

---

# Programming Assignment III

---

CS5691: PATTERN RECOGNITION AND MACHINE LEARNING

TEAM 17 CCS SECTION SPRING 2021

SEPTEMBER 23, 2022

AANAND KRISHNAN  
BE17B001

MANORANJAN J  
NA17B112

RENEETH KRISHNA MG  
BS17B025

# Contents

|            |  |           |
|------------|--|-----------|
| <b>I</b>   | <b>Pattern classification on linearly separable data</b>                               | <b>2</b>  |
| I.1        | <a href="#">Python Code</a> . . . . .  | 2         |
| <b>II</b>  | <b>Pattern classification on non-linearly separable data</b>                           | <b>19</b> |
| II.1       | K nearest Neighbours Method and Bayes classifier with KNN for density estimation . . . | 19        |
| II.1.1     | <a href="#">Python Code</a> . . . . .  | 19        |
| <b>III</b> | <b>Static Pattern Classification on Real World Dataset 2A</b>                          | <b>30</b> |
| III.1      | <a href="#">Python Code</a> . . . . .  | 30        |

# I Pattern classification on linearly separable data

## I.1 Python Code

---

```
1  #1A part I perceptron
2
3  import pandas as pd
4  import numpy as np
5  from matplotlib import pyplot as plt
6  import matplotlib.cm as cm
7  import matplotlib.colors as colors
8  from tqdm import tqdm
9
10 class perceptron():
11     def __init__(self, data1, data2):
12         self.data1 = data1
13         self.data2 = data2
14         self.data = np.concatenate((data1, data2), axis=0)
15         self.labels = [data1[0, -1], data2[0, -1]]
16         self.weights = None
17         self.error_epoch = None
18     def train(self, eta, epochs):
19         y1 = np.ones((len(self.data1), 1))
20         y2 = -1 * np.ones((len(self.data2), 1))
21         data = self.data
22         N, dim = data[:, :-1].shape
23         X = np.concatenate((np.ones((N, 1)), data[:, :-1]), axis=1)
24         y = np.concatenate((y1, y2), axis=0)
25         Xy = np.concatenate((X, np.reshape(y, (len(y), 1))), axis=1)
26         w = np.random.rand(1, 1+dim)
27         e = 1
28         error_epoch = []
29
30         pbar = tqdm(total=epochs, position=0, leave=True)
31         while e <= epochs:
32             e += 1
33             pbar.update(1)
34             np.random.shuffle(Xy)
35             for i in range(N):
36                 w = w + ((eta/2) * (Xy[i, -1] - np.sign(Xy[i, :-1] @ w.T)) * Xy[i, :-1])
37             error = 0
38             for i in range(N):
39                 error += (1/2) * abs((Xy[i, -1] - np.sign(Xy[i, :-1] @ w.T))) *
40                     (Xy[i, :-1] @ w.T) * Xy[i, -1]
41             error_epoch.append(-error)
42         pbar.close()
43         self.weights = w
44         self.error_epoch = np.array(error_epoch)
45         return None
46     def classify(self, data):
```

```

47         w = self.weights
48         labels = self.labels
49         l = [0]+labels
50         return np.array([l[int(i)] for i in np.sign(np.concatenate((np.ones(
51             (len(data),1)),data),axis=1)*w.T)])
52     def plot_error(self):
53         plt.plot(self.error_epoch)
54         return None
55
56 f = pd.read_csv('17/train.csv',header = None)
57 data_tr0 = (f[f[2]==0]).to_numpy()
58 data_tr1 = (f[f[2]==1]).to_numpy()
59 data_tr2 = (f[f[2]==2]).to_numpy()
60 data_tr3 = (f[f[2]==3]).to_numpy()
61 f = pd.read_csv('17/dev.csv',header = None)
62 data_te0 = (f[f[2]==0]).to_numpy()
63 data_te1 = (f[f[2]==1]).to_numpy()
64 data_te2 = (f[f[2]==2]).to_numpy()
65 data_te3 = (f[f[2]==3]).to_numpy()
66
67 eta = 0.1
68 epochs = 10
69 #01
70 p01 = perceptron(data_tr0,data_tr1)
71 p01.train(eta,epochs)
72 data_tr01 = np.concatenate((data_tr0,data_tr1),axis=0)
73 train_acc_01 = 100*((data_tr01[:, -1]==p01.classify(data_tr01[:, -1]))
74                     .mean())
75 print('train_acc_01 = %f'%(train_acc_01))
76 data_te01 = np.concatenate((data_te0,data_te1),axis=0)
77 test_acc_01 = 100*((data_te01[:, -1]==p01.classify(data_te01[:, -1]))
78                    .mean())
79 print('test_acc_01 = %f'%(test_acc_01))
80
81 #02
82 p02 = perceptron(data_tr0,data_tr2)
83 p02.train(eta,epochs)
84 data_tr02 = np.concatenate((data_tr0,data_tr2),axis=0)
85 train_acc_02 = 100*((data_tr02[:, -1]==p02.classify(data_tr02[:, -1]))
86                    .mean())
87 print('train_acc_02 = %f'%(train_acc_02))
88 data_te02 = np.concatenate((data_te0,data_te2),axis=0)
89 test_acc_02 = 100*((data_te02[:, -1]==p02.classify(data_te02[:, -1]))
90                    .mean())
91 print('test_acc_02 = %f'%(test_acc_02))
92
93 #03
94 p03 = perceptron(data_tr0,data_tr3)
95 p03.train(eta,epochs)
96 data_tr03 = np.concatenate((data_tr0,data_tr3),axis=0)

