=1)**2/var).reshape(-1,1), axis=1)
142         #lmbda = 1
143         return(phi,mean_centers,var)
144 #####...
145     [markdown]
146     # W1 = (np.linalg.inv(phi.T @ phi + l*np.identity(phi.shape[1])) @ phi...
147     T) @ dataset2["y"]
148     # W1 = W1.reshape(-1,1)
149     # pred = phi @ W1
150     # return(W1,)
151     # error = np.linalg.norm(dataset2["y"].to_numpy().reshape(-1,1)-pred)
152     # error_list.append(error)
153
154 #####
155
156 #####
157 error_tr = []
158 error_cv = []
159 error_t = []
160 for l in range(len(lambda_list)):
161     phi_tr = f(trDS2,l,optClds2)[0]
162     phi_cv = f(cvDS2,l,optClds2)[0]
163     phi_t = f(tDS2,l,optClds2)[0]
164     w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) ...
165         @ phi_tr.T) @ trDS2["y"]

```

```

165     pred_cv = phi_cv @ w
166     pred_t = phi_t @ w
167     pred_tr = phi_tr @ w
168     error_cv.append(np.linalg.norm(cvDS2["y"].to_numpy().reshape(-1,1)-...
169         pred_cv))
169     error_t.append(np.linalg.norm(tDS2["y"].to_numpy().reshape(-1,1)-pred_t...
170         ))
170     error_tr.append(np.linalg.norm(trDS2["y"].to_numpy().reshape(-1,1)-...
171         pred_tr))
171
172
173     #####
174     pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
175         Lambda", "RMSE Train","RMSE CV","RMSE test"])
175
176
177     #####
178     plt.plot(lambda_list,error_cv)
179
180
181     #####
182     phi_tr = f(trDS2,l,optClds2)[0]
183     phi_t = f(tDS2,l,optClds2)[0]
184     w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.01*np.identity(phi_tr.shape[1])) @...
185         phi_tr.T @ trDS2["y"])
185     pred_t = phi_t @ w
186     pred_tr = phi_tr @ w
187
188
189     #####
190     plt.scatter(trDS2.iloc[:,2],pred_tr)
191     plt.xlabel("Target output,training data")
192     plt.ylabel("Model output")
193     plt.title("Scatter plot of target vs model output for linear regression in ...
194         gaussian basis and quadratic regularization")
194     plt.savefig("scatter_ds2quad.png")
195     plt.show()
196
197
198     #####
199     plt.scatter(cvDS2.iloc[:,2],pred_cv)
200
201
202     #####
203     plt.scatter(tDS2.iloc[:,2],pred_t)
204     plt.xlabel("Target output,test data")
205     plt.ylabel("Model output")
206     plt.title("Scatter plot of target vs model output for linear regression in ...
207         gaussian basis and quadratic regularization")
207     plt.savefig("scatter_ds2quadtest.png")
208     plt.show()
209
210
211     #####
212
213
214
215     #####
216     def tikhanov_reg(phi,mu,sigma,l):
217         K = len(mu)

```

```

218     phiT = np.zeros((K+1,K+1))
219     phiT[0,0] = 1
220     for i in range(1,K+1):
221         for j in range(1,K+1):
222             phiT[i,j] = np.exp(-(np.linalg.norm(mu[i-1]-mu[j-1])**2)/sigma...
                **2)
223     #l = 300
224     #print(phiT.shape)
225     pinv = np.linalg.inv(phi.T @ phi+l*phiT) @ phi.T
226     return(pinv)
227
228
229 #####
230 lambda_list = [0,0.01,0.1,1,5]#,10,50,100]
231
232
233 #####
234 error_tr = []
235 error_cv = []
236 error_t = []
237 for l in range(len(lambda_list)):
238     #for l in [1]:
239         phi_tr,mu_list,sig = f(trDS2,l,optClds2)
240         phi_cv = f(cvDS2,l,optClds2)[0]
241         phi_t = f(tDS2,l,optClds2)[0]
242         #print(phi_tr.shape)
243         #print(mu_list.shape)
244         tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
245         #print("tikh shape ", tikh.shape)
246         #print("phi_tr shape ", phi_tr.shape)
247         #print("trDS2_y shape ", trDS2["y"].shape)
248
249         w = tikh @ trDS2["y"]
250         pred_cv = phi_cv @ w
251         pred_t = phi_t @ w
252         pred_tr = phi_tr @ w
253         error_cv.append(np.linalg.norm(cvDS2["y"].to_numpy().reshape(-1,1)-...
            pred_cv))
254         error_t.append(np.linalg.norm(tDS2["y"].to_numpy().reshape(-1,1)-pred_t...
            ))
255         error_tr.append(np.linalg.norm(trDS2["y"].to_numpy().reshape(-1,1)-...
            pred_tr))
256
257
258 #####
259 pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])
260
261
262 #####
263 plt.plot(lambda_list,error_cv)
264 plt.title('Erms on cross-validation data vs regularization factor')
265 plt.xlabel('lambda')
266 plt.ylabel('rmse error')
267
268
269 #####
270 plt.hist(pred_tr, alpha=0.5)
271 plt.hist(trDS2["y"], alpha=0.5)
272

```

