

ASSIGNMENT 3

CS5691 Pattern Recognition and Machine Learning

CS5691 Assignment 3 Code

Team Members:

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

Indian Institute of Technology, Madras



Contents

1	Dataset 1A	2
1.1	Perceptron	2
1.1.1	Helper Function	6
1.1.1.1	Perceptron	6
1.2	MLFFNN	8
1.2.1	Helper Function	10
1.2.1.1	Gridsearch	11
1.3	Linear SVM	13
2	Dataset 1B	18
2.1	MLFFNN	18
2.1.1	Helper Function	23
2.1.1.1	Gridsearch	23
2.2	Non-Linear SVM	25
3	Dataset 2A	31
3.1	MLFFNN	31
3.1.1	Helper Function	33
3.1.1.1	Data Consolidation	33
3.1.1.2	Gridsearch	34
3.2	Gaussian-kernel SVM	36

1 Dataset 1A

1.1 Perceptron

The code written for analyzing Dataset 1A, using Perceptron model is as follows:

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import numpy as np
8  import pandas as pd
9  import tensorflow as tf
10 import matplotlib.pyplot as plt
11 import random
12 from sklearn.model_selection import train_test_split
13 from sklearn.metrics import confusion_matrix
14 import seaborn as sns
15
16 from perceptron import Perceptron
17
18 import warnings
19 warnings.filterwarnings("ignore")
20
21
22 # In[2]:
23
24
25 get_ipython().run_line_magic('matplotlib', 'inline')
26
27
28 # In[3]:
29
30
31 plt.rcParams["font.size"] = 18
32 plt.rcParams["axes.grid"] = True
33 plt.rcParams["figure.figsize"] = 12,8
34 plt.rcParams['font.serif'] = "Cambria"
35 plt.rcParams['font.family'] = "serif"
36
37
38 # In[4]:
39
40
41 ds1_train = pd.read_csv("train1.csv",header = None)
42 ds1_test = pd.read_csv("dev1.csv", header=None)
43 ds1_train.insert(0,"theta",pd.Series(np.ones(len(ds1_train))))
44 ds1_test.insert(0,"theta",pd.Series(np.ones(len(ds1_test))))
45 cv, test = train_test_split(ds1_test, test_size = 0.3, random_state = 0)
46
47
48 # In[5]:
49
50
51 fil1 = ds1_train[2] == 0.
52 fil2 = ds1_train[2] == 1.
53 ds0_1 = ds1_train.where(fil1 | fil2).dropna()
54 fil1 = ds1_train[2] == 0.
55 fil2 = ds1_train[2] == 2.
56 ds0_2 = ds1_train.where(fil1 | fil2).dropna()
57 fil1 = ds1_train[2] == 0.
58 fil2 = ds1_train[2] == 3.
59 ds0_3 = ds1_train.where(fil1 | fil2).dropna()
```

```

60 fil1 = ds1_train[2] == 1.
61 fil2 = ds1_train[2] == 2.
62 ds1_2 = ds1_train.where(fil1 | fil2).dropna()
63 fil1 = ds1_train[2] == 1.
64 fil2 = ds1_train[2] == 3.
65 ds1_3 = ds1_train.where(fil1 | fil2).dropna()
66 fil1 = ds1_train[2] == 2.
67 fil2 = ds1_train[2] == 3.
68 ds2_3 = ds1_train.where(fil1 | fil2).dropna()
69
70 fil1 = cv[2] == 0.
71 fil2 = cv[2] == 1.
72 cv0_1 = cv.where(fil1 | fil2).dropna()
73 fil1 = cv[2] == 0.
74 fil2 = cv[2] == 2.
75 cv0_2 = cv.where(fil1 | fil2).dropna()
76 fil1 = cv[2] == 0.
77 fil2 = cv[2] == 3.
78 cv0_3 = cv.where(fil1 | fil2).dropna()
79 fil1 = cv[2] == 1.
80 fil2 = cv[2] == 2.
81 cv1_2 = cv.where(fil1 | fil2).dropna()
82 fil1 = cv[2] == 1.
83 fil2 = cv[2] == 3.
84 cv1_3 = cv.where(fil1 | fil2).dropna()
85 fil1 = cv[2] == 2.
86 fil2 = cv[2] == 3.
87 cv2_3 = cv.where(fil1 | fil2).dropna()
88
89 fil1 = test[2] == 0.
90 fil2 = test[2] == 1.
91 test0_1 = test.where(fil1 | fil2).dropna()
92 fil1 = test[2] == 0.
93 fil2 = test[2] == 2.
94 test0_2 = test.where(fil1 | fil2).dropna()
95 fil1 = test[2] == 0.
96 fil2 = test[2] == 3.
97 test0_3 = test.where(fil1 | fil2).dropna()
98 fil1 = test[2] == 1.
99 fil2 = test[2] == 2.
100 test1_2 = test.where(fil1 | fil2).dropna()
101 fil1 = test[2] == 1.
102 fil2 = test[2] == 3.
103 test1_3 = test.where(fil1 | fil2).dropna()
104 fil1 = test[2] == 2.
105 fil2 = test[2] == 3.
106 test2_3 = test.where(fil1 | fil2).dropna()
107
108
109 # In[6]:
110
111
112 ds0_1[2] = ds0_1[2].replace([0.,1],[-1,1])
113 ds0_2[2] = ds0_2[2].replace([0.,2],[-1,1])
114 ds0_3[2] = ds0_3[2].replace([0.,3],[-1,1])
115 ds1_2[2] = ds1_2[2].replace([1,2],[-1,1])
116 ds1_3[2] = ds1_3[2].replace([1,3],[-1,1])
117 ds2_3[2] = ds2_3[2].replace([2,3],[-1,1])
118
119
120 cv0_1[2] = cv0_1[2].replace([0.,1],[-1,1])
121 cv0_2[2] = cv0_2[2].replace([0.,2],[-1,1])
122 cv0_3[2] = cv0_3[2].replace([0.,3],[-1,1])
123 cv1_2[2] = cv1_2[2].replace([1,2],[-1,1])
124 cv1_3[2] = cv1_3[2].replace([1,3],[-1,1])
125 cv2_3[2] = cv2_3[2].replace([2,3],[-1,1])
126
127 test0_1[2] = test0_1[2].replace([0.,1],[-1,1])
128 test0_2[2] = test0_2[2].replace([0.,2],[-1,1])

```

```

129 test0_3[2] = test0_3[2].replace([0.,3],[-1,1])
130 test1_2[2] = test1_2[2].replace([1,2],[-1,1])
131 test1_3[2] = test1_3[2].replace([1,3],[-1,1])
132 test2_3[2] = test2_3[2].replace([2,3],[-1,1])
133
134
135 # In[7]:
136
137
138 def hyperparameter_testing(train_dat, cv_dat):
139     eta_range = [0.001,0.005,0.01,0.05,0.1,1,5,10,100]
140     acc_train = []
141     acc_cv = []
142     for eta in eta_range:
143         model = Perceptron(train_dat, learning_rate = eta)
144         model.train()
145         acc_train.append(model.accuracy(train_dat))
146         acc_cv.append(model.accuracy(cv_dat))
147     dictionary = {"Hyperparameter": eta_range, "Training Accuracy":acc_train,"CV ...
148                 Accuracy":acc_cv}
149     df = pd.DataFrame(dictionary)
150     max_val = np.argmax(np.array(acc_cv))
151     print("Maximum accuracy on CV is achieved for the learning rate value: " , ...
152           eta_range[max_val])
153     return(df)
154
155
156
157 # In[12]:
158
159
160
161 tab_01 = hyperparameter_testing(ds0_1,cv0_1)
162 tab_01.to_csv("acc_02.csv")
163
164
165 # In[8]:
166
167
168
169 nn0_1 = Perceptron(ds0_1, learning_rate = 0.01)
170 nn0_1.train()
171 print(nn0_1.accuracy(test0_1))
172
173
174 # In[23]:
175
176
177
178 nn0_1.confusionMatrix(ds0_1, name = "training classes 0 and 1", save_fig = True)
179
180
181 # In[24]:
182
183
184
185 nn0_1.confusionMatrix(test0_1, name = "test classes 0 and 1", save_fig = True)
186
187
188 # In[25]:
189
190
191
192 nn0_1.plot_decision_region(name = "training classes 0 and 1", savefig = True)
193
194
195 # In[155]:
196
197
198
199 tab_02 = hyperparameter_testing(ds0_2,cv0_2)
200 tab_02.to_csv("acc_02.csv")
201 tab_02
202
203 # In[20]:

```

```

196
197
198 nn0_2 = Perceptron(ds0_2)
199 nn0_2.train()
200 print(nn0_2.accuracy(test0_2))
201
202
203 # In[26]:
204
205
206 nn0_2.confusionMatrix(ds0_2, name = "training classes 0 and 2",save_fig=True)
207
208
209 # In[27]:
210
211
212 nn0_2.confusionMatrix(test0_2, name = "test classes 0 and 2",save_fig=True)
213
214
215 # In[28]:
216
217
218 nn0_2.plot_decision_region(name = "training classes 0 and 2",savefig = True)
219
220
221 # In[11]:
222
223
224 tab_03 = hyperparameter_testing(ds0_3,cv0_3)
225 tab_03.to_csv("acc_03.csv")
226 tab_03
227
228
229 # In[35]:
230
231
232 print(nn0_3.accuracy(test0_3))
233
234
235 # In[29]:
236
237
238 nn0_3 = Perceptron(ds0_3)
239 nn0_3.train()
240 nn0_3.plot_decision_region(name = "training classes 0 and 3",savefig = True)
241
242
243 # In[30]:
244
245
246 nn0_3.confusionMatrix(ds0_3, name = "training classes 0 and 3",save_fig=True)
247 nn0_3.confusionMatrix(test0_3, name = "test classes 0 and 3",save_fig=True)
248
249
250 # In[12]:
251
252
253 tab_13 = hyperparameter_testing(ds1_3,cv1_3)
254 tab_13.to_csv("acc_13_perc.csv")
255 tab_13
256
257
258 # In[36]:
259
260
261 print(nn1_3.accuracy(test1_3))
262
263
264 # In[31]:

```