```

```

97 train_acc_03 = 100*((data_tr03[:, -1] == p03.classify(data_tr03[:, :-1]))
98     .mean())
99 print('train_acc_03 = %f'%(train_acc_03))
100 data_te03 = np.concatenate((data_te0, data_te3), axis=0)
101 test_acc_03 = 100*((data_te03[:, -1] == p03.classify(data_te03[:, :-1]))
102     .mean())
103 print('test_acc_03 = %f'%(test_acc_03))
104 #12
105 p12 = perceptron(data_tr1, data_tr2)
106 p12.train(eta, epochs)
107 data_tr12 = np.concatenate((data_tr1, data_tr2), axis=0)
108 train_acc_12 = 100*((data_tr12[:, -1] == p12.classify(data_tr12[:, :-1]))
109     .mean())
110 print('train_acc_12 = %f'%(train_acc_12))
111 data_te12 = np.concatenate((data_te1, data_te2), axis=0)
112 test_acc_12 = 100*((data_te12[:, -1] == p12.classify(data_te12[:, :-1]))
113     .mean())
114 print('test_acc_12 = %f'%(test_acc_12))
115 #13
116 p13 = perceptron(data_tr1, data_tr3)
117 p13.train(eta, epochs)
118 data_tr13 = np.concatenate((data_tr1, data_tr3), axis=0)
119 train_acc_13 = 100*((data_tr13[:, -1] == p13.classify(data_tr13[:, :-1]))
120     .mean())
121 print('train_acc_13 = %f'%(train_acc_13))
122 data_te13 = np.concatenate((data_te1, data_te3), axis=0)
123 test_acc_13 = 100*((data_te13[:, -1] == p13.classify(data_te13[:, :-1]))
124     .mean())
125 print('test_acc_13 = %f'%(test_acc_13))
126 #23
127 p23 = perceptron(data_tr2, data_tr3)
128 p23.train(eta, epochs)
129 data_tr23 = np.concatenate((data_tr2, data_tr3), axis=0)
130 train_acc_23 = 100*((data_tr23[:, -1] == p23.classify(data_tr23[:, :-1]))
131     .mean())
132 print('train_acc_23 = %f'%(train_acc_23))
133 data_te23 = np.concatenate((data_te2, data_te3), axis=0)
134 test_acc_23 = 100*((data_te23[:, -1] == p23.classify(data_te23[:, :-1]))
135     .mean())
136 print('test_acc_23 = %f'%(test_acc_23))
137
138 #plots
139 X_train = data_tr01
140 X_test = data_te01
141 # Decision Region Plots #####
142 # define bounds of the domain
143 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
144 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
145
146 # define the x and y scale

```

```

147 x1_grid = np.arange(min1, max1, 0.1)
148 x2_grid = np.arange(min2, max2, 0.1)
149
150 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
151
152 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
153 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
154
155 x = np.hstack((c1,c2))
156
157 y_pred = p01.classify(x)
158
159 x3_grid = y_pred.reshape(x1_grid.shape)
160
161
162
163 fig = plt.figure(1,figsize=(7.5,4))
164 ax = fig.add_subplot(111)
165 cmap =plt.get_cmap('Paired',2)
166 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
167 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
168           color = 'blue',label='train data')
169 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
170           color='black',label='test data')
171 ax.legend()
172 ax.set_xlabel('x1',fontsize=10)
173 ax.set_ylabel('x2',fontsize=10)
174 ax.set_title('Perceptron : Labels (0,1)', fontsize=10)
175 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
176                    ticks =[0.25,0.75])
177 cbar.ax.invert_yaxis()
178 cbar.set_ticklabels(['0','1'])
179
180 X_train = data_tr02
181 X_test = data_te02
182 # Decision Region Plots #####
183 # define bounds of the domain
184 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
185 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
186
187 # define the x and y scale
188 x1_grid = np.arange(min1, max1, 0.1)
189 x2_grid = np.arange(min2, max2, 0.1)
190
191 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
192
193 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
194 c1, c2 = x1_grid.reshape((len(c1), 1)),
195           x2_grid.reshape((len(c2), 1))
196

```

```

197 x = np.hstack((c1,c2))
198
199 y_pred = p02.classify(x)
200
201 x3_grid = y_pred.reshape(x1_grid.shape)
202
203
204
205 fig = plt.figure(2,figsize=(7.5,4))
206 ax = fig.add_subplot(111)
207 cmap =plt.get_cmap('Paired',2)
208 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
209 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
210           color = 'blue',label='train data')
211 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
212           color='black',label='test data')
213 ax.legend()
214 ax.set_xlabel('x1',fontsize=10)
215 ax.set_ylabel('x2',fontsize=10)
216 ax.set_title('Perceptron : Labels (0,2)', fontsize=10)
217 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),ticks =[0.25,0.75])
218 cbar.ax.invert_yaxis()
219 cbar.set_ticklabels(['0','2'])
220
221 X_train = data_tr03
222 X_test = data_te03
223 # Decision Region Plots #####
224 # define bounds of the domain
225 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
226 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
227
228 # define the x and y scale
229 x1_grid = np.arange(min1, max1, 0.1)
230 x2_grid = np.arange(min2, max2, 0.1)
231
232 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
233
234 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
235 c1, c2 = x1_grid.reshape((len(c1), 1)),
236           x2_grid.reshape((len(c2), 1))
237
238 x = np.hstack((c1,c2))
239
240 y_pred = p03.classify(x)
241
242 x3_grid = y_pred.reshape(x1_grid.shape)
243
244
245
246 fig = plt.figure(3,figsize=(7.5,4))