```

273
274 #####
275 error_list = np.array(error_list)
276 df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
277 df_error.sort_values(by=["Error"], ascending=True, inplace=True)
278 df_error
279
280
281 #####
282 l = 0.01
283 phi_tr = f(trDS2,l,optClds2)[0]
284 phi_t = f(tDS2,l,optClds2)[0]
285 tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
286 w = tikh @ trDS2["y"]
287 w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
      phi_tr.T) @ trDS2["y"]
288 pred_t = phi_t @ w
289 pred_tr = phi_tr @ w
290
291
292 #####
293 plt.scatter(trDS2.iloc[:,2],pred_tr)
294 plt.xlabel("Target output,training data")
295 plt.ylabel("Model output")
296 plt.title("Scatter plot of target vs model output for linear regression in ...
      gaussian basis and Tikhonov regularization")
297 plt.savefig("scatter_ds2tikhtr.png")
298 plt.show()
299
300
301 #####
302 plt.scatter(tDS2.iloc[:,2],pred_t)
303 plt.xlabel("Target output,test data")
304 plt.ylabel("Model output")
305 plt.title("Scatter plot of target vs model output for linear regression in ...
      gaussian basis and Tikhonov regularization")
306 plt.savefig("scatter_ds2tikhtest.png")
307 plt.show()
308
309
310 #####
311
312
313
314 #####
315 df_new = pd.read_csv("processed_biasclean.csv",index_col = 0)
316
317
318 #####
319 df_new.head()
320
321
322 #####
323 trDS3, cvDS3, tDS3 = create_datasets(df_new,1400,400)
324
325
326 #####
327 num_clusters = [1]
328 num_clusters.extend(range(2,10))
329 num_clusters.extend(range(15, 31, 5))

```



```

330 num_clusters.extend(range(40, 101, 10))
331
332 sse_list = []
333 label_list = []
334 cluster_centers_list = []
335 error_list = []
336
337 for n_clu in num_clusters:
338     kmeans = KMeans(n_clusters=n_clu, random_state=42).fit(df_new.to_numpy...
339         ())
340     sse_list.append(kmeans.inertia_)
341     label_list.append(kmeans.labels_)
342     cluster_centers_list.append(kmeans.cluster_centers_)
343
344     mean_centers = cluster_centers_list[-1]
345     # print("Mean shape:", mean_centers.shape)
346     corresponding_center = mean_centers[label_list[-1],:]
347
348     X = df_new.to_numpy()
349     distance = np.linalg.norm(X-corresponding_center, axis=1)
350     var = np.var(distance)*distance.size
351
352     phi = np.ones((X.shape[0], 1))
353     for i in range(n_clu):
354         A = X-mean_centers[i,:].
355         # print("A shape:", A.shape)
356         A = np.exp(-np.linalg.norm(X-mean_centers[i,:], axis=1)**2/var)
357         # print("A shape:", A.shape)
358         phi = np.append(phi, np.exp(-np.linalg.norm(X-mean_centers[i,:], ...
359             axis=1)**2/var).reshape(-1,1), axis=1)
360
361     lambda = 0
362     W1 = (np.linalg.inv(phi.T @ phi + lambda*np.identity(phi.shape[1])) @ ...
363         phi.T) @ df_new["Next_Tmin"]
364     W1 = W1.reshape(-1,1)
365     pred = phi @ W1
366
367     plt.figure(figsize=[12,8])
368     plt.title("Clusters: "+str(n_clu))
369     plt.subplot(1, 2, 1)
370     plt.hist(pred, alpha=0.5)
371     plt.hist(df_new["Next_Tmin"], alpha=0.5)
372     plt.title("Clusters: "+str(n_clu))
373     plt.grid()
374     plt.subplot(1, 2, 2)
375     plt.plot(df_new["Next_Tmin"], pred, ".")
376     plt.plot(df_new["Next_Tmin"], df_new["Next_Tmin"], '.')
377     plt.title("Clusters: "+str(n_clu))
378     plt.grid()
379     plt.show()
380     error = np.linalg.norm(df_new["Next_Tmin"].to_numpy().reshape(-1,1)-...
381         pred)
382     error_list.append(error)
383
384     #####
385     plt.figure(figsize=[12,8])
386     plt.subplot(1,2,1)
387     plt.plot(num_clusters, sse_list)
388     plt.xlabel("Number of Clusters")

```