```

265
266
267 nn1_3 = Perceptron(ds1_3)
268 nn1_3.train()
269 nn1_3.plot_decision_region(name = "training classes 1 and 3",savefig = True)
270 nn1_3.confusionMatrix(ds1_3, name = "training classes 1 and 3",save_fig=True)
271 nn1_3.confusionMatrix(test1_3, name = "test classes 1 and 3",save_fig=True)
272
273
274 # In[13]:
275
276
277 tab_23 = hyperparameter_testing(ds2_3,cv2_3)
278 tab_23.to_csv("acc_23_perc.csv")
279 tab_23
280
281
282 # In[37]:
283
284
285 print(nn2_3.accuracy(test2_3))
286
287
288 # In[32]:
289
290
291 nn2_3 = Perceptron(ds2_3)
292 nn2_3.train()
293 nn2_3.plot_decision_region(name = "training classes 2 and 3",savefig = True)
294 nn2_3.confusionMatrix(ds2_3, name = "training classes 2 and 3",save_fig=True)
295 nn2_3.confusionMatrix(test2_3, name = "test classes 2 and 3",save_fig=True)
296
297
298 # In[14]:
299
300
301 tab_12 = hyperparameter_testing(ds1_2,cv1_2)
302 tab_12.to_csv("acc_12_perc.csv")
303 tab_12
304
305
306 # In[38]:
307
308
309 print(nn1_2.accuracy(test1_2))
310
311
312 # In[33]:
313
314
315 nn1_2 = Perceptron(ds1_2,learning_rate = 0.05)
316 nn1_2.train()
317 nn1_2.plot_decision_region(name = "training classes 1 and 2",savefig = True)
318 nn1_2.confusionMatrix(ds1_2, name = "training classes 1 and 2",save_fig=True)
319 nn1_2.confusionMatrix(test1_2, name = "test classes 1 and 2",save_fig=True)
320
321
322 # In[ ]:

```

1.1.1 Helper Function

The helper function used is as follows:

1.1.1.1 Perceptron

```

1 import numpy as np
2 import pandas as pd

```

```

3 import random
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.metrics import confusion_matrix
7
8 import warnings
9 warnings.filterwarnings("ignore")
10
11 plt.rcParams["font.size"] = 18
12 plt.rcParams["axes.grid"] = True
13 plt.rcParams["figure.figsize"] = 12,8
14 plt.rcParams['font.serif'] = "Cambria"
15 plt.rcParams['font.family'] = "serif"
16
17
18 class Perceptron():
19     def __init__(self,data,epochs = 25,learning_rate = 0.01,verbose = False):
20         self.d = data.shape[1]-1
21         self.x = np.array(data.iloc[:, :self.d])
22         self.y = np.array(data.iloc[:, self.d])
23         self.epoch = epochs
24         self.len_data = data.shape[0]
25         self.eta = learning_rate
26         self.verbose = verbose
27     def one_epoch(self,w):
28         flag = 0
29         lst = list(range(self.len_data))
30         random.shuffle(lst)
31         for j in range(self.len_data):
32             m = lst.pop()
33             if w@self.x[m]<0:
34                 s = -1
35             else:
36                 s = 1
37             Δ = (self.y[m] - s)
38             if Δ==0:
39                 flag+=1
40             w += (self.eta)*(Δ/2)*self.x[m]
41         return(w,flag)
42     def train(self):
43         w = np.ones(self.d)
44         for i in range(self.epoch):
45             result = self.one_epoch(w)
46             w = result[0]
47             if (self.verbose==True):
48                 print("No. of correctly classified data points in "+str(i)+"th ...
49                     epoch : ", result[1])
49             if result[1] == self.len_data:
50                 break
51             if (self.verbose==True):
52                 print("Convergence reached in "+str(i)+" epochs")
53         self.W = w
54         #return(self.W)
55     def Y(self,m,c,xRange):
56         return(m*xRange+c)
57
58     def plot_decision_region(self, name, savefig = False):
59         x1Max = max(np.array(self.x[:,1]))+0.5
60         x1Min = min(np.array(self.x[:,1]))-0.5
61         x2Max = max(np.array(self.x[:,2]))+0.5
62         x2Min = min(np.array(self.x[:,2]))-0.5
63         color_list = ["palevioletred", "royalblue"]
64         xx, yy = np.meshgrid(np.arange(x1Min, x1Max, .02), np.arange(x2Min, x2Max, ...
65             .02))
66         z = self.predict(np.c_[np.ones(xx.ravel().shape),xx.ravel(),yy.ravel()])
67         z = z.reshape(xx.shape)
68         plt.contourf(xx, yy, z, colors= color_list, alpha=0.1)
69         plt.contour(xx, yy, z, colors=color_list, alpha=.1)
70         plt.scatter(self.x[:,1],self.x[:,2], c=[color_list[j] for j in self.y==1.])

```



```

70         plt.plot(np.linspace(x1Min,x1Max),self.Y*(-self.W[1]/self.W[2],-self.W[0]/...
71                 self.W[2],np.linspace(x1Min,x1Max)),label = "Decision Region")
72     plt.xlabel("X1")
73     plt.ylabel("X2")
74     plt.title("Decision region plot for the " + name + " data points")
75     #plt.legend()
76     if savefig == True:
77         plt.savefig(name+"dec_reg_perceptron.png")
78     plt.show()
79
80     def predict(self,mat):
81         return(np.sign(mat @ self.W))
82
83     def accuracy(self,test_data):
84         prediction = self.predict(test_data.iloc[:,self.d])
85         compare = prediction == test_data.iloc[:,self.d]
86         return(np.sum(compare)/len(test_data))
87
88     def confusionMatrix(self, dat, name, save_fig = False):
89         prediction = self.predict(dat.iloc[:,self.d])
90         conf_mat = confusion_matrix(dat.iloc[:,self.d],prediction)
91         plt.figure()
92         sns.heatmap(conf_mat, annot=True)
93         plt.title("1A - Confusion Matrix for " + name + " data (Perceptron)")
94         plt.xlabel("Predicted Class")
95         plt.ylabel("Actual Class")
96         if (save_fig == True):
97             plt.savefig("perceptron_" + name + "_confmat.png")
98         plt.show()

```

1.2 MLFFNN

The code written for analyzing Dataset 1A, using an MLFFNN model is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Assignment 3 - 1A (MLFFNN)
5  #
6  # Team members:
7  # - N Sowmya Manojna (BE17B007)
8  # - Thakkar Riya Anandbhai (PH17B010)
9  # - Chaithanya Krishna Moorthy (PH17B011)
10
11 # ## Importing Essential Libraries
12
13 # In[1]:
14
15
16 import numpy as np
17 import pandas as pd
18 import seaborn as sns
19 from sklearn.pipeline import Pipeline
20 from sklearn.metrics import confusion_matrix
21 from sklearn.neural_network import MLPClassifier
22 from sklearn.preprocessing import StandardScaler
23 from sklearn.model_selection import GridSearchCV
24 from sklearn.model_selection import train_test_split
25 from sklearn.model_selection import StratifiedShuffleSplit
26
27 import matplotlib.pyplot as plt
28 plt.rcParams["font.size"] = 18
29 plt.rcParams["axes.grid"] = True
30 plt.rcParams["figure.figsize"] = 12,8
31 plt.rcParams['font.serif'] = "Cambria"
32 plt.rcParams['font.family'] = "serif"
33
34 get_ipython().run_line_magic('load_ext', 'autoreload')

```

```

35 get_ipython().run_line_magic('autoreload', '2')
36
37 import warnings
38 warnings.filterwarnings("ignore")
39
40 from gridsearch import GridSearch1A
41
42
43 # ## Reading the data, Splitting it
44
45 # In[2]:
46
47
48 # Get the data
49 column_names = ["x1", "x2", "y"]
50 df = pd.read_csv("../datasets/1A/train.csv", names=column_names)
51 df_test = pd.read_csv("../datasets/1A/dev.csv", names=column_names)
52 display(df.head())
53
54 # Split dev into test and validation
55 df_val, df_test = train_test_split(df_test, test_size=0.3, random_state=42)
56 display(df_val.head())
57 display(df_test.head())
58
59
60 # In[3]:
61
62
63 X_train = df.drop("y", axis=1).to_numpy()
64 y_train = df["y"].to_numpy().astype("int")
65
66 X_val = df_val.drop("y", axis=1).to_numpy()
67 y_val = df_val["y"].to_numpy().astype("int")
68
69 X_test = df_test.drop("y", axis=1).to_numpy()
70 y_test = df_test["y"].to_numpy().astype("int")
71
72
73 # ## Training the Model
74
75 # In[4]:
76
77
78 parameters = {"hidden_layer_sizes": [5, 8, 10, 15], "activation": ["logistic", "tanh", "...",
    "relu"],
    "solver": ["lbfgs", "sgd", "adam"], "batch_size": [100, ...,
    200],
    "alpha": [0, 0.0001], "learning_rate": ["constant", "adaptive"...
    , "invscaling"],
    }
79
80 mlp = MLPClassifier(random_state=1)
81
82 clf = GridSearch1A(mlp, parameters)
83 clf.fit(X_train, y_train, X_val, y_val)
84 result_df = pd.DataFrame(clf.cv_results_)
85 result_df.to_csv("../parameter_search/1A_MLFFNN_train_val.csv")
86 result_df.head()
87
88
89 # In[5]:
90
91
92 print("Best Parameters Chosen:")
93 for i in clf.best_params_:
94     print("    - ", i, ": ", clf.best_params_[i], sep="")
95
96 best_mlp = MLPClassifier(random_state=1, **clf.best_params_)
97 best_mlp.fit(X_train, y_train)
98
99
100 # ## Best Model Predictions

```

```

101
102 # In[6]:
103
104
105 y_pred = best_mlp.predict(X_train)
106 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
107 conf_mat = confusion_matrix(y_train, y_pred)
108 plt.figure()
109 sns.heatmap(conf_mat, annot=True)
110 plt.title("1A - Train Confusion Matrix (MLFFNN)")
111 plt.xlabel("Predicted Class")
112 plt.ylabel("Actual Class")
113 plt.savefig("images/1A_MLFFNN_train_confmat.png")
114 plt.show()
115
116 y_val_pred = best_mlp.predict(X_val)
117 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
118 val_conf_mat = confusion_matrix(y_val, y_val_pred)
119 plt.figure()
120 sns.heatmap(val_conf_mat, annot=True)
121 plt.title("1A - Validation Confusion Matrix (MLFFNN)")
122 plt.xlabel("Predicted Class")
123 plt.ylabel("Actual Class")
124 plt.savefig("images/1A_MLFFNN_val_confmat.png")
125 plt.show()
126
127 y_test_pred = best_mlp.predict(X_test)
128 print("Validation Accuracy:", 100*np.sum(y_test_pred==y_test)/y_test.size)
129 test_conf_mat = confusion_matrix(y_test, y_test_pred)
130 plt.figure()
131 sns.heatmap(test_conf_mat, annot=True)
132 plt.title("1A - Test Confusion Matrix (MLFFNN)")
133 plt.xlabel("Predicted Class")
134 plt.ylabel("Actual Class")
135 plt.savefig("images/1A_MLFFNN_test_confmat.png")
136 plt.show()
137
138
139 # ## Visualising the decision boundaries
140
141 # In[7]:
142
143
144 h = 0.02
145 x_min, x_max = X_train[:,0].min() - .5, X_train[:,0].max() + .5
146 y_min, y_max = X_train[:,1].min() - .5, X_train[:,1].max() + .5
147
148 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
149 Z_pro = np.argmax(best_mlp.predict_proba(np.c_[xx.ravel(), yy.ravel()]), axis=1)
150 Z_pro = Z_pro.reshape(xx.shape)
151
152 color_list = ["springgreen", "gold", "palevioletred", "royalblue"]
153 plt.title("1A - Decision Region Plot (MLFFNN)")
154 plt.contourf(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=0.1)
155 plt.contour(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=1)
156 plt.scatter(X_train[:,0], X_train[:,1], c=[color_list[i] for i in y_train])
157 plt.xlabel("X1")
158 plt.ylabel("X2")
159 plt.savefig("images/1A_MLFFNN_Decision_Plot.png")
160 plt.show()
161
162
163 # In[ ]:

```

1.2.1 Helper Function

The helper functions used are as follows:

1.2.1.1 Gridsearch

```
1 import numpy as np
2 import pandas as pd
3 from time import time
4 from tqdm import tqdm
5 from collections import defaultdict
6 from sklearn.neural_network import MLPClassifier
7
8 class GridSearch1A():
9     def __init__(self, model, parameters, verbose=0):
10         self.model = model
11         self.parameters = parameters
12         self.verbose = verbose
13         params_list = []
14         self.params_keys = self.parameters.keys()
15
16         for hls in parameters["hidden_layer_sizes"]:
17             for act in parameters["activation"]:
18                 for s in parameters["solver"]:
19                     for bs in parameters["batch_size"]:
20                         for a in parameters["alpha"]:
21                             for lr in parameters["learning_rate"]:
22                                 params_list.append({"hidden_layer_sizes":hls, \
23                                                     "activation":act, \
24                                                     "solver":s, \
25                                                     "batch_size":bs, \
26                                                     "alpha":a, \
27                                                     "learning_rate":lr})
28
29         self.params_list = params_list
30
31     def fit(self, X_train, y_train, X_val, y_val):
32         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
33
34         self.params_ = defaultdict(list)
35         self.acc_list_ = []
36         self.val_acc_list_ = []
37         self.t_inv_list_ = []
38
39         for params in tqdm(self.params_list):
40             st = time()
41             mlp = MLPClassifier(random_state=1, **params)
42
43             mlp.fit(X_train, y_train)
44             et = time()
45
46             y_pred = mlp.predict(X_train)
47             acc = 100*np.sum(y_pred==y_train)/y_train.size
48
49             y_val_pred = mlp.predict(X_val)
50             val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
51
52             for i in params:
53                 self.params_[i].append(params[i])
54
55             self.acc_list_.append(acc)
56             self.val_acc_list_.append(val_acc)
57             self.t_inv_list_.append(1/(et-st))
58
59         for i in params:
60             self.cv_results_[i] = self.params_[i]
61
62         self.cv_results_["accuracy"] = self.acc_list_
63         self.cv_results_["val_accuracy"] = self.val_acc_list_
64         self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self....
65         self.cv_results_["val_accuracy"]
66         self.cv_results_["t_inv"] = self.t_inv_list_
```

```

65     self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
        sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
66
67     self.best_params_ = self.cv_results_.iloc[0].to_dict()
68     del self.best_params_["accuracy"]
69     del self.best_params_["val_accuracy"]
70     del self.best_params_["sum_accuracy"]
71     del self.best_params_["t_inv"]
72
73
74 class GridSearch1B():
75     def __init__(self, model, parameters, verbose=0):
76         self.model = model
77         self.parameters = parameters
78         self.verbose = verbose
79         params_list = []
80         self.params_keys = self.parameters.keys()
81
82         for hls in parameters["hidden_layer_sizes"]:
83             for act in parameters["activation"]:
84                 for bs in parameters["batch_size"]:
85                     for a in parameters["alpha"]:
86                         for lr in parameters["learning_rate"]:
87                             for es in parameters["early_stopping"]:
88                                 params_list.append({"hidden_layer_sizes":hls, \
89                                                     "early_stopping":es, \
90                                                     "learning_rate":lr, \
91                                                     "activation":act, \
92                                                     "batch_size":bs, \
93                                                     "alpha":a})
94
95         self.params_list = params_list
96
97     def fit(self, X_train, y_train, X_val, y_val):
98         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
99
100         self.params_ = defaultdict(list)
101         self.acc_list_ = []
102         self.val_acc_list_ = []
103         self.t_inv_list_ = []
104
105         for params in tqdm(self.params_list):
106             st = time()
107             mlp = MLPClassifier(random_state=1, **params)
108
109             mlp.fit(X_train, y_train)
110             et = time()
111
112             y_pred = mlp.predict(X_train)
113             acc = 100*np.sum(y_pred==y_train)/y_train.size
114
115             y_val_pred = mlp.predict(X_val)
116             val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
117
118             for i in params:
119                 self.params_[i].append(params[i])
120
121             self.acc_list_.append(acc)
122             self.val_acc_list_.append(val_acc)
123             self.t_inv_list_.append(1/(et-st))
124
125         for i in params:
126             self.cv_results_[i] = self.params_[i]
127
128         self.cv_results_["accuracy"] = self.acc_list_
129         self.cv_results_["val_accuracy"] = self.val_acc_list_
130         self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self...
            cv_results_["val_accuracy"]
131         self.cv_results_["t_inv"] = self.t_inv_list_

```

```

131         self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
            sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
132
133         self.best_params_ = self.cv_results_.iloc[0].to_dict()
134         self.best_params_["early_stopping"] = bool(self.best_params_["...
            early_stopping"])
135         del self.best_params_["accuracy"]
136         del self.best_params_["val_accuracy"]
137         del self.best_params_["sum_accuracy"]
138         del self.best_params_["t_inv"]

```

1.3 Linear SVM

The code written for analyzing Dataset 1A, using the Linear SVM model is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import numpy as np
8  import pandas as pd
9  from sklearn.metrics import confusion_matrix
10 from sklearn.model_selection import train_test_split
11 from sklearn import svm
12 import matplotlib.pyplot as plt
13 get_ipython().run_line_magic('matplotlib', 'inline')
14 plt.rcParams["font.size"]=18
15 plt.rcParams["axes.grid"]=True
16 plt.rcParams["figure.figsize"]=12,8
17 plt.rcParams["font.serif"]="Cambria"
18 plt.rcParams["font.family"]="serif"
19
20
21 # In[31]:
22
23
24 from statistics import mode
25
26
27 # In[2]:
28
29
30 from sklearn.metrics import classification_report
31
32
33 # In[3]:
34
35
36 from sklearn.model_selection import GridSearchCV
37
38
39 # In[4]:
40
41
42 import seaborn as sns
43
44
45 # In[5]:
46
47
48 color_list=["springgreen","gold","palevioletred","royalblue"]
49
50
51 # In[6]:
52

```

```

53
54 cols=["x1", "x2", "y"]
55 train_data=pd.read_csv("train.csv",names=cols)
56 dev_data=pd.read_csv("dev.csv",names=cols)
57
58
59 # In[7]:
60
61
62 data_cv,data_test=train_test_split(dev_data,test_size=0.3,random_state=42)
63
64
65 # In[8]:
66
67
68 X_train=train_data[["x1", "x2"]].to_numpy()
69 y_train=train_data["y"].to_numpy().astype("int")
70
71 X_cv=data_cv[["x1", "x2"]].to_numpy()
72 y_cv=data_cv["y"].to_numpy().astype("int")
73
74 X_test=data_test[["x1", "x2"]].to_numpy()
75 y_test=data_test["y"].to_numpy().astype("int")
76
77
78 # In[9]:
79
80
81 train_data.head()
82
83
84 # ## Training the Model
85
86 # ## we proceed with C=1:
87
88 # ## Linear SVM classifier for every pair of classes:
89
90 # In[15]:
91
92
93 def linear_ovo_plot(y1,y2,df,save_name,title,color,conf_title_train,conf_title_test...,
94 ,conf_train_save_name,conf_test_save_name,df_dev):
95     df2=df.loc[df["y"].isin([y1,y2])]
96     df2_dev=df_dev.loc[df_dev["y"].isin([y1,y2])]
97     df2_cv,df2_test=train_test_split(df2_dev,test_size=0.3,random_state=42)
98     predictor=svm.SVC(kernel="linear",C=1,decision_function_shape="ovo").fit(df2....
99         iloc[:,-1],df2.iloc[:,-1])
100     h=0.1
101     x1_min,x1_max=df2["x1"].min()-1,df2["x1"].max()+1
102     x2_min,x2_max=df2["x2"].min()-1,df2["x2"].max()+1
103     xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h),np.arange(x2_min,x2_max,h))
104     z=predictor.predict(np.c_[xx.ravel(),yy.ravel()])
105     z=z.reshape(xx.shape)
106
107     w=predictor.coef_[0]
108     a=-w[0]/w[1]
109
110     plt.figure()
111     x2=np.linspace(xx.min(),xx.max())
112     yx=a*x2-predictor.intercept_[0]/w[1]
113     plt.plot(x2,yx,label="Decision Boundary")
114
115     yx=a*x2-(predictor.intercept_[0]-1)/w[1]
116     plt.plot(x2,yx,"k--",label="Support Vector")
117
118     yx=a*x2-(predictor.intercept_[0]+1)/w[1]
119     plt.plot(x2,yx,"k--",label="Support Vector")
120     c1=color_list[y1]

```