```

```

247 ax = fig.add_subplot(111)
248 cmap =plt.get_cmap('Paired',2)
249 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
250 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
251           color = 'blue',label='train data')
252 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
253           color='black',label='test data')
254 ax.legend()
255 ax.set_xlabel('x1',fontsize=10)
256 ax.set_ylabel('x2',fontsize=10)
257 ax.set_title('Perceptron : Labels (0,3)', fontsize=10)
258 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
259                   ticks =[0.25,0.75])
260 cbar.ax.invert_yaxis()
261 cbar.set_ticklabels(['0','3'])
262
263 X_train = data_tr12
264 X_test = data_te12
265 # Decision Region Plots #####
266 # define bounds of the domain
267 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
268 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
269
270 # define the x and y scale
271 x1_grid = np.arange(min1, max1, 0.1)
272 x2_grid = np.arange(min2, max2, 0.1)
273
274 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
275
276 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
277 c1, c2 = x1_grid.reshape((len(c1), 1)),
278           x2_grid.reshape((len(c2), 1))
279
280 x = np.hstack((c1,c2))
281
282 y_pred = p12.classify(x)
283
284 x3_grid = y_pred.reshape(x1_grid.shape)
285
286
287
288 fig = plt.figure(4,figsize=(7.5,4))
289 ax = fig.add_subplot(111)
290 cmap =plt.get_cmap('Paired',2)
291 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
292 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
293           color = 'blue',label='train data')
294 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
295           color='black',label='test data')
296 ax.legend()

```



```

297 ax.set_xlabel('x1',fontsize=10)
298 ax.set_ylabel('x2',fontsize=10)
299 ax.set_title('Perceptron : Labels (1,2)', fontsize=10)
300 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
301                      ticks = [0.25,0.75])
302 cbar.ax.invert_yaxis()
303 cbar.set_ticklabels(['1','2'])
304
305 X_train = data_tr13
306 X_test = data_te13
307 # Decision Region Plots #####
308 # define bounds of the domain
309 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
310 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
311
312 # define the x and y scale
313 x1_grid = np.arange(min1, max1, 0.1)
314 x2_grid = np.arange(min2, max2, 0.1)
315
316 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
317
318 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
319 c1, c2 = x1_grid.reshape((len(c1), 1)),
320           x2_grid.reshape((len(c2), 1))
321
322 x = np.hstack((c1,c2))
323
324 y_pred = p13.classify(x)
325
326 x3_grid = y_pred.reshape(x1_grid.shape)
327
328
329
330 fig = plt.figure(5,figsize=(7.5,4))
331 ax = fig.add_subplot(111)
332 cmap =plt.get_cmap('Paired',2)
333 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
334 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
335            color = 'blue',label='train data')
336 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
337            color='black',label='test data')
338 ax.legend()
339 ax.set_xlabel('x1',fontsize=10)
340 ax.set_ylabel('x2',fontsize=10)
341 ax.set_title('Perceptron : Labels (1,3)', fontsize=10)
342 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
343                      ticks = [0.25,0.75])
344 cbar.ax.invert_yaxis()
345 cbar.set_ticklabels(['1','3'])
346

```

```

347 X_train = data_tr23
348 X_test = data_te23
349 # Decision Region Plots #####
350 # define bounds of the domain
351 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
352 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
353
354 # define the x and y scale
355 x1_grid = np.arange(min1, max1, 0.1)
356 x2_grid = np.arange(min2, max2, 0.1)
357
358 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
359
360 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
361 c1, c2 = x1_grid.reshape((len(c1), 1)),
362          x2_grid.reshape((len(c2), 1))
363
364 x = np.hstack((c1,c2))
365
366 y_pred = p23.classify(x)
367
368 x3_grid = y_pred.reshape(x1_grid.shape)
369
370
371
372 fig = plt.figure(6,figsize=(7.5,4))
373 ax = fig.add_subplot(111)
374 cmap =plt.get_cmap('Paired',2)
375 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
376 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
377           color = 'blue',label='train data')
378 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
379           color='black',label='test data')
380 ax.legend()
381 ax.set_xlabel('x1',fontsize=10)
382 ax.set_ylabel('x2',fontsize=10)
383 ax.set_title('Perceptron : Labels (2,3)', fontsize=10)
384 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
385                   ticks =[0.25,0.75])
386 cbar.ax.invert_yaxis()
387 cbar.set_ticklabels(['2','3'])
388
389 #1A part II MLFNN
390
391 from sklearn import preprocessing
392 import pandas as pd
393 import numpy as np
394 import torch as tc
395 from matplotlib import pyplot as plt
396 import matplotlib.cm as cm