```

386 plt.ylabel("SSE")
387 plt.title("Knee Plot for determining the number of clusters")
388 plt.grid()
389 plt.subplot(1,2,2)
390 plt.plot(num_clusters, error_list)
391 plt.xlabel("Number of Clusters")
392 plt.ylabel("SSE")
393 plt.title(("L2 Error for fit"))
394 plt.grid()
395 plt.show()
396
397
398 #####
399 error_list = np.array(error_list)
400 df_error = pd.DataFrame({"Clusters":num_clusters, "Error":error_list})
401 df_error.sort_values(by=["Error"], ascending=True, inplace=True)
402 df_error
403
404
405 #####
406 optClds3 = 9
407
408
409 #####
410 lambda_list = [0.01,0.1,1,5,10]
411
412
413 #####
414 error_tr = []
415 error_cv = []
416 error_t = []
417 for l in range(len(lambda_list)):
418     phi_tr = f(trDS3,l,optClds3)[0]
419     phi_cv = f(cvDS3,l,optClds3)[0]
420     phi_t = f(tDS3,l,optClds3)[0]
421     w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) ...
         @ phi_tr.T) @ trDS3['Next_Tmin']
422     pred_cv = phi_cv @ w
423     pred_t = phi_t @ w
424     pred_tr = phi_tr @ w
425     error_cv.append(np.linalg.norm(cvDS3['Next_Tmin'].to_numpy().reshape...
         (-1,1)-pred_cv))
426     error_t.append(np.linalg.norm(tDS3['Next_Tmin'].to_numpy().reshape...
         (-1,1)-pred_t))
427     error_tr.append(np.linalg.norm(trDS3['Next_Tmin'].to_numpy().reshape...
         (-1,1)-pred_tr))
428
429
430 #####
431 pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
         Lambda", "RMSE Train","RMSE CV","RMSE test"])
432
433
434 #####
435 plt.plot(lambda_list,error_cv)
436 plt.title('Error on cross-validation data vs regularization factor, ...
         Quadratic regularization')
437 plt.xlabel('lambda')
438 plt.ylabel('rmse error')
439

```

```

440
441 #####
442 error_list = np.array(error_list)
443 df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
444 df_error.sort_values(by=["Error"], ascending=True, inplace=True)
445 df_error
446
447
448 #####
449 l = .1
450 phi_tr = f(trDS3,l,optClds3)[0]
451 phi_t = f(tDS3,l,optClds3)[0]
452 w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ ...
      phi_tr.T) @ trDS3['Next_Tmin']
453 pred_t = phi_t @ w
454 pred_tr = phi_tr @ w
455
456
457 #####
458 plt.scatter(trDS3.iloc[:,17],pred_tr)
459 plt.xlabel("Target output,train data")
460 plt.ylabel("Model output")
461 plt.title("Scatter plot of target vs model with quadratic regularization, ...
      for Next_Tmin")
462 plt.savefig("scatter_ds3quadtrainT_min.png")
463 plt.show()
464
465
466 #####
467 plt.scatter(tDS3.iloc[:,17],pred_t)
468 plt.xlabel("Target output,test data")
469 plt.ylabel("Model output")
470 plt.title("Scatter plot of target vs model output with quadratic ...
      regularization, for Next_Tmin")
471 plt.savefig("scatter_ds3quadtestT_min.png")
472 plt.show()
473
474
475 #####
476 error_tr = []
477 error_cv = []
478 error_t = []
479 for l in range(len(lambda_list)):
480     phi_tr = f(trDS3,l,optClds3)[0]
481     phi_cv = f(cvDS3,l,optClds3)[0]
482     phi_t = f(tDS3,l,optClds3)[0]
483     w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ ...
          phi_tr.T) @ trDS3['Next_Tmax']
484     pred_cv = phi_cv @ w
485     pred_t = phi_t @ w
486     pred_tr = phi_tr @ w
487     error_cv.append(np.linalg.norm(cvDS3['Next_Tmax'].to_numpy().reshape...
          (-1,1)-pred_cv))
488     error_t.append(np.linalg.norm(tDS3['Next_Tmax'].to_numpy().reshape...
          (-1,1)-pred_t))
489     error_tr.append(np.linalg.norm(trDS3['Next_Tmax'].to_numpy().reshape...
          (-1,1)-pred_tr))
490
491
492 #####

```

