

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
1.2	MLFFNN	6
1.2.1	Helper Function	9
1.2.1.1	Gridsearch	9
1.3	Linear SVM	11
2	Dataset 1B	11
2.1	MLFFNN	11
2.1.1	Helper Function	16
2.1.1.1	Gridsearch	16
2.2	Non-Linear SVM	18
3	Dataset 2A	18
3.1	MLFFNN	18
3.1.1	Helper Function	21
3.1.1.1	Data Consolidation	21
3.1.1.2	Gridsearch	21
3.2	Gaussian-kernel SVM	24

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[ ]:

```

1.2 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 = GridSearchCV(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_ = []

```

```

37
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_ = []

```

```

104     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     self.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"]

```

1.3 Linear SVM

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)], ...
78               "activation":["logistic", "relu"], "batch_size":[50, ...
79               100, 200], "early_stopping":[True, False], "learning_rate":["...
80               constant", "adaptive", "invscaling"], "alpha":[0.01, 0.001]
81               }
82
83 mlp = MLPClassifier(random_state=1)
84
85 clf = GridSearch1B(mlp, parameters)
86 clf.fit(X_train, y_train, X_val, y_val)
87 result_df = pd.DataFrame(clf.cv_results_)
88 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 Chooosen:")
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")

```

```

223     ax.set_ylabel("X2")
224     ax.set_zlabel("HL2-Neuron "+str(i+1));
225     ax.set_title("Epoch: " + str(epochs) + "; Surface for Layer 2, Neuron "+str(...
226         i+1))
227     plt.tight_layout()
228     plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_HL2_N"+str(i+1)+".png")
229     plt.show()
230
231 for i in range(op.shape[0]):
232     fig = plt.figure(figsize=(8,8))
233     ax = plt.axes(projection="3d")
234
235     # ax.contour3D(xx, yy, op[i,:], 500)
236     ax.contourf(xx, yy, op[i,:], 500, cmap=cm.CMRmap)
237     ax.set_xlabel("X1")
238     ax.set_ylabel("X2")
239     ax.set_zlabel("OP-Neuron "+str(i+1));
240     ax.set_title("Epoch: " + str(epochs) + "; Surface for Output Layer, Neuron "...
241         +str(i+1))
242     plt.tight_layout()
243     plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_OP_N"+str(i+1)+".png")
244     plt.show()
245
246 mlp = MLPClassifier(random_state=1, max_iter=1000, **clf.best_params_)
247 mlp.fit(X_train, y_train)
248
249 weights = mlp.coefs_
250 biases = mlp.intercepts_
251
252 a1, a2, op = get_values(weights, biases, grid)
253 a1 = a1.reshape(a1.shape[0], *xx.shape)
254 a2 = a2.reshape(a2.shape[0], *xx.shape)
255 op = op.reshape(op.shape[0], *xx.shape)
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         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
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

```

```

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):
107             st = time()
108             mlp = MLPClassifier(random_state=1, **params)
109
110             mlp.fit(X_train, y_train)
111             et = time()
112
113             y_pred = mlp.predict(X_train)
114             acc = 100*np.sum(y_pred==y_train)/y_train.size
115
116             y_val_pred = mlp.predict(X_val)
117             val_acc = 100*np.sum(y_val_pred==y_val)/y_val.size

```

```

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"]

```

2.2 Non-Linear SVM

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 wandb
17 import numpy as np
18 import pandas as pd
19 import seaborn as sns
20 from ast import literal_eval
21 from sklearn.decomposition import PCA
22 from sklearn.pipeline import Pipeline
23 from sklearn.metrics import confusion_matrix
24 from sklearn.neural_network import MLPClassifier
25 from sklearn.preprocessing import StandardScaler
26 from sklearn.model_selection import GridSearchCV
27 from sklearn.model_selection import train_test_split
28 from sklearn.model_selection import StratifiedShuffleSplit
29
30 import matplotlib.pyplot as plt
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 get_ipython().run_line_magic('load_ext', 'autoreload')
38 get_ipython().run_line_magic('autoreload', '2')
39
40 import warnings
41 warnings.filterwarnings("ignore")
42
43 from gridsearch import GridSearch2A
44
45
46 # ## Reading the data, Splitting it
47
48 # In[2]:
49
50
51 # Get the data
52 df = pd.read_csv("../datasets/2A/train_new.csv")
53 df_test = pd.read_csv("../datasets/2A/dev_new.csv")
54 display(df.head())
55
56 # Split dev into test and validation
57 df_val, df_test = train_test_split(df_test, test_size=0.3, random_state=42)
58 display(df_val.head())
59 display(df_test.head())
60
61
62 # In[3]:
63
64
65 X_train = df.drop("class", axis=1)
66 y_train = df["class"].to_numpy().astype("int")
67
68 X_val = df_val.drop("class", axis=1)
69 y_val = df_val["class"].to_numpy().astype("int")
70
71 X_test = df_test.drop("class", axis=1)
72 y_test = df_test["class"].to_numpy().astype("int")
73
74
75 # In[4]:
76
77
78 display(df.describe())
79 display(df_val.describe())
80 display(df_test.describe())
81
82
83 # ## Preprocessing Dataset
84
85 # In[5]:
86
87
88 scaler = StandardScaler()
89 scaler.fit(X_train)
90 X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
91 X_val_scaled = pd.DataFrame(scaler.transform(X_val), columns=X_val.columns)
92 X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
93
94 display(X_train_scaled.describe())
95 display(X_val_scaled.describe())
96 display(X_test_scaled.describe())
97
98
99 # ## Training the Model
100
101 # In[6]:

```

```

102
103
104 parameters = {
105     "pca__n_components":list(range(1,25)),
106     "mlp__hidden_layer_sizes":[(10,10), (25,25), (50,50), (75,75)], \
107     "mlp__batch_size":[50, 100, "auto"], "mlp__alpha":[0.01, 0.001], \
108     "mlp__learning_rate":["constant", "adaptive", "invscaling"], \
109 }
110
111 model = Pipeline([('pca', PCA()), ('mlp', MLPClassifier(max_iter=500, random_state...
112     =1))])
113
114 clf = GridSearch2A(model, parameters, verbose=1)
115 clf.fit(X_train, y_train, X_val, y_val)
116 result_df = pd.DataFrame(clf.cv_results_)
117 result_df.to_csv("../parameter_search/2A_MLFFNN_train_val.csv")
118 display(result_df.head(10))
119
120 # In[7]:
121
122
123 clf.cv_results_ = clf.cv_results_.sort_values(by=["val_accuracy", "accuracy", "...
124     sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
125
126 clf.best_params_ = clf.cv_results_.iloc[0].to_dict()
127 del clf.best_params_["accuracy"]
128 del clf.best_params_["val_accuracy"]
129 del clf.best_params_["sum_accuracy"]
130 del clf.best_params_["t_inv"]
131
132 # In[8]:
133
134
135 print("Best Parameters Chosen:")
136 for i in clf.best_params_:
137     print(" - ", i, ": ", clf.best_params_[i], sep="")
138
139 pca_params = {}
140 pca_params["n_components"] = clf.best_params_["n_components"]
141 mlp_params = clf.best_params_
142 mlp_params["hidden_layer_sizes"] = literal_eval(mlp_params["hidden_layer_sizes"])
143 try:
144     mlp_params["batch_size"] = int(mlp_params["batch_size"])
145 except:
146     pass
147
148 del mlp_params["n_components"]
149
150 best_model = Pipeline([('pca', PCA(**pca_params)), ('mlp', ...
151     MLPClassifier(max_iter=500, random_state=1, **mlp_params))])
152 best_model.fit(X_train, y_train)
153
154 # In[9]:
155
156
157 y_pred = best_model.predict(X_train)
158 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
159 conf_mat = confusion_matrix(y_train, y_pred)
160 plt.figure()
161 sns.heatmap(conf_mat, annot=True)
162 plt.title("2A - Train Confusion Matrix (MLFFNN)")
163 plt.xlabel("Predicted Class")
164 plt.ylabel("Actual Class")
165 plt.savefig("images/2A_MLFFNN_train_confmat.png")
166 plt.show()
167

```

```

168 y_val_pred = best_model.predict(X_val)
169 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
170 val_conf_mat = confusion_matrix(y_val, y_val_pred)
171 plt.figure()
172 sns.heatmap(val_conf_mat, annot=True)
173 plt.title("2A - Validation Confusion Matrix (MLFFNN)")
174 plt.xlabel("Predicted Class")
175 plt.ylabel("Actual Class")
176 plt.savefig("images/2A_MLFFNN_val_confmat.png")
177 plt.show()
178
179 y_test_pred = best_model.predict(X_test)
180 print("Test Accuracy:", 100*np.sum(y_test_pred==y_test)/y_test.size)
181 test_conf_mat = confusion_matrix(y_test, y_test_pred)
182 plt.figure()
183 sns.heatmap(test_conf_mat, annot=True)
184 plt.title("2A - Test Confusion Matrix (MLFFNN)")
185 plt.xlabel("Predicted Class")
186 plt.ylabel("Actual Class")
187 plt.savefig("images/2A_MLFFNN_test_confmat.png")
188 plt.show()
189
190
191 # 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         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             cv_results_["val_accuracy"]
66         self.cv_results_["t_inv"] = self.t_inv_list_
67         self.cv_results_ = self.cv_results_.sort_values(by=["accuracy", "...
68             sum_accuracy", "t_inv"], ascending=False, ignore_index=True)
69
70         self.best_params_ = self.cv_results_.iloc[0].to_dict()
71         del self.best_params_["accuracy"]
72         del self.best_params_["val_accuracy"]
73         del self.best_params_["sum_accuracy"]
74         del self.best_params_["t_inv"]

```

```

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

```


3.2 Gaussian-kernel SVM