```

```

397 import matplotlib.colors as colors
398 from tqdm import tqdm
399
400 f = pd.read_csv('17/train.csv',header = None)
401 data_tr = f.to_numpy()
402 f = pd.read_csv('17/dev.csv',header = None)
403 data_te = f.to_numpy()
404 #1 to K categorical 1 hot vector transformation
405 labelenc = preprocessing.LabelBinarizer()
406 Xtr = (tc.from_numpy(data_tr[:, :-1])).float()
407 ytr = (tc.from_numpy(labelenc.fit_transform(data_tr[:, -1]))).float()
408
409 Xte = (tc.from_numpy(data_te[:, :-1])).float()
410 yte = (tc.from_numpy(labelenc.fit_transform(data_te[:, -1]))).float()
411
412
413 epochs = 10000
414
415 inp_l = 2
416 hid_l = 4
417 out_l = 4
418 #activation fns : Sigmoid, Tanh, Softmax, ReLU, ELU, SELU, CELU,
419 mlfnn = tc.nn.Sequential(tc.nn.Linear(inp_l, hid_l),
420                          tc.nn.ELU(), tc.nn.Linear(hid_l, out_l))
421 MSE = tc.nn.MSELoss()
422 optimizer = tc.optim.SGD(mlfnn.parameters(), lr=0.001)
423 from tqdm import tqdm
424 epochs = 10000
425 pbar = tqdm(total=epochs, position=0, leave=True)
426 for i in range(epochs):
427
428     optimizer.zero_grad()
429     ytrp = mlfnn(Xtr)
430     loss = MSE(ytrp, ytr)
431     loss.backward()
432     optimizer.step()
433     pbar.update(1)
434 pbar.close()
435 tr_acc = 100*((data_tr[:, -1] == labelenc.
436                inverse_transform(mlfnn(Xtr))).mean())
437 te_acc = 100*((data_te[:, -1] == labelenc.
438                inverse_transform(mlfnn(Xte))).mean())
439 print('train acc = %f'%(tr_acc))
440 print('test acc = %f'%(te_acc))
441
442 #plotting
443 X_train = data_tr[:, :-1]
444 X_test = data_te[:, :-1]
445 # Decision Region Plots #####
446 # define bounds of the domain

```

```

447 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
448 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
449
450 # define the x and y scale
451 x1_grid = np.arange(min1, max1, 0.1)
452 x2_grid = np.arange(min2, max2, 0.1)
453
454 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
455
456 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
457 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
458                      reshape((len(c2), 1))
459
460 x = np.hstack((c1,c2))
461
462 y_pred = labelenc.inverse_transform(mlfnn(tc.from_numpy(x)
463                                     .float()))
464
465 x3_grid = y_pred.reshape(x1_grid.shape)
466
467
468 fig = plt.figure(1,figsize=(7.5,4))
469 ax = fig.add_subplot(111)
470 cmap =plt.get_cmap('Paired',4)
471 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
472 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
473           color = 'blue',label='train data')
474 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
475           color='black',label='test data')
476 ax.legend()
477 ax.set_xlabel('x1',fontsize=10)
478 ax.set_ylabel('x2',fontsize=10)
479 ax.set_title('MLFNN : All Four Labels', fontsize=10)
480 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
481                    ticks=np.linspace(0.125,1-0.125,4))
482 cbar.ax.invert_yaxis()
483 cbar.set_ticklabels(['0','1','2','3'])
484
485 #1A part III linear svm
486 import numpy as np
487 import pandas as pd
488 from sklearn.svm import SVC
489 from matplotlib import pyplot as plt
490 import matplotlib.cm as cm
491 import matplotlib.colors as colors
492 from tqdm import tqdm
493
494 f = pd.read_csv('17/train.csv',header = None)
495 data_tr0 = (f[f[2]==0]).to_numpy()
496 data_tr1 = (f[f[2]==1]).to_numpy()

```

```

497 data_tr2 = (f[f[2]==2]).to_numpy()
498 data_tr3 = (f[f[2]==3]).to_numpy()
499 f = pd.read_csv('17/dev.csv',header = None)
500 data_te0 = (f[f[2]==0]).to_numpy()
501 data_te1 = (f[f[2]==1]).to_numpy()
502 data_te2 = (f[f[2]==2]).to_numpy()
503 data_te3 = (f[f[2]==3]).to_numpy()
504
505 #01
506 data_tr01 = np.concatenate((data_tr0,data_tr1),axis=0)
507 data_te01 = np.concatenate((data_te0,data_te1),axis=0)
508 svc01 = SVC(kernel='linear')
509 svc01.fit(data_tr01[:, :-1],data_tr01[:, -1])
510 train_acc_01 = 100*((data_tr01[:, -1]==svc01.
511                     predict(data_tr01[:, :-1])).mean())
512 print('train_acc_01 = %f'%(train_acc_01))
513 test_acc_01 = 100*((data_te01[:, -1]==svc01.
514                     predict(data_te01[:, :-1])).mean())
515 print('test_acc_01 = %f'%(test_acc_01))
516
517 #02
518 data_tr02 = np.concatenate((data_tr0,data_tr2),axis=0)
519 data_te02 = np.concatenate((data_te0,data_te2),axis=0)
520 svc02 = SVC(kernel='linear')
521 svc02.fit(data_tr02[:, :-1],data_tr02[:, -1])
522 train_acc_02 = 100*((data_tr02[:, -1]==svc02.
523                     predict(data_tr02[:, :-1])).mean())
524 print('train_acc_02 = %f'%(train_acc_02))
525 test_acc_02 = 100*((data_te02[:, -1]==svc02.
526                     predict(data_te02[:, :-1])).mean())
527 print('test_acc_02 = %f'%(test_acc_02))
528
529 #03
530 data_tr03 = np.concatenate((data_tr0,data_tr3),axis=0)
531 data_te03 = np.concatenate((data_te0,data_te3),axis=0)
532 svc03 = SVC(kernel='linear')
533 svc03.fit(data_tr03[:, :-1],data_tr03[:, -1])
534 train_acc_03 = 100*((data_tr03[:, -1]==svc03.
535                     predict(data_tr03[:, :-1])).mean())
536 print('train_acc_03 = %f'%(train_acc_03))
537 test_acc_03 = 100*((data_te03[:, -1]==svc03.
538                     predict(data_te03[:, :-1])).mean())
539 print('test_acc_03 = %f'%(test_acc_03))
540
541 #12
542 data_tr12 = np.concatenate((data_tr1,data_tr2),axis=0)
543 data_te12 = np.concatenate((data_te1,data_te2),axis=0)
544 svc12 = SVC(kernel='linear')
545 svc12.fit(data_tr12[:, :-1],data_tr12[:, -1])
546 train_acc_12 = 100*((data_tr12[:, -1]==svc12.