```

120     c2=color_list[y2]
121     colors_list=[c1,c2]
122
123     plt.contourf(xx,yy,z,np.unique(z).size-1,colors=color,alpha=0.25)
124     plt.scatter(df2["x1"],df2["x2"],c=[color_list[i] for i in df2["y"].astype(int)...
125         ])
126     plt.xlabel("X1")
127     plt.ylabel("X2")
128     plt.xlim(xx.min(),xx.max())
129     plt.ylim(yy.min(),yy.max())
130     plt.legend(loc="upper right")
131     plt.savefig(save_name)
132     plt.title(title)
133     plt.show()
134
135     y_train=df2["y"]
136     ytrain_pred=predictor.predict(df2.iloc[:, :-1])
137
138     y_cv=df2_cv["y"]
139     y_test=df2_test["y"]
140     ytest_pred=predictor.predict(df2_test.iloc[:, :-1])
141
142     conf_mat=confusion_matrix(y_train,ytrain_pred)
143     plt.figure()
144     sns.heatmap(conf_mat,annot=True)
145     plt.title(conf_title_train)
146     plt.xlabel("Predicted Class")
147     plt.ylabel("Actual Class")
148     plt.savefig(conf_train_save_name)
149     plt.show()
150
151     conf_mat=confusion_matrix(y_test,ytest_pred)
152     plt.figure()
153     sns.heatmap(conf_mat,annot=True)
154     plt.title(conf_title_test)
155     plt.xlabel("Predicted Class")
156     plt.ylabel("Actual Class")
157     plt.savefig(conf_test_save_name)
158     plt.show()
159
160
161
162 # In[16]:
163
164
165 linear_ovo_plot(1,2,train_data,"images/1A_ovo_12.png","Support vectors and Boundary...
166     region between y=1.0 and y=2.0",color=[color_list[1],color_list[2]],...
167     conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...
168     matrix on test data",conf_train_save_name="images/1A_ovo_conf12_train.png",...
169     conf_test_save_name="images/1A_ovo_conf12_test.png",df_dev=dev_data)
170
171
172
173
174 # In[17]:
175
176
177 linear_ovo_plot(1,3,train_data,"images/1A_ovo_13.png","Support vectors and Boundary...
178     region between y=1.0 and y=3.0",color=[color_list[1],color_list[3]],...
179     conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...
180     matrix on test data",conf_train_save_name="images/1A_ovo_conf13_train.png",...
181     conf_test_save_name="images/1A_ovo_conf13_test.png",df_dev=dev_data)
182
183
184
185
186 # In[18]:
187
188
189 linear_ovo_plot(0,1,train_data,"images/1A_ovo_01.png","Support vectors and Boundary...
190     region between y=0.0 and y=1.0",color=[color_list[0],color_list[1]],...
191     conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...

```



```

        matrix on test data",conf_train_save_name="images/1A_ovo_conf01_train.png",...
        conf_test_save_name="images/1A_ovo_conf01_test.png",df_dev=dev_data)
178
179
180 # In[19]:
181
182
183 linear_ovo_plot(0,2,train_data,"images/1A_ovo_02.png","Support vectors and Boundary...
        region between y=0.0 and y=2.0",color=[color_list[0],color_list[2]],...
        conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...
        matrix on test data",conf_train_save_name="images/1A_ovo_conf02_train.png",...
        conf_test_save_name="images/1A_ovo_conf02_test.png",df_dev=dev_data)
184
185
186 # In[20]:
187
188
189 linear_ovo_plot(0,3,train_data,"images/1A_ovo_03.png","Support vectors and Boundary...
        region between y=0.0 and y=3.0",color=[color_list[0],color_list[3]],...
        conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...
        matrix on test data",conf_train_save_name="images/1A_ovo_conf03_train.png",...
        conf_test_save_name="images/1A_ovo_conf03_test.png",df_dev=dev_data)
190
191
192 # In[21]:
193
194
195 linear_ovo_plot(2,3,train_data,"images/1A_ovo_23.png","Support vectors and Boundary...
        region between y=2.0 and y=3.0",color=[color_list[2],color_list[3]],...
        conf_title_train="Confusion Matrix on train data",conf_title_test="Confusion ...
        matrix on test data",conf_train_save_name="images/1A_ovo_conf23_train.png",...
        conf_test_save_name="images/1A_ovo_conf23_test.png",df_dev=dev_data)
196
197
198 # # Using one-vs-one models to predict for a test sample:
199
200 # In[28]:
201
202
203 def class_model(df,y1,y2,C=1):
204     df2=df.loc[df["y"].isin([y1,y2])]
205     predictor=svm.SVC(kernel="linear",C=C,decision_function_shape="ovo").fit(df2....
        iloc[:,-1],df2.iloc[:,-1])
206     return(predictor)
207
208
209 # In[29]:
210
211
212 model01=class_model(train_data,0,1)
213 model02=class_model(train_data,0,2)
214 model03=class_model(train_data,0,3)
215 model12=class_model(train_data,1,2)
216 model13=class_model(train_data,1,3)
217 model23=class_model(train_data,2,3)
218
219
220 # In[54]:
221
222
223 from collections import Counter
224
225
226 # In[57]:
227
228
229 def ovo_predictor(x):
230     c=[]
231     c.append(model01.predict(x)[0])

```

```

232     c.append(model02.predict(x)[0])
233     c.append(model03.predict(x)[0])
234     c.append(model12.predict(x)[0])
235     c.append(model13.predict(x)[0])
236     c.append(model23.predict(x)[0])
237     count=Counter(c)
238     freq=0
239     label=0
240     for i in count.keys():
241         if count[i]>freq:
242             freq=count[i]
243             label=i
244     return label
245
246
247 # In[58]:
248
249
250 ytrain_pred=[]
251 ycv_pred=[]
252 ytest_pred=[]
253 for i in range(len(X_train)):
254     x=X_train[i,:].reshape(1,-1)
255     ytrain_pred.append(ovo_predictor(x))
256 for x in X_cv:
257     x=x.reshape(1,-1)
258     ycv_pred.append(ovo_predictor(x))
259 for x in X_test:
260     x=x.reshape(1,-1)
261     ytest_pred.append(ovo_predictor(x))
262
263
264 # In[59]:
265
266
267 def accuracy(actual,predicted):
268     return 100*np.sum(predicted==actual)/actual.size
269
270
271 # In[60]:
272
273
274 accuracy(y_train,ytrain_pred)
275
276
277 # In[67]:
278
279
280 conf_mat=confusion_matrix(y_train,ytrain_pred)
281 plt.figure()
282 sns.heatmap(conf_mat,annot=True)
283 plt.title("1a - Confusion matrix for train data" )
284 plt.xlabel("Predicted Class")
285 plt.ylabel("Actual Class")
286 plt.savefig("images/1a_confmatrix_train.png" )
287 plt.show()
288
289 conf_mat=confusion_matrix(y_test,ytest_pred)
290 plt.figure()
291 sns.heatmap(conf_mat,annot=True)
292 plt.title("1a - Confusion matrix for test data")
293 plt.xlabel("Predicted Class")
294 plt.ylabel("Actual Class")
295 plt.savefig("images/1a_confmatrix_test.png")
296 plt.show()
297
298
299 # In[64]:
300

```

```

301
302 h=0.1
303 x1_min,x1_max=train_data["x1"].min()-1,train_data["x1"].max()+1
304 x2_min,x2_max=train_data["x2"].min()-1,train_data["x2"].max()+1
305 xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h),np.arange(x2_min,x2_max,h))
306 X=np.c_[xx.ravel(),yy.ravel()]
307 z=[]
308 for i in X:
309     x=i.reshape(1,-1)
310     z.append(ovo_predictor(x))
311 z=np.array(z)
312 z=z.reshape(xx.shape)
313 plt.figure()
314 plt.contour(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=1)
315 plt.contourf(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=0.25)
316 plt.scatter(train_data["x1"],train_data["x2"],c=[color_list[i] for i in y_train])
317 plt.xlabel("X1")
318 plt.ylabel("X2")
319 plt.xlim(xx.min(),xx.max())
320 plt.ylim(yy.min(),yy.max())
321 plt.title("1A-Full Decision Region Plot(SVM)")
322 plt.savefig("images/1A_SVM_full_decision_plot.png")
323 plt.show()
324
325
326 # In[ ]:

```

2 Dataset 1B

2.1 MLFFNN

The code written for analyzing Dataset 1B, using an MLFFNN model is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Assignment 3 - 1B (MLFFNN)
5  #
6  # Team members:
7  # - N Sowmya Manojna (BE17B007)
8  # - Thakkar Riya Anandbhai (PH17B010)
9  # - Chaithanya Krishna Moorthy (PH17B011)
10
11 # ## Import Essential Libraries
12
13 # In[1]:
14
15
16 import numpy as np
17 import pandas as pd
18 import seaborn as sns
19 from sklearn.pipeline import Pipeline
20 from sklearn.metrics import confusion_matrix
21 from sklearn.neural_network import MLPClassifier
22 from sklearn.preprocessing import StandardScaler
23 from sklearn.model_selection import GridSearchCV
24 from sklearn.model_selection import train_test_split
25 from sklearn.model_selection import StratifiedShuffleSplit
26
27 import matplotlib.pyplot as plt
28 plt.rcParams["font.size"] = 18
29 plt.rcParams["axes.grid"] = True
30 plt.rcParams['font.serif'] = "Cambria"
31 plt.rcParams['font.family'] = "serif"
32
33 get_ipython().run_line_magic('load_ext', 'autoreload')

```

```

34 get_ipython().run_line_magic('autoreload', '2')
35
36 import warnings
37 warnings.filterwarnings("ignore")
38
39 from gridsearch import GridSearch1B
40
41
42 # ## Read the data, Split it
43
44 # In[2]:
45
46
47 # Get the data
48 column_names = ["x1", "x2", "y"]
49 df = pd.read_csv("../datasets/1B/train.csv", names=column_names)
50 df_test = pd.read_csv("../datasets/1B/dev.csv", names=column_names)
51 display(df.head())
52
53 # Split dev into test and validation
54 df_val, df_test = train_test_split(df_test, test_size=0.3, random_state=42)
55 display(df_val.head())
56 display(df_test.head())
57
58
59 # In[3]:
60
61
62 X_train = df[["x1", "x2"]].to_numpy()
63 y_train = df["y"].to_numpy().astype("int")
64
65 X_val = df_val[["x1", "x2"]].to_numpy()
66 y_val = df_val["y"].to_numpy().astype("int")
67
68 X_test = df_test[["x1", "x2"]].to_numpy()
69 y_test = df_test["y"].to_numpy().astype("int")
70
71
72 # ## Training the Model
73
74 # In[4]:
75
76
77 parameters = {"hidden_layer_sizes": [(5,5), (6,6), (7,7), (8,8), (9,9), (10,10)], ...
               "activation": ["logistic", "relu"], "batch_size": [50, ...
               100, 200], "early_stopping": [True, False], "learning_rate": [...
               constant", "adaptive", "invscaling"], "alpha": [0.01, 0.001]
               }
78
79
80 mlp = MLPClassifier(random_state=1)
81
82 clf = GridSearch1B(mlp, parameters)
83 clf.fit(X_train, y_train, X_val, y_val)
84 result_df = pd.DataFrame(clf.cv_results_)
85 result_df.to_csv("../parameter_search/1B_MLFFNN_train_val.csv")
86 result_df.head(10)
87
88
89 # In[5]:
90
91
92 print("Best Parameters Chosen:")
93 for i in clf.best_params_:
94     print("    - ", i, ": ", clf.best_params_[i], sep="")
95
96 best_mlp = MLPClassifier(random_state=1, **clf.best_params_)
97 best_mlp.fit(X_train, y_train)
98
99