```

493 pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
    Lambda", "RMSE Train","RMSE CV","RMSE test"])
494
495
496 #####
497 plt.plot(lambda_list,error_cv)
498 plt.title('Error on cross-validation data vs regularization factor, ...
    Quadratic regularization')
499 plt.xlabel('lambda')
500 plt.ylabel('rmse error')
501
502
503 #####
504 error_list = np.array(error_list)
505 df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
506 df_error.sort_values(by=["Error"], ascending=True, inplace=True)
507 df_error
508
509
510 #####
511 l = 1
512 phi_tr = f(trDS3,l,optClds3)[0]
513 phi_t = f(tDS3,l,optClds3)[0]
514 w = (np.linalg.inv(phi_tr.T @ phi_tr + l*np.identity(phi_tr.shape[1])) @ ...
    phi_tr.T) @ trDS3['Next_Tmax']
515 pred_t = phi_t @ w
516 pred_tr = phi_tr @ w
517
518
519 #####
520 plt.scatter(trDS3.iloc[:,18],pred_tr)
521 plt.xlabel("Target output,train data")
522 plt.ylabel("Model output")
523 plt.title("Scatter plot of target vs model with quadratic regularization, ...
    for Next_Tmax")
524 plt.savefig("scatter_ds3quadtrainT_max.png")
525 plt.show()
526
527
528 #####
529 plt.scatter(tDS3.iloc[:,18],pred_t)
530 plt.xlabel("Target output,test data")
531 plt.ylabel("Model output")
532 plt.title("Scatter plot of target vs model with quadratic regularization, ...
    for Next_Tmax")
533 plt.savefig("scatter_ds3quadtestT_max.png")
534 plt.show()
535
536
537 #####
538 #Tikhonov reg for "Next_Tmin"
539
540 error_tr = []
541 error_cv = []
542 error_t = []
543 for l in range(len(lambda_list)):
544     #for l in [1]:
545         phi_tr,mu_list,sig = f(trDS3,l,optClds3)
546         phi_cv = f(cvDS2,l,optClds3)[0]
547         phi_t = f(tDS2,l,optClds3)[0]

```

```

548     #print(phi_tr.shape)
549     #print(mu_list.shape)
550     tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
551     #print("tikh shape ", tikh.shape)
552     #print("phi_tr shape ", phi_tr.shape)
553     #print("trDS2_y shape ", trDS2["y"].shape)
554
555     w = tikh @ trDS3['Next_Tmin']
556     pred_cv = phi_cv @ w
557     pred_t = phi_t @ w
558     pred_tr = phi_tr @ w
559     error_cv.append(np.linalg.norm(cvDS3['Next_Tmin'].to_numpy().reshape...
560                             (-1,1)-pred_cv))
561     error_t.append(np.linalg.norm(tDS3['Next_Tmin'].to_numpy().reshape...
562                             (-1,1)-pred_t))
563     error_tr.append(np.linalg.norm(trDS3['Next_Tmin'].to_numpy().reshape...
564                             (-1,1)-pred_tr))
565
566     #####
567     pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
568         Lambda", "RMSE Train","RMSE CV","RMSE test"])
569
570     #####
571     plt.plot(lambda_list,error_cv)
572     plt.title('Error on cross-validation data vs regularization factor, ...
573         tikhonov regularization for Dataset 3')
574     plt.xlabel('lambda')
575     plt.ylabel('rmse error')
576
577     #####
578     plt.hist(pred_tr, alpha=0.5)
579     plt.hist(trDS3["Next_Tmin"], alpha=0.5)
580
581     #####
582     error_list = np.array(error_list)
583     df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
584     df_error.sort_values(by=["Error"], ascending=True, inplace=True)
585     df_error
586
587     #####
588     l = .1
589     phi_tr = f(trDS3,l,optClds3)[0]
590     phi_t = f(tDS3,l,optClds3)[0]
591     tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
592     w = tikh @ trDS3["Next_Tmin"]
593     w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
594         phi_tr.T) @ trDS3["Next_Tmin"]
595     pred_t = phi_t @ w
596     pred_tr = phi_tr @ w
597
598     #####
599     plt.scatter(trDS3.iloc[:,17],pred_tr)
600     plt.xlabel("Target output,train data")
601     plt.ylabel("Model output")

```