```

```

547         predict(data_tr12[:, :-1])).mean())
548 print('train_acc_12 = %f'%(train_acc_12))
549 test_acc_12 = 100*((data_te12[:, -1]==svc12.
550         predict(data_te12[:, :-1])).mean())
551 print('test_acc_12 = %f'%(test_acc_12))
552
553 #13
554 data_tr13 = np.concatenate((data_tr1, data_tr3), axis=0)
555 data_te13 = np.concatenate((data_te1, data_te3), axis=0)
556 svc13 = SVC(kernel='linear')
557 svc13.fit(data_tr13[:, :-1], data_tr13[:, -1])
558 train_acc_13 = 100*((data_tr13[:, -1]==svc13.
559         predict(data_tr13[:, :-1])).mean())
560 print('train_acc_13 = %f'%(train_acc_13))
561 test_acc_13 = 100*((data_te13[:, -1]==svc13.
562         predict(data_te13[:, :-1])).mean())
563 print('test_acc_13 = %f'%(test_acc_13))
564
565 #23
566 data_tr23 = np.concatenate((data_tr2, data_tr3), axis=0)
567 data_te23 = np.concatenate((data_te2, data_te3), axis=0)
568 svc23 = SVC(kernel='linear')
569 svc23.fit(data_tr23[:, :-1], data_tr23[:, -1])
570 train_acc_23 = 100*((data_tr23[:, -1]==svc23.
571         predict(data_tr23[:, :-1])).mean())
572 print('train_acc_23 = %f'%(train_acc_23))
573 test_acc_23 = 100*((data_te23[:, -1]==svc23.
574         predict(data_te23[:, :-1])).mean())
575 print('test_acc_23 = %f'%(test_acc_23))
576
577 #plots
578 X_train = data_tr01
579 X_test = data_te01
580 # Decision Region Plots #####
581 # define bounds of the domain
582 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
583 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
584
585 # define the x and y scale
586 x1_grid = np.arange(min1, max1, 0.1)
587 x2_grid = np.arange(min2, max2, 0.1)
588
589 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
590
591 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
592 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
593         reshape((len(c2), 1))
594
595 x = np.hstack((c1, c2))
596

```

```

597 y_pred = svc01.predict(x)
598
599 x3_grid = y_pred.reshape(x1_grid.shape)
600
601 suppvvec = svc01.support_vectors_
602
603 fig = plt.figure(1,figsize=(7.5,4))
604 ax = fig.add_subplot(111)
605 cmap =plt.get_cmap('Paired',2)
606 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
607 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
608           color = 'blue',label='train data')
609 ax.scatter(suppvvec[:,0],suppvvec[:,1],marker='x',
610           color='yellow',label='support vector')
611 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
612           color='black',label='test data')
613 ax.legend()
614 ax.set_xlabel('x1',fontsize=10)
615 ax.set_ylabel('x2',fontsize=10)
616 ax.set_title('SVM : Labels (0,1)', fontsize=10)
617 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
618                    ticks =[0.25,0.75])
619 cbar.ax.invert_yaxis()
620 cbar.set_ticklabels(['0','1'])
621
622
623
624 X_train = data_tr02
625 X_test = data_te02
626 # Decision Region Plots #####
627 # define bounds of the domain
628 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
629 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
630
631 # define the x and y scale
632 x1_grid = np.arange(min1, max1, 0.1)
633 x2_grid = np.arange(min2, max2, 0.1)
634
635 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
636
637 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
638 c1, c2 = x1_grid.reshape((len(c1), 1)),
639           x2_grid.reshape((len(c2), 1))
640
641 x = np.hstack((c1,c2))
642
643 y_pred = svc02.predict(x)
644
645 x3_grid = y_pred.reshape(x1_grid.shape)
646

```

```

647 suppvec = svc02.support_vectors_
648
649 fig = plt.figure(2,figsize=(7.5,4))
650 ax = fig.add_subplot(111)
651 cmap =plt.get_cmap('Paired',2)
652 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
653 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
654           color = 'blue',label='train data')
655 ax.scatter(suppvec[:,0],suppvec[:,1],marker='x',
656           color='yellow',label='support vector')
657 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
658           color='black',label='test data')
659 ax.legend()
660 ax.set_xlabel('x1',fontsize=10)
661 ax.set_ylabel('x2',fontsize=10)
662 ax.set_title('SVM : Labels (0,2)', fontsize=10)
663 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
664                    ticks =[0.25,0.75])
665 cbar.ax.invert_yaxis()
666 cbar.set_ticklabels(['0', '2'])
667
668 X_train = data_tr03
669 X_test = data_te03
670 # Decision Region Plots #####
671 # define bounds of the domain
672 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
673 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
674
675 # define the x and y scale
676 x1_grid = np.arange(min1, max1, 0.1)
677 x2_grid = np.arange(min2, max2, 0.1)
678
679 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
680
681 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
682 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
683           reshape((len(c2), 1))
684
685 x = np.hstack((c1,c2))
686
687 y_pred = svc03.predict(x)
688
689 x3_grid = y_pred.reshape(x1_grid.shape)
690
691 suppvec = svc03.support_vectors_
692
693 fig = plt.figure(3,figsize=(7.5,4))
694 ax = fig.add_subplot(111)
695 cmap =plt.get_cmap('Paired',2)
696 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)