```

```

100 # ## Best Model Predictions
101
102 # In[6]:
103
104
105 y_pred = best_mlp.predict(X_train)
106 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
107 conf_mat = confusion_matrix(y_train, y_pred)
108 plt.figure(figsize=(8,6))
109 sns.heatmap(conf_mat, annot=True)
110 plt.title("1B - Train Confusion Matrix (MLFFNN)")
111 plt.xlabel("Predicted Class")
112 plt.ylabel("Actual Class")
113 plt.savefig("images/1B_MLFFNN_train_confmat.png")
114 plt.show()
115
116 y_val_pred = best_mlp.predict(X_val)
117 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
118 val_conf_mat = confusion_matrix(y_val, y_val_pred)
119 plt.figure(figsize=(8,6))
120 sns.heatmap(val_conf_mat, annot=True)
121 plt.title("1B - Validation Confusion Matrix (MLFFNN)")
122 plt.xlabel("Predicted Class")
123 plt.ylabel("Actual Class")
124 plt.savefig("images/1B_MLFFNN_val_confmat.png")
125 plt.show()
126
127 y_test_pred = best_mlp.predict(X_test)
128 print("Test Accuracy:", 100*np.sum(y_test_pred==y_test)/y_test.size)
129 test_conf_mat = confusion_matrix(y_test, y_test_pred)
130 plt.figure(figsize=(8,6))
131 sns.heatmap(test_conf_mat, annot=True)
132 plt.title("1B - Test Confusion Matrix (MLFFNN)")
133 plt.xlabel("Predicted Class")
134 plt.ylabel("Actual Class")
135 plt.savefig("images/1B_MLFFNN_test_confmat.png")
136 plt.show()
137
138
139 # ## Visualising the decision boundaries
140
141 # In[7]:
142
143
144 h = 0.02
145 x_min, x_max = X_train[:,0].min() - .5, X_train[:,0].max() + .5
146 y_min, y_max = X_train[:,1].min() - .5, X_train[:,1].max() + .5
147
148 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
149 Z_pro = np.argmax(best_mlp.predict_proba(np.c_[xx.ravel(), yy.ravel()]), axis=1)
150 Z_pro = Z_pro.reshape(xx.shape)
151
152 color_list = ["springgreen", "gold", "palevioletred", "royalblue"]
153 plt.figure(figsize=(12,8))
154 plt.title("1B - Decision Region Plot (MLFFNN)")
155 plt.contourf(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=0.1)
156 plt.contour(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=1)
157 plt.scatter(X_train[:,0], X_train[:,1], c=[color_list[i] for i in y_train])
158 plt.xlabel("X1")
159 plt.ylabel("X2")
160 plt.savefig("images/1B_MLFFNN_Decision_Plot.png")
161 plt.show()
162
163
164 # ## Visualising Neuron Responses
165
166 # In[8]:
167
168

```

```

169 def get_values(weights, biases, X_train):
170     ip = X_train.T
171     h1 = weights[0].T @ ip + biases[0].reshape(-1,1)
172     a1 = np.maximum(0, h1)
173     h2 = weights[1].T @ a1 + biases[1].reshape(-1,1)
174     a2 = np.maximum(0, h2)
175     h3 = weights[2].T @ a2 + biases[2].reshape(-1,1)
176     pred = np.exp(h3)/np.sum(np.exp(h3))
177
178     return a1, a2, pred
179
180
181 # In[9]:
182
183
184 from matplotlib import cm
185 from mpl_toolkits import mplot3d
186 from mpl_toolkits.mplot3d import axes3d
187 grid = np.c_[xx.ravel(), yy.ravel()]
188
189 for epochs in [1, 5, 20, 100]:
190     mlp = MLPClassifier(random_state=1, max_iter=epochs, **clf.best_params_)
191     mlp.fit(X_train, y_train)
192
193     weights = mlp.coefs_
194     biases = mlp.intercepts_
195
196     a1, a2, op = get_values(weights, biases, grid)
197     a1 = a1.reshape(a1.shape[0], *xx.shape)
198     a2 = a2.reshape(a2.shape[0], *xx.shape)
199     op = op.reshape(op.shape[0], *xx.shape)
200
201
202     for i in range(a1.shape[0]):
203         fig = plt.figure(figsize=(8,8))
204         ax = plt.axes(projection="3d")
205
206         # ax.contour3D(xx, yy, a1[i,:], 500)
207         ax.contourf(xx, yy, a1[i,:], 500, cmap=cm.CMRmap)
208         ax.set_xlabel("X1")
209         ax.set_ylabel("X2")
210         ax.set_zlabel("HL1-Neuron "+str(i+1));
211         ax.set_title("Epoch: "+ str(epochs) + "; Surface for Layer 1, Neuron "+str(...
212             i+1))
213         plt.tight_layout()
214         plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_HL1_N"+str(i+1)+".png")
215         plt.show()
216
217     for i in range(a2.shape[0]):
218         fig = plt.figure(figsize=(8,8))
219         ax = plt.axes(projection="3d")
220
221         # ax.contour3D(xx, yy, a2[i,:], 500)
222         ax.contourf(xx, yy, a2[i,:], 500, cmap=cm.CMRmap)
223         ax.set_xlabel("X1")
224         ax.set_ylabel("X2")
225         ax.set_zlabel("HL2-Neuron "+str(i+1));
226         ax.set_title("Epoch: "+ str(epochs) + "; Surface for Layer 2, Neuron "+str(...
227             i+1))
228         plt.tight_layout()
229         plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_HL2_N"+str(i+1)+".png")
230         plt.show()
231
232     for i in range(op.shape[0]):
233         fig = plt.figure(figsize=(8,8))
234         ax = plt.axes(projection="3d")
235
236         # ax.contour3D(xx, yy, op[i,:], 500)
237         ax.contourf(xx, yy, op[i,:], 500, cmap=cm.CMRmap)

```

```

236         ax.set_xlabel("X1")
237         ax.set_ylabel("X2")
238         ax.set_zlabel("OP-Neuron "+str(i+1));
239         ax.set_title("Epoch: "+ str(epochs) + "; Surface for Output Layer, Neuron "...
                +str(i+1))
240         plt.tight_layout()
241         plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_OP_N"+str(i+1)+".png")
242         plt.show()
243
244
245 mlp = MLPClassifier(random_state=1, max_iter=1000, **clf.best_params_)
246 mlp.fit(X_train, y_train)
247
248 weights = mlp.coefs_
249 biases = mlp.intercepts_
250
251 a1, a2, op = get_values(weights, biases, grid)
252 a1 = a1.reshape(a1.shape[0], *xx.shape)
253 a2 = a2.reshape(a2.shape[0], *xx.shape)
254 op = op.reshape(op.shape[0], *xx.shape)
255
256
257 for i in range(a1.shape[0]):
258     fig = plt.figure(figsize=(8,8))
259     ax = plt.axes(projection="3d")
260
261     # ax.contour3D(xx, yy, a1[i,:], 500)
262     ax.contourf(xx, yy, a1[i,:], 500, cmap=cm.CMRmap)
263     ax.set_xlabel("X1")
264     ax.set_ylabel("X2")
265     ax.set_zlabel("HL1-Neuron "+str(i+1));
266     ax.set_title("Converged; Surface for Layer 1, Neuron "+str(i+1))
267     plt.tight_layout()
268     plt.savefig("images/1B_MLFFNN_conv_HL1_N"+str(i+1)+".png")
269     plt.show()
270
271 for i in range(a2.shape[0]):
272     fig = plt.figure(figsize=(8,8))
273     ax = plt.axes(projection="3d")
274
275     # ax.contour3D(xx, yy, a2[i,:], 500)
276     ax.contourf(xx, yy, a2[i,:], 500, cmap=cm.CMRmap)
277     ax.set_xlabel("X1")
278     ax.set_ylabel("X2")
279     ax.set_zlabel("HL2-Neuron "+str(i+1));
280     ax.set_title("Converged; Surface for Layer 2, Neuron "+str(i+1))
281     plt.tight_layout()
282     plt.savefig("images/1B_MLFFNN_conv_HL2_N"+str(i+1)+".png")
283     plt.show()
284
285 for i in range(op.shape[0]):
286     fig = plt.figure(figsize=(8,8))
287     ax = plt.axes(projection="3d")
288
289     # ax.contour3D(xx, yy, op[i,:], 500)
290     ax.contourf(xx, yy, op[i,:], 500, cmap=cm.CMRmap)
291     ax.set_xlabel("X1")
292     ax.set_ylabel("X2")
293     ax.set_zlabel("OP-Neuron "+str(i+1));
294     ax.set_title("Converged; Surface for Output Layer, Neuron "+str(i+1))
295     plt.tight_layout()
296     plt.savefig("images/1B_MLFFNN_conv_OP_N"+str(i+1)+".png")
297     plt.show()
298
299
300 # In[ ]:

```

2.1.1 Helper Function

The helper functions used are as follows:

2.1.1.1 Gridsearch

```
1 import numpy as np
2 import pandas as pd
3 from time import time
4 from tqdm import tqdm
5 from collections import defaultdict
6 from sklearn.neural_network import MLPClassifier
7
8 class GridSearch1A():
9     def __init__(self, model, parameters, verbose=0):
10         self.model = model
11         self.parameters = parameters
12         self.verbose = verbose
13         params_list = []
14         self.params_keys = self.parameters.keys()
15
16         for hls in parameters["hidden_layer_sizes"]:
17             for act in parameters["activation"]:
18                 for s in parameters["solver"]:
19                     for bs in parameters["batch_size"]:
20                         for a in parameters["alpha"]:
21                             for lr in parameters["learning_rate"]:
22                                 params_list.append({"hidden_layer_sizes":hls, \
23                                                     "activation":act, \
24                                                     "solver":s, \
25                                                     "batch_size":bs, \
26                                                     "alpha":a, \
27                                                     "learning_rate":lr})
28
29         self.params_list = params_list
30
31     def fit(self, X_train, y_train, X_val, y_val):
32         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
33
34         self.params_ = defaultdict(list)
35         self.acc_list_ = []
36         self.val_acc_list_ = []
37         self.t_inv_list_ = []
38
39         for params in tqdm(self.params_list):
40             st = time()
41             mlp = MLPClassifier(random_state=1, **params)
42
43             mlp.fit(X_train, y_train)
44             et = time()
45
46             y_pred = mlp.predict(X_train)
47             acc = 100*np.sum(y_pred==y_train)/y_train.size
48
49             y_val_pred = mlp.predict(X_val)
50             val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
51
52             for i in params:
53                 self.params_[i].append(params[i])
54
55             self.acc_list_.append(acc)
56             self.val_acc_list_.append(val_acc)
57             self.t_inv_list_.append(1/(et-st))
58
59         for i in params:
60             self.cv_results_[i] = self.params_[i]
61
62         self.cv_results_["accuracy"] = self.acc_list_
63         self.cv_results_["val_accuracy"] = self.val_acc_list_
```



```

63     self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self....
        cv_results_["val_accuracy"]
64     self.cv_results_["t_inv"] = self.t_inv_list_
65     self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
        sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
66
67     self.best_params_ = self.cv_results_.iloc[0].to_dict()
68     del self.best_params_["accuracy"]
69     del self.best_params_["val_accuracy"]
70     del self.best_params_["sum_accuracy"]
71     del self.best_params_["t_inv"]
72
73
74 class GridSearch1B():
75     def __init__(self, model, parameters, verbose=0):
76         self.model = model
77         self.parameters = parameters
78         self.verbose = verbose
79         params_list = []
80         self.params_keys = self.parameters.keys()
81
82         for hls in parameters["hidden_layer_sizes"]:
83             for act in parameters["activation"]:
84                 for bs in parameters["batch_size"]:
85                     for a in parameters["alpha"]:
86                         for lr in parameters["learning_rate"]:
87                             for es in parameters["early_stopping"]:
88                                 params_list.append({"hidden_layer_sizes":hls, \
89                                                     "early_stopping":es, \
90                                                     "learning_rate":lr, \
91                                                     "activation":act, \
92                                                     "batch_size":bs, \
93                                                     "alpha":a})
94
95         self.params_list = params_list
96
97     def fit(self, X_train, y_train, X_val, y_val):
98         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
99
100         self.params_ = defaultdict(list)
101         self.acc_list_ = []
102         self.val_acc_list_ = []
103         self.t_inv_list_ = []
104
105         for params in tqdm(self.params_list):
106             st = time()
107             mlp = MLPClassifier(random_state=1, **params)
108
109             mlp.fit(X_train, y_train)
110             et = time()
111
112             y_pred = mlp.predict(X_train)
113             acc = 100*np.sum(y_pred==y_train)/y_train.size
114
115             y_val_pred = mlp.predict(X_val)
116             val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
117
118             for i in params:
119                 self.params_[i].append(params[i])
120
121             self.acc_list_.append(acc)
122             self.val_acc_list_.append(val_acc)
123             self.t_inv_list_.append(1/(et-st))
124
125         for i in params:
126             self.cv_results_[i] = self.params_[i]
127
128         self.cv_results_["accuracy"] = self.acc_list_
129         self.cv_results_["val_accuracy"] = self.val_acc_list_

```

```

129         self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self....
            cv_results_["val_accuracy"]
130         self.cv_results_["t_inv"] = self.t_inv_list_
131         self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
            sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
132
133         self.best_params_ = self.cv_results_.iloc[0].to_dict()
134         self.best_params_["early_stopping"] = bool(self.best_params_["...
            early_stopping"])
135         del self.best_params_["accuracy"]
136         del self.best_params_["val_accuracy"]
137         del self.best_params_["sum_accuracy"]
138         del self.best_params_["t_inv"]

```

2.2 Non-Linear SVM

The code written for analyzing Dataset 1B, using the Non-Linear SVM models is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import numpy as np
8  import pandas as pd
9  from sklearn.metrics import confusion_matrix
10 from sklearn.model_selection import train_test_split
11 from sklearn import svm
12 import matplotlib.pyplot as plt
13 get_ipython().run_line_magic('matplotlib', 'inline')
14 plt.rcParams["font.size"]=18
15 plt.rcParams["axes.grid"]=True
16 plt.rcParams["figure.figsize"]=12,8
17 plt.rcParams["font.serif"]="Cambria"
18 plt.rcParams["font.family"]="serif"
19 from sklearn.metrics import classification_report
20 from sklearn.model_selection import GridSearchCV
21 import seaborn as sns
22 color_list=["springgreen","gold","palevioletred","cyan"]
23
24
25 # ## Importing and splitting the 1b datasets
26
27 # In[2]:
28
29
30 cols=["x1","x2","y"]
31 train_data=pd.read_csv("train1b.csv",names=cols)
32 dev_data=pd.read_csv("dev1b.csv",names=cols)
33 data_cv,data_test=train_test_split(dev_data,test_size=0.3,random_state=42)
34 X_train=train_data[["x1","x2"]].to_numpy()
35 y_train=train_data["y"].to_numpy().astype("int")
36
37 X_cv=data_cv[["x1","x2"]].to_numpy()
38 y_cv=data_cv["y"].to_numpy().astype("int")
39
40 X_test=data_test[["x1","x2"]].to_numpy()
41 y_test=data_test["y"].to_numpy().astype("int")
42
43
44 # In[3]:
45
46
47 plt.scatter(train_data["x1"],train_data["x2"],c=[color_list[i] for i in y_train])
48
49

```

```

50 # # Training the polynomial Kernel:
51
52 # In[4]:
53
54
55 C_list=[1,10,100,1000]
56 degree_list=[1,2,3,4,5,6]
57 gamma_list=[1,0.1,0.01,"auto"]
58 coef0_list=[10,100]
59
60
61 # In[5]:
62
63
64 param_grid={"C":C_list,"degree":degree_list,"gamma":gamma_list,"coef0":coef0_list,"...
        kernel":["poly"]}
65
66
67 # In[6]:
68
69
70 grid=GridSearchCV(svm.SVC(),param_grid,verbose=7,return_train_score=True,cv=2)
71
72
73 # In[7]:
74
75
76 grid.fit(X_train,y_train)
77
78
79 # In[8]:
80
81
82 results_df=pd.DataFrame(grid.cv_results_)
83
84
85 # In[9]:
86
87
88 results_df=results_df.sort_values(by="rank_test_score")
89
90
91 # In[40]:
92
93
94 results_df.head(10)
95
96
97 # In[48]:
98
99
100 results_df["params"].iloc[9]
101
102
103 # In[12]:
104
105
106 print("Best Parameters Chooosen:")
107 for i in grid.best_params_:
108     print("    - ", i, ": ", grid.best_params_[i], sep="")
109
110
111 # In[13]:
112
113
114 best_poly=svm.SVC(C=1000,coef0=100,degree=5,gamma=0.1,kernel="poly")
115
116
117 # In[14]:

```

```

118
119
120 best_poly.fit(X_train,y_train)
121 y_cv_polypred=best_poly.predict(X_cv)
122 y_test_polypred=best_poly.predict(X_test)
123 y_train_polypred=best_poly.predict(X_train)
124
125
126 # In[15]:
127
128
129 y_poly_trainaccuracy=100*np.sum(y_train_polypred==y_train)/y_train.size
130 y_poly_cvaccuracy=100*np.sum(y_cv_polypred==y_cv)/y_cv.size
131
132
133 # In[16]:
134
135
136 y_poly_trainaccuracy
137
138
139 # In[17]:
140
141
142 y_poly_cvaccuracy
143
144
145 # In[18]:
146
147
148 y_poly_testaccuracy=100*np.sum(y_test_polypred==y_test)/y_test.size
149
150
151 # In[19]:
152
153
154 y_poly_testaccuracy
155
156
157 # In[20]:
158
159
160 conf_mat=confusion_matrix(y_train,y_train_polypred)
161 plt.figure()
162 sns.heatmap(conf_mat,annot=True)
163 plt.title("1B - Train Confusion Matrix (SVM with Polynomial Kernel)")
164 plt.xlabel("Predicted Class")
165 plt.ylabel("Actual Class")
166 plt.savefig("images/1B_SVM_poly_train_confmat.png")
167 plt.show()
168
169 print(" Test Accuracy:",y_poly_testaccuracy)
170 test_conf_mat=confusion_matrix(y_test,y_test_polypred)
171 plt.figure()
172 sns.heatmap(test_conf_mat,annot=True)
173 plt.title("1B - Test Confusion Matrix (SVM with Polynomial Kernel)")
174 plt.xlabel("Predicted Class")
175 plt.ylabel("Actual Class")
176 plt.savefig("images/1B_SVM_poly_Test_confmat.png")
177 plt.show()
178
179
180 # # Decision Region Plot for Polynomial Kernel:
181
182 # In[21]:
183
184
185 sv=(best_poly.support_vectors_)
186

```

```

187
188 # In[ ]:
189
190
191
192
193
194 # In[49]:
195
196
197 h=0.1
198 x1_min,x1_max=train_data["x1"].min()-1,train_data["x1"].max()+1
199 x2_min,x2_max=train_data["x2"].min()-1,train_data["x2"].max()+1
200 xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h),np.arange(x2_min,x2_max,h))
201 z=best_poly.predict(np.c_[xx.ravel(),yy.ravel()])
202 z=z.reshape(xx.shape)
203 plt.figure()
204 plt.contour(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=1)
205 plt.contourf(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=0.1)
206 plt.scatter(train_data["x1"],train_data["x2"],c=[color_list[i] for i in y_train])
207 plt.scatter(sv[:,0],sv[:,1],marker="x",c="k",label="Support Vectors",alpha=0.5)
208 plt.xlabel("X1")
209 plt.ylabel("X2")
210 plt.legend()
211 plt.xlim(xx.min(),xx.max())
212 plt.ylim(yy.min(),yy.max())
213 plt.title("1B - Decision Region Plot (Polynomial SVM)")
214 plt.savefig("images/1B_SVM_poly_decision_plot.png")
215 plt.show()
216
217
218 # In[ ]:
219
220
221
222
223
224 # # Training the Gaussian Kernel:
225
226 # In[23]:
227
228
229 gamma_list=[1,0.01,0.001,0.0001]
230 C_list=[0.1,1,10,100,1000]
231
232 gauss_cv_accuracy={}
233 gauss_train_accuracy={}
234 for i in gamma_list:
235     gauss_train_accuracy[i]=[]
236     gauss_cv_accuracy[i]=[]
237     for j in C_list:
238         model=svm.SVC(kernel="rbf",decision_function_shape="ovr",C=j,gamma=i)
239         model.fit(X_train,y_train)
240         ytrain_pred=model.predict(X_train)
241         ycv_pred=model.predict(X_cv)
242         gauss_train_accuracy[i].append(100*np.sum(ytrain_pred==y_train)/y_train.size)
243         gauss_cv_accuracy[i].append(100*np.sum(ycv_pred==y_cv)/y_cv.size)
244
245
246 # In[24]:
247
248
249 gauss_cv_accuracy;
250
251
252 # In[25]:
253
254

```