```

602 plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
        for Next_Tmin")
603 plt.savefig("scatter_ds3tikhtrainT_min.png")
604 plt.show()
605
606
607 #####
608 plt.scatter(tDS3.iloc[:,17],pred_t)
609 plt.xlabel("Target output,test data")
610 plt.ylabel("Model output")
611 plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
        for Next_Tmin")
612 plt.savefig("scatter_ds3tikhtrainT_min.png")
613 plt.show()
614
615
616 #####
617 #Tikhonov reg for "Next_Tmax"
618 error_tr = []
619 error_cv = []
620 error_t = []
621 for l in range(len(lambda_list)):
622     #for l in [1]:
623         phi_tr,mu_list,sig = f(trDS3,l,optClds3)
624         phi_cv = f(cvDS2,l,optClds3)[0]
625         phi_t = f(tDS2,l,optClds3)[0]
626         #print(phi_tr.shape)
627         #print(mu_list.shape)
628         n = len(trDS3)
629         tikh = tikhanov_reg(phi_tr,mu_list,sig,l)
630         #print("tikh shape ", tikh.shape)
631         #print("phi_tr shape ", phi_tr.shape)
632         #print("trDS2_y shape ", trDS2["y"].shape)
633
634         w = tikh @ trDS3['Next_Tmax']
635         pred_cv = phi_cv @ w
636         pred_t = phi_t @ w
637         pred_tr = phi_tr @ w
638         error_cv.append(np.linalg.norm(cvDS3['Next_Tmax'].to_numpy().reshape...
            (-1,1)-pred_cv))
639         error_t.append(np.linalg.norm(tDS3['Next_Tmax'].to_numpy().reshape...
            (-1,1)-pred_t))
640         error_tr.append(np.linalg.norm(trDS3['Next_Tmax'].to_numpy().reshape...
            (-1,1)-pred_tr))
641
642
643 #####
644 pd.DataFrame(list(zip(lambda_list,error_tr,error_cv,error_t)),columns=["...
        Lambda", "RMSE Train","RMSE CV","RMSE test"])
645
646
647 #####
648 plt.plot(lambda_list,error_cv)
649 plt.title('Error on cross-validation data vs regularization factor, ...
        tikhonov regularization for Dataset 3, for T_max')
650 plt.xlabel('lambda')
651 plt.ylabel('rmse error')
652
653
654 #####

```

```

655 plt.hist(pred_tr, alpha=0.5)
656 plt.hist(trDS3["Next_Tmax"], alpha=0.5)
657
658
659 #####
660 error_list = np.array(error_list)
661 df_error = pd.DataFrame({"lambda":lambda_list, "Error":error_cv})
662 df_error.sort_values(by=["Error"], ascending=True, inplace=True)
663 df_error
664
665
666 #####
667 l = .01
668 phi_tr = f(trDS3,l,optCls3)[0]
669 phi_t = f(tDS3,l,optCls3)[0]
670 tikh = tikhanov_reg(phi_tr,mu_list,sig,0.01)
671 w = tikh @ trDS3["Next_Tmax"]
672 w = (np.linalg.inv(phi_tr.T @ phi_tr + 0.1*np.identity(phi_tr.shape[1])) @ ...
        phi_tr.T) @ trDS3["Next_Tmax"]
673 pred_t = phi_t @ w
674 pred_tr = phi_tr @ w
675
676
677 #####
678 plt.scatter(trDS3.iloc[:,18],pred_tr)
679 plt.xlabel("Target output,train data")
680 plt.ylabel("Model output")
681 plt.title("Scatter plot of target vs model output with Tikhonov ...
        regularization, for Next_Tmax")
682 plt.savefig("scatter_ds3tikhttrainT_max.png")
683 plt.show()
684
685
686 #####
687 plt.scatter(trDS3.iloc[:,18],pred_tr)
688 plt.xlabel("Target output,test data")
689 plt.ylabel("Model output")
690 plt.title("Scatter plot of target vs model with Tikhonov regularization, ...
        for Next_Tmax")
691 plt.savefig("scatter_ds3tikhttestT_min.png")
692 plt.show()
693
694
695 #####

```