```



```

697 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
698             color = 'blue',label='train data')
699 ax.scatter(suppvec[:,0],suppvec[:,1],marker='x',
700             color='yellow',label='support vector')
701 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
702             color='black',label='test data')
703 ax.legend()
704 ax.set_xlabel('x1',fontsize=10)
705 ax.set_ylabel('x2',fontsize=10)
706 ax.set_title('SVM : Labels (0,3)', fontsize=10)
707 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
708                     ticks =[0.25,0.75])
709 cbar.ax.invert_yaxis()
710 cbar.set_ticklabels(['0','3'])
711
712 X_train = data_tr12
713 X_test = data_te12
714 # Decision Region Plots #####
715 # define bounds of the domain
716 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
717 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
718
719 # define the x and y scale
720 x1_grid = np.arange(min1, max1, 0.1)
721 x2_grid = np.arange(min2, max2, 0.1)
722
723 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
724
725 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
726 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
727                      reshape((len(c2), 1))
728
729 x = np.hstack((c1,c2))
730
731 y_pred = svc12.predict(x)
732
733 x3_grid = y_pred.reshape(x1_grid.shape)
734
735 suppvec = svc12.support_vectors_
736
737 fig = plt.figure(4,figsize=(7.5,4))
738 ax = fig.add_subplot(111)
739 cmap =plt.get_cmap('Paired',2)
740 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
741 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
742             color = 'blue',label='train data')
743 ax.scatter(suppvec[:,0],suppvec[:,1],marker='x',
744             color='yellow',label='support vector')
745 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
746             color='black',label='test data')

```

```

747 ax.legend()
748 ax.set_xlabel('x1',fontsize=10)
749 ax.set_ylabel('x2',fontsize=10)
750 ax.set_title('SVM : Labels (1,2)', fontsize=10)
751 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
752                      ticks =[0.25,0.75])
753 cbar.ax.invert_yaxis()
754 cbar.set_ticklabels(['1','2'])
755
756 X_train = data_tr13
757 X_test = data_te13
758 # Decision Region Plots #####
759 # define bounds of the domain
760 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
761 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
762
763 # define the x and y scale
764 x1_grid = np.arange(min1, max1, 0.1)
765 x2_grid = np.arange(min2, max2, 0.1)
766
767 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
768
769 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
770 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
771                      reshape((len(c2), 1))
772
773 x = np.hstack((c1,c2))
774
775 y_pred = svc13.predict(x)
776
777 x3_grid = y_pred.reshape(x1_grid.shape)
778
779 suppvec = svc13.support_vectors_
780
781 fig = plt.figure(5,figsize=(7.5,4))
782 ax = fig.add_subplot(111)
783 cmap =plt.get_cmap('Paired',2)
784 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
785 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
786            color = 'blue',label='train data')
787 ax.scatter(suppvec[:,0],suppvec[:,1],marker='x',
788            color='yellow',label='support vector')
789 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
790            color='black',label='test data')
791 ax.legend()
792 ax.set_xlabel('x1',fontsize=10)
793 ax.set_ylabel('x2',fontsize=10)
794 ax.set_title('SVM : Labels (1,3)', fontsize=10)
795 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),ticks =[0.25,0.75])
796 cbar.ax.invert_yaxis()

```

```

797 cbar.set_ticklabels(['1','3'])
798
799 X_train = data_tr23
800 X_test = data_te23
801 # Decision Region Plots #####
802 # define bounds of the domain
803 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
804 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
805
806 # define the x and y scale
807 x1_grid = np.arange(min1, max1, 0.1)
808 x2_grid = np.arange(min2, max2, 0.1)
809
810 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
811
812 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
813 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.
814                      reshape((len(c2), 1))
815
816 x = np.hstack((c1,c2))
817
818 y_pred = svc23.predict(x)
819
820 x3_grid = y_pred.reshape(x1_grid.shape)
821
822 suppvec = svc23.support_vectors_
823
824 fig = plt.figure(6,figsize=(7.5,4))
825 ax = fig.add_subplot(111)
826 cmap =plt.get_cmap('Paired',2)
827 cs = ax.contourf(x1_grid, x2_grid, x3_grid, cmap=cmap)
828 ax.scatter(X_train[:,0],X_train[:,1],marker='x',
829           color = 'blue',label='train data')
830 ax.scatter(suppvec[:,0],suppvec[:,1],marker='x',
831           color='yellow',label='support vector')
832 ax.scatter(X_test[:,0],X_test[:,1],marker='x',
833           color='black',label='test data')
834 ax.legend()
835 ax.set_xlabel('x1',fontsize=10)
836 ax.set_ylabel('x2',fontsize=10)
837 ax.set_title('SVM : Labels (2,3)', fontsize=10)
838 cbar = plt.colorbar(cm.ScalarMappable(cmap=cmap),
839                   ticks =[0.25,0.75])
840 cbar.ax.invert_yaxis()
841 cbar.set_ticklabels(['2','3'])