```

321
322
323 test_gauss_accuracy
324
325
326 # # Confusion matrix for train and test data set, best gaussian model
327
328 # In[36]:
329
330
331 ytraingauss_pred=best_gauss_model.predict(X_train)
332 print(" Train Accuracy:",100*np.sum(ytraingauss_pred==y_train)/y_train.size)
333 conf_mat=confusion_matrix(y_train,ytraingauss_pred)
334 plt.figure()
335 sns.heatmap(conf_mat,annot=True)
336 plt.title("1B - Train Confusion Matrix (SVM with Gaussian Kernel)")
337 plt.xlabel("Predicted Class")
338 plt.ylabel("Actual Class")
339 plt.savefig("images/1B_SVM_gauss_train_confmat.png")
340 plt.show()
341
342 print(" Test Accuracy:",test_gauss_accuracy)
343 test_conf_mat=confusion_matrix(y_test,ygauss_testpred)
344 plt.figure()
345 sns.heatmap(test_conf_mat,annot=True)
346 plt.title("1B - Test Confusion Matrix (SVM with Gaussian Kernel)")
347 plt.xlabel("Predicted Class")
348 plt.ylabel("Actual Class")
349 plt.savefig("images/1B_SVM_gauss_Test_confmat.png")
350 plt.show()
351
352
353
354
355 # # Decision function plot:
356
357 # In[37]:
358
359
360 sv_gauss=best_gauss_model.support_vectors_
361
362
363 # In[38]:
364
365
366 h=0.1
367 x1_min,x1_max=train_data["x1"].min()-1,train_data["x1"].max()+1
368 x2_min,x2_max=train_data["x2"].min()-1,train_data["x2"].max()+1
369 xx,yy=np.meshgrid(np.arange(x1_min,x1_max,h),np.arange(x2_min,x2_max,h))
370 z=best_gauss_model.predict(np.c_[xx.ravel(),yy.ravel()])
371 z=z.reshape(xx.shape)
372 plt.figure()
373 plt.contour(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=1)
374 plt.contourf(xx,yy,z,np.unique(z).size-1,colors=color_list,alpha=0.1)
375 plt.scatter(train_data["x1"],train_data["x2"],c=[color_list[i] for i in y_train])
376 plt.scatter(sv_gauss[:,0],sv_gauss[:,1],marker="x",c="k",label="Support Vectors",...
377             alpha=1)
378 plt.legend()
379 plt.xlabel("X1")
380 plt.ylabel("X2")
381 plt.xlim(xx.min(),xx.max())
382 plt.ylim(yy.min(),yy.max())
383 plt.title("1B - Decision Region Plot (Gaussian SVM)")
384 plt.savefig("images/1B_SVM_gauss_decision_plot.png")
385 plt.show()
386
387 # In[ ]:

```

3 Dataset 2A

3.1 MLFFNN

The code written for analyzing Dataset 2A, using an MLFFNN model is as follows:

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Assignment 3 - 2 (MLFFNN)
5  #
6  # Team members:
7  # - N Sowmya Manojna (BE17B007)
8  # - Thakkar Riya Anandbhai (PH17B010)
9  # - Chaithanya Krishna Moorthy (PH17B011)
10
11 # ## Import Essential Libraries
12
13 # In[1]:
14
15
16 import numpy as np
17 import pandas as pd
18 import seaborn as sns
19 from ast import literal_eval
20 from sklearn.decomposition import PCA
21 from sklearn.pipeline import Pipeline
22 from sklearn.metrics import confusion_matrix
23 from sklearn.neural_network import MLPClassifier
24 from sklearn.preprocessing import StandardScaler
25 from sklearn.model_selection import GridSearchCV
26 from sklearn.model_selection import train_test_split
27 from sklearn.model_selection import StratifiedShuffleSplit
28
29 import matplotlib.pyplot as plt
30 plt.rcParams["font.size"] = 18
31 plt.rcParams["axes.grid"] = True
32 plt.rcParams["figure.figsize"] = 12,8
33 plt.rcParams['font.serif'] = "Cambria"
34 plt.rcParams['font.family'] = "serif"
35
36 get_ipython().run_line_magic('load_ext', 'autoreload')
37 get_ipython().run_line_magic('autoreload', '2')
38
39 import warnings
40 warnings.filterwarnings("ignore")
41
42 from gridsearch import GridSearch2A
43
44
45 # ## Reading the data, Splitting it
46
47 # In[2]:
48
49
50 # Get the data
51 df = pd.read_csv("../datasets/2A/train_new.csv")
52 df_test = pd.read_csv("../datasets/2A/dev_new.csv")
53 display(df.head())
54
55 # Split dev into test and validation
56 df_val, df_test = train_test_split(df_test, test_size=0.3, random_state=42)
57 display(df_val.head())
58 display(df_test.head())
59
60
61 # In[3]:
62
```



```

63
64 X_train = df.drop("class", axis=1)
65 y_train = df["class"].to_numpy().astype("int")
66
67 X_val = df_val.drop("class", axis=1)
68 y_val = df_val["class"].to_numpy().astype("int")
69
70 X_test = df_test.drop("class", axis=1)
71 y_test = df_test["class"].to_numpy().astype("int")
72
73
74 # In[4]:
75
76
77 display(df.describe())
78 display(df_val.describe())
79 display(df_test.describe())
80
81
82 # ## Preprocessing Dataset
83
84 # In[5]:
85
86
87 scaler = StandardScaler()
88 scaler.fit(X_train)
89 X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
90 X_val_scaled = pd.DataFrame(scaler.transform(X_val), columns=X_val.columns)
91 X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
92
93 display(X_train_scaled.describe())
94 display(X_val_scaled.describe())
95 display(X_test_scaled.describe())
96
97
98 # ## Training the Model
99
100 # In[6]:
101
102
103 parameters = {
104     "pca__n_components":list(range(1,25)),
105     "mlp__hidden_layer_sizes":[(10,10), (25,25), (50,50), (75,75)], \
106     "mlp__batch_size":[50, 100, "auto"], "mlp__alpha":[0.01, 0.001], \
107     "mlp__learning_rate":["constant", "adaptive", "invscaling"], \
108 }
109
110 model = Pipeline([('pca', PCA()), ('mlp', MLPClassifier(max_iter=500, random_state...
111     =1))])
112
113 clf = GridSearch2A(model, parameters, verbose=1)
114 clf.fit(X_train, y_train, X_val, y_val)
115 result_df = pd.DataFrame(clf.cv_results_)
116 result_df.to_csv("../parameter_search/2A_MLFFNN_train_val.csv")
117 display(result_df.head(10))
118
119 # In[7]:
120
121
122 clf.cv_results_ = clf.cv_results_.sort_values(by=["val_accuracy", "accuracy", "...
123     sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
124
125 clf.best_params_ = clf.cv_results_.iloc[0].to_dict()
126 del clf.best_params_["accuracy"]
127 del clf.best_params_["val_accuracy"]
128 del clf.best_params_["sum_accuracy"]
129 del clf.best_params_["t_inv"]
129

```

```

130
131 # In[8]:
132
133
134 print("Best Parameters Chosen:")
135 for i in clf.best_params_:
136     print(" - ", i, ":", clf.best_params_[i], sep="")
137
138 pca_params = {}
139 pca_params["n_components"] = clf.best_params_["n_components"]
140 mlp_params = clf.best_params_
141 mlp_params["hidden_layer_sizes"] = literal_eval(mlp_params["hidden_layer_sizes"])
142 try:
143     mlp_params["batch_size"] = int(mlp_params["batch_size"])
144 except:
145     pass
146
147 del mlp_params["n_components"]
148
149 best_model = Pipeline([('pca', PCA(**pca_params)),                                ('mlp', ...
150                        MLPClassifier(max_iter=500, random_state=1, **mlp_params))])
151 best_model.fit(X_train, y_train)
152
153 # In[9]:
154
155
156 y_pred = best_model.predict(X_train)
157 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
158 conf_mat = confusion_matrix(y_train, y_pred)
159 plt.figure()
160 sns.heatmap(conf_mat, annot=True)
161 plt.title("2A - Train Confusion Matrix (MLFFNN)")
162 plt.xlabel("Predicted Class")
163 plt.ylabel("Actual Class")
164 plt.savefig("images/2A_MLFFNN_train_confmat.png")
165 plt.show()
166
167 y_val_pred = best_model.predict(X_val)
168 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
169 val_conf_mat = confusion_matrix(y_val, y_val_pred)
170 plt.figure()
171 sns.heatmap(val_conf_mat, annot=True)
172 plt.title("2A - Validation Confusion Matrix (MLFFNN)")
173 plt.xlabel("Predicted Class")
174 plt.ylabel("Actual Class")
175 plt.savefig("images/2A_MLFFNN_val_confmat.png")
176 plt.show()
177
178 y_test_pred = best_model.predict(X_test)
179 print("Test Accuracy:", 100*np.sum(y_test_pred==y_test)/y_test.size)
180 test_conf_mat = confusion_matrix(y_test, y_test_pred)
181 plt.figure()
182 sns.heatmap(test_conf_mat, annot=True)
183 plt.title("2A - Test Confusion Matrix (MLFFNN)")
184 plt.xlabel("Predicted Class")
185 plt.ylabel("Actual Class")
186 plt.savefig("images/2A_MLFFNN_test_confmat.png")
187 plt.show()
188
189
190 # In[ ]:

```

3.1.1 Helper Function

The helper functions used are as follows:

3.1.1.1 Data Consolidation

```

1 import os
2 import numpy as np
3 import pandas as pd
4 from tqdm import tqdm
5
6 def get_consolidated_data2A(classes_present):
7     df = pd.DataFrame()
8     df_test = pd.DataFrame()
9     for i in classes_present:
10         df_new = pd.read_csv("../datasets/2A/"+i+"/train.csv")
11         df_new["image_names"] = classes_present[i]
12         df_new = df_new.rename(columns={"image_names": "class"})
13         df = df.append(df_new)
14
15         df_new_test = pd.read_csv("../datasets/2A/"+i+"/dev.csv")
16         df_new_test["image_names"] = classes_present[i]
17         df_new_test = df_new_test.rename(columns={"image_names": "class"})
18         df_test = df_test.append(df_new_test)
19
20     df.to_csv("../datasets/2A/train.csv", index=False)
21     df_test.to_csv("../datasets/2A/dev.csv", index=False)
22
23 if __name__ == "__main__":
24     classes_present = {"coast":0, "highway":1, "mountain":2, "opencountry":3, "...
25         tallbuilding":4}
26     get_consolidated_data2A(classes_present)