```

---

## II Pattern classification on non-linearly separable data

### II.1 K nearest Neighbours Method and Bayes classifier with KNN for density estimation

#### II.1.1 Python Code

---

```
1  # In[1]:
2  # Import Relevant Libraries
3  import numpy as np
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  from sklearn.metrics import confusion_matrix
7  from mlxtend.plotting import plot_confusion_matrix
8  from sklearn.metrics import accuracy_score
9
10 import torch
11 from tqdm import tqdm
12
13 import torch.nn as nn
14 import torch.nn.functional as F
15 import torch.optim as optim
16 from functools import partial
17
18 #####
19
20 # In[2]:
21 # Read Data
22 # train data
23 train_data = pd.read_csv("datasets/Dataset_1b/train.csv",header=None)
24
25 # shuffle dataset
26 train_data = train_data.sample(frac=1)
27
28 # get train data
29 train_data = np.array(train_data)
30
31 # test data
32 test_data = pd.read_csv("datasets/Dataset_1b/dev.csv",header=None)
33
34 # shuffle dataset
35 test_data = test_data.sample(frac=1)
36
37 # get test data
38 test_data = np.array(test_data)
39
40 #####
41
42 # In[3]:
43 # Split training data
44 # length of data for fit
```

```

45 train_len = int(np.shape(train_data)[0])
46 val_len = int(np.shape(test_data)[0]*0.5)
47 test_len = int(np.shape(test_data)[0]*0.5)
48
49 X_train = train_data[:,0:2]
50 X_val = test_data[0:val_len,0:2]
51 X_test = test_data[val_len:val_len+test_len,0:2]
52
53 y_train = train_data[:,2]
54 y_val = test_data[0:val_len,2]
55 y_test = test_data[val_len:val_len+test_len,2]
56
57 #####
58
59 # In[4]:
60
61 # Build MLFFNN with 2 hidden layers using pytorch
62
63 class Net(nn.Module):
64     def __init__(self):
65         super(Net, self).__init__()
66
67         # in_features = input dimension out_features -> to be tuned
68         self.fc1 = nn.Linear(in_features=2, out_features=4)
69
70         # out_features -> to be tuned
71         self.fc2 = nn.Linear(in_features=4, out_features=4)
72
73         # out_features = number of classes to be predicted
74         self.fc3 = nn.Linear(in_features=4, out_features=3)
75
76     def forward(self, x):
77         x = F.relu(self.fc1(x))
78         out1 = x
79         x = F.relu(self.fc2(x))
80         out2 = x
81         x = self.fc3(x)
82         return x,out1,out2
83
84 # In[5]:
85
86 # Initialize neural network class
87 net = Net()
88
89 ## transfer the model to GPU if available
90 # if torch.cuda.is_available():
91 #     print("using GPU")
92 #     net = net.cuda()
93
94 # In[6]:

```

```

95
96 #####
97 # Define a Loss function and optimizer
98
99 # Both variables are to be tuned
100 num_epochs = 500          # desired number of training epochs.
101 learning_rate = 0.001
102
103 # loss function and optimizers can be changed
104 criterion = nn.CrossEntropyLoss()
105 optimizer = optim.Adam(net.parameters(), lr=learning_rate, weight_decay=5e-4)
106
107 num_params = np.sum([p.nelement() for p in net.parameters()])
108 print(num_params, ' parameters')
109
110 # In[7]:
111 # create dataloader for loading data
112
113 class myDataset(torch.utils.data.Dataset):
114
115     #'Characterizes a dataset for PyTorch'
116     def __init__(self, X, y, total_samples):
117         #'Initialization'
118         self.X = X
119         self.y = y
120         self.total_samples = total_samples
121
122     def __len__(self):
123         #'Denotes the total number of samples'
124         return self.total_samples
125
126     def __getitem__(self, index):
127         #'Generates one sample of data'
128         # Select sample
129         # Load data and get label
130
131         x_data = self.X[index, :]
132         y_data = self.y[index]
133
134         return x_data, y_data
135
136 # batch size can be changed to make epochs faster
137 params = {'batch_size': 16,
138           'shuffle': False,
139           'num_workers': 0}
140
141 # training dataset
142 training_set = myDataset(X_train, y_train, train_len)
143
144 training_generator = torch.utils.data.DataLoader(training_set, **params)

```

```

145
146 # validation dataset
147 validation_set = myDataset(X_val,y_val,val_len)
148 validation_generator = torch.utils.data.DataLoader(validation_set, **params)
149
150 # test dataset
151 test_set = myDataset(X_test,y_test,test_len)
152
153 test_generator = torch.utils.data.DataLoader(test_set, **params)
154
155 # In[8]:
156
157 def validation(model, loader):
158     total_loss = 0
159     accuracy = []
160     tq = partial(tqdm, position=0, leave=True)
161
162     model.eval()
163     with torch.no_grad():
164         for X, y in tq(loader):
165             X = X.float()
166             y = y.long()
167
168             # if torch.cuda.is_available():
169             #     X = X.cuda()
170             #     y = y.cuda()
171
172             prediction,_,_ = model(X)
173
174             loss = criterion(prediction, y)
175
176             prediction = F.softmax(prediction)
177
178             acc = np.mean(np.array((torch.argmax(prediction,1) == y)))*100
179
180
181             total_loss += loss.item()
182             accuracy.append(acc)
183
184     # print('Validation Accuracy: ', np.mean(np.array(accuracy)))
185     return total_loss/len(loader),np.mean(np.array(accuracy))
186
187
188 # In[9]:
189
190 def test(model, loader):
191     y_pred = []
192     accuracy = []
193     tq = partial(tqdm, position=0, leave=True)
194