```

3.1.1.2 Gridsearch

```

1 import numpy as np
2 import pandas as pd
3 from time import time
4 from tqdm import tqdm
5 from collections import defaultdict
6 from sklearn.neural_network import MLPClassifier
7
8 class GridSearch1A():
9     def __init__(self, model, parameters, verbose=0):
10         self.model = model
11         self.parameters = parameters
12         self.verbose = verbose
13         self.params_list = []
14         self.params_keys = self.parameters.keys()
15
16         for hls in parameters["hidden_layer_sizes"]:
17             for act in parameters["activation"]:
18                 for s in parameters["solver"]:
19                     for bs in parameters["batch_size"]:
20                         for a in parameters["alpha"]:
21                             for lr in parameters["learning_rate"]:
22                                 params_list.append({"hidden_layer_sizes":hls, \
23                                     "activation":act, \
24                                     "solver":s, \
25                                     "batch_size":bs, \
26                                     "alpha":a, \
27                                     "learning_rate":lr})
28
29         self.params_list = params_list
30
31     def fit(self, X_train, y_train, X_val, y_val):
32         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
33
34         self.params_ = defaultdict(list)
35         self.acc_list_ = []
36         self.val_acc_list_ = []
37         self.t_inv_list_ = []

```

```

38     for params in tqdm(self.params_list):
39         st = time()
40         mlp = MLPClassifier(random_state=1, **params)
41
42         mlp.fit(X_train, y_train)
43         et = time()
44
45         y_pred = mlp.predict(X_train)
46         acc = 100*np.sum(y_pred==y_train)/y_train.size
47
48         y_val_pred = mlp.predict(X_val)
49         val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
50
51         for i in params:
52             self.params_[i].append(params[i])
53
54         self.acc_list_.append(acc)
55         self.val_acc_list_.append(val_acc)
56         self.t_inv_list_.append(1/(et-st))
57
58     for i in params:
59         self.cv_results_[i] = self.params_[i]
60
61     self.cv_results_["accuracy"] = self.acc_list_
62     self.cv_results_["val_accuracy"] = self.val_acc_list_
63     self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self....
64         cv_results_["val_accuracy"]
65     self.cv_results_["t_inv"] = self.t_inv_list_
66     self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
67         sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
68
69     self.best_params_ = self.cv_results_.iloc[0].to_dict()
70     del self.best_params_["accuracy"]
71     del self.best_params_["val_accuracy"]
72     del self.best_params_["sum_accuracy"]
73     del self.best_params_["t_inv"]
74
75 class GridSearch1B():
76     def __init__(self, model, parameters, verbose=0):
77         self.model = model
78         self.parameters = parameters
79         self.verbose = verbose
80         params_list = []
81         self.params_keys = self.parameters.keys()
82
83         for hls in parameters["hidden_layer_sizes"]:
84             for act in parameters["activation"]:
85                 for bs in parameters["batch_size"]:
86                     for a in parameters["alpha"]:
87                         for lr in parameters["learning_rate"]:
88                             for es in parameters["early_stopping"]:
89                                 params_list.append({"hidden_layer_sizes":hls, \
90                                                         "early_stopping":es, \
91                                                         "learning_rate":lr, \
92                                                         "activation":act, \
93                                                         "batch_size":bs, \
94                                                         "alpha":a})
95
96         self.params_list = params_list
97
98     def fit(self, X_train, y_train, X_val, y_val):
99         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
100
101         self.params_ = defaultdict(list)
102         self.acc_list_ = []
103         self.val_acc_list_ = []
104         self.t_inv_list_ = []
105
106         for params in tqdm(self.params_list):

```

```

105         st = time()
106         mlp = MLPClassifier(random_state=1, **params)
107
108         mlp.fit(X_train, y_train)
109         et = time()
110
111         y_pred = mlp.predict(X_train)
112         acc = 100*np.sum(y_pred==y_train)/y_train.size
113
114         y_val_pred = mlp.predict(X_val)
115         val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size
116
117         for i in params:
118             self.params_[i].append(params[i])
119
120         self.acc_list_.append(acc)
121         self.val_acc_list_.append(val_acc)
122         self.t_inv_list_.append(1/(et-st))
123
124     for i in params:
125         self.cv_results_[i] = self.params_[i]
126
127     self.cv_results_["accuracy"] = self.acc_list_
128     self.cv_results_["val_accuracy"] = self.val_acc_list_
129     self.cv_results_["sum_accuracy"] = self.cv_results_["accuracy"] + self....
130         cv_results_["val_accuracy"]
131     self.cv_results_["t_inv"] = self.t_inv_list_
132     self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
133         sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
134
135     self.best_params_ = self.cv_results_.iloc[0].to_dict()
136     self.best_params_["early_stopping"] = bool(self.best_params_["...
137         early_stopping"])
138     del self.best_params_["accuracy"]
139     del self.best_params_["val_accuracy"]
140     del self.best_params_["sum_accuracy"]
141     del self.best_params_["t_inv"]

```

3.2 Gaussian-kernel SVM

The code written for analyzing Dataset 1A, using the Gaussian-kernel SVM model is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import numpy as np
8  import pandas as pd
9  from sklearn.metrics import confusion_matrix
10 from sklearn.model_selection import train_test_split
11 from sklearn import svm
12 import matplotlib.pyplot as plt
13 get_ipython().run_line_magic('matplotlib', 'inline')
14 plt.rcParams["font.size"]=18
15 plt.rcParams["axes.grid"]=True
16 plt.rcParams["figure.figsize"]=12,8
17 plt.rcParams["font.serif"]="Cambria"
18 plt.rcParams["font.family"]="serif"
19 from sklearn.metrics import classification_report
20 from sklearn.model_selection import GridSearchCV
21 import seaborn as sns
22 color_list=["springgreen","gold","palevioletred","cyan"]
23
24
25 # In[2]:

```

```

26
27
28 train_data=pd.read_csv("train_new.csv")
29 dev_data=pd.read_csv("dev_new.csv")
30 data_cv,data_test=train_test_split(dev_data,test_size=0.3,random_state=42)
31 X_train=train_data.drop("class",axis=1).to_numpy()
32 y_train=train_data["class"].to_numpy().astype("int")
33
34 X_cv=data_cv.drop("class",axis=1).to_numpy()
35 y_cv=data_cv["class"].to_numpy().astype("int")
36
37 X_test=data_test.drop("class",axis=1).to_numpy()
38 y_test=data_test["class"].to_numpy().astype("int")
39
40
41 # In[3]:
42
43
44 train_data.describe()
45
46
47 # In[4]:
48
49
50 dev_data.describe()
51
52
53 # In[5]:
54
55
56 plt.figure(figsize=(30,30))
57 cor=train_data.corr()
58 sns.heatmap(cor,annot=True,cmap=plt.cm.Red)
59 plt.show()
60
61
62 # In[ ]:
63
64
65 gamma_list=[50,1,0.01,0.001,0.0001,10,100,"auto","scale"]
66 C_list=[0.01,0.1,1,10,100,1000]
67
68 cv_accuracy={}
69 train_accuracy={}
70 for i in gamma_list:
71     train_accuracy[i]=[]
72     cv_accuracy[i]=[]
73     for j in C_list:
74         model=svm.SVC(kernel="rbf",decision_function_shape="ovr",C=j,gamma=i,...
75             probability=True)
76         model.fit(X_train,y_train)
77         ytrain_pred=model.predict(X_train)
78         ycv_pred=model.predict(X_cv)
79         train_accuracy[i].append(100*np.sum(ytrain_pred==y_train)/y_train.size)
80         cv_accuracy[i].append(100*np.sum(ycv_pred==y_cv)/y_cv.size)
81
82 # In[ ]:
83
84
85 cv_accuracy
86
87
88 # In[22]:
89
90
91 C_list=[0.1,0.01,1,10,100,1000]
92 gamma_list=[0.1,0.01,1,5,10,100,1000,"auto","scale"]
93 param_grid={"C":C_list,"gamma":gamma_list,"kernel":["rbf"],"tol":[0.1,0.01,1],"...

```

```

        class_weight":["balanced",None], "break_ties":[True,False], "shrinking":[True,...
        False]}}
94 grid=GridSearchCV(svm.SVC(),param_grid,verbose=7,return_train_score=True,cv=2)
95
96
97 # In[23]:
98
99
100 grid.fit(X_train,y_train)
101
102
103 # In[67]:
104
105
106 results_df=pd.DataFrame(grid.cv_results_)
107
108
109 # In[73]:
110
111
112 results_df=results_df.sort_values(by="rank_test_score")
113 results_df=results_df.reset_index(drop=True)
114
115
116 # In[74]:
117
118
119 results_df.head(10)
120
121
122 # In[28]:
123
124
125 results_df.iloc[0,:]
126
127
128 # In[33]:
129
130
131 params=grid.best_params_
132
133
134 # In[34]:
135
136
137 params
138
139
140 # In[39]:
141
142
143 model=svm.SVC(C=10,break_ties=False,class_weight=None,gamma=1,kernel="rbf",...
        shrinking=True,tol=0.01)
144
145
146 # In[40]:
147
148
149 model.fit(X_train,y_train)
150
151
152 # In[42]:
153
154
155 ytrain_pred=model.predict(X_train)
156 ytest_pred=model.predict(X_test)
157 ycv_pred=model.predict(X_cv)
158
159

```

```

160 # In[43]:
161
162
163 y_trainaccuracy=100*np.sum(ytrain_pred==y_train)/y_train.size
164 y_cvaccuracy=100*np.sum(ycv_pred==y_cv)/y_cv.size
165 y_testaccuracy=100*np.sum(ytest_pred==y_test)/y_test.size
166
167
168 # In[44]:
169
170
171 y_trainaccuracy
172
173
174 # In[45]:
175
176
177 y_cvaccuracy
178
179
180 # In[46]:
181
182
183 y_testaccuracy
184
185
186 # In[48]:
187
188
189 conf_mat=confusion_matrix(y_train,ytrain_pred)
190 plt.figure()
191 sns.heatmap(conf_mat,annot=True)
192 plt.title("2A - Train Confusion Matrix (SVM with Gaussian Kernel)")
193 plt.xlabel("Predicted Class")
194 plt.ylabel("Actual Class")
195 plt.savefig("images/2A_SVM_gauss_train_confmat.png")
196 plt.show()
197
198 print(" Test Accuracy:",y_testaccuracy)
199 test_conf_mat=confusion_matrix(y_test,ytest_pred)
200 plt.figure()
201 sns.heatmap(test_conf_mat,annot=True)
202 plt.title("2A - Test Confusion Matrix (SVM with Gaussian Kernel)")
203 plt.xlabel("Predicted Class")
204 plt.ylabel("Actual Class")
205 plt.savefig("images/2A_SVM_gauss_Test_confmat.png")
206 plt.show()
207
208
209 # In[ ]:
210
211
212
213
214
215 # In[32]:
216
217
218 X_train
219
220
221 # In[49]:
222
223
224 results_df.iloc[0,:]
225
226
227 # In[122]:
228

```



```
229
230 results_df["params"][620]
231
232
233 # In[123]:
234
235
236 results_df["mean_train_score"][620]
237
238
239 # In[124]:
240
241
242 results_df["mean_test_score"][620]
243
244
245 # In[71]:
246
247
248 results_df.head()
249
250
251 # In[ ]:
```