```

```

195     model.eval()
196     with torch.no_grad():
197         for X, y in tq(loader):
198             X = X.float()
199             y = y.long()
200
201             # if torch.cuda.is_available():
202             #     X = X.cuda()
203             #     y = y.cuda()
204
205             prediction,_,_ = model(X)
206
207             prediction = F.softmax(prediction)
208
209             acc = np.mean(np.array((torch.argmax(prediction,1) == y)))*100
210
211             accuracy.append(acc)
212
213             prediction = torch.argmax(prediction,1)
214             y_pred = y_pred + list(np.array(prediction))
215
216     print('Test Accuracy: ', np.mean(np.array(accuracy)))
217     return y_pred,np.mean(np.array(accuracy))
218
219 # In[]:
220
221 def surface_plot(x1_grid,x2_grid,out1,out2,out3):
222
223     for i in range(0,4):
224         x3_grid = out1[:,i].detach().numpy()
225         x3_grid = x3_grid.reshape(np.shape(x1_grid))
226
227         fig = plt.figure(figsize=(16,12))
228         ax = fig.add_subplot(111,projection='3d')
229         ax.plot_surface(x1_grid,x2_grid,x3_grid,cmap='rainbow')
230         ax.set_xlabel('X-axis(x1)',fontsize = 15)
231         ax.set_ylabel('Y-axis(x2)',fontsize = 15)
232         ax.set_zlabel('Z-axis(Node Output)',fontsize = 15)
233         ax.set_title('Hidden Layer 1: Node' + str(i+1), fontsize=20)
234
235         x3_grid = out2[:,i].detach().numpy()
236         x3_grid = x3_grid.reshape(np.shape(x1_grid))
237
238         fig = plt.figure(figsize=(16,12))
239         ax = fig.add_subplot(111,projection='3d')
240         ax.plot_surface(x1_grid,x2_grid,x3_grid,cmap='rainbow')
241         ax.set_xlabel('X-axis(x1)',fontsize = 15)
242         ax.set_ylabel('Y-axis(x2)',fontsize = 15)
243         ax.set_zlabel('Z-axis(Node Output)',fontsize = 15)
244         ax.set_title('Hidden Layer 2: Node' + str(i+1), fontsize=20)

```



```

245
246     for i in range(0,3):
247         x3_grid = out3[:,i].detach().numpy()
248         x3_grid = x3_grid.reshape(np.shape(x1_grid))
249
250         fig = plt.figure(figsize=(16,12))
251         ax = fig.add_subplot(111,projection='3d')
252         ax.plot_surface(x1_grid,x2_grid,x3_grid,cmap='rainbow')
253         ax.set_xlabel('X-axis(x1)',fontsize = 15)
254         ax.set_ylabel('Y-axis(x2)',fontsize = 15)
255         ax.set_zlabel('Z-axis(Node Output)',fontsize = 15)
256         ax.set_title('Output Layer: Node' + str(i+1), fontsize=20)
257
258
259
260
261     # In[10]:
262
263     tq = partial(tqdm, position=0, leave=True)
264
265     print('Start Training')
266     train_loss_list = []
267     train_accuracy_list = []
268
269     validation_loss_list = []
270     validation_accuracy_list = []
271
272     for epoch in range(0,num_epochs):
273         print('epoch ', epoch + 1)
274         loss = 0
275         train_accuracy = []
276
277         for X,y in tq(training_generator):
278
279             X = X.float()
280             y = y.long()
281             # y = torch.squeeze(y,1)
282
283             # if torch.cuda.is_available():
284             #     X = X.cuda()
285             #     y = y.cuda()
286
287
288             optimizer.zero_grad()
289
290             output,out1,out2 = net(X)
291             loss = criterion(output,y)
292
293             loss.backward()
294             optimizer.step()

```

```

295     loss += loss.item()
296
297     prediction = F.softmax(output)
298
299     accuracy = np.mean(np.array((torch.argmax(prediction,1) == y)))*100
300     train_accuracy.append(accuracy)
301
302
303
304
305     train_loss_list.append([loss/len(training_generator)])
306     # print(loss/len(training_generator))
307
308     train_accuracy_list.append(np.mean(np.array(train_accuracy)))
309
310     val_loss, val_accuracy = validation(net, validation_generator)
311
312     validation_loss_list.append(val_loss)
313     validation_accuracy_list.append(val_accuracy)
314
315     if epoch == 0 or epoch == 4 or epoch == 19 or epoch == 99
316     or epoch == num_epochs-1:
317         torch.save(net.state_dict(), 'models/model-'+str(epoch)+'.pth')
318
319
320     print(train_accuracy_list[-1])
321     print(validation_accuracy_list[-1])
322
323     # In[]:
324     net = Net()
325
326     # Import saved Model
327     net.load_state_dict(torch.load('models/model-499.pth'))
328     net.eval()
329
330     # In[11]:
331
332     # plt.plot(train_loss_list, label='Training Loss')
333     plt.plot(validation_loss_list, label='Validation Loss')
334     plt.xlabel('Number of Epochs')
335     plt.ylabel('Loss')
336     plt.title('Loss curve')
337     plt.legend()
338     plt.plot()
339
340     # In[12]:
341
342     plt.plot(train_accuracy_list, label='Training Accuracy')
343     plt.plot(validation_accuracy_list, label='Validation Accuracy')
344     plt.xlabel('Number of Epochs')

```

```

345 plt.ylabel('Accuracy')
346 plt.title('Accuracy curve')
347 plt.legend()
348 plt.show()
349
350 # In[13]:
351
352 y_pred, test_accuracy = test(net, test_generator)
353 print(test_accuracy)
354 c_matrix = confusion_matrix(y_test, y_pred)
355
356 afig, ax = plot_confusion_matrix(conf_mat=c_matrix, figsize=(7,7), cmap=plt.cm.RdBu)
357 ax.set(title = "Confusion Matrix for test data")
358
359
360 y_pred, train_accuracy = test(net, training_generator)
361
362 c_matrix = confusion_matrix(y_train, y_pred)
363
364 afig, ax = plot_confusion_matrix(conf_mat=c_matrix, figsize=(7,7), cmap=plt.cm.RdBu)
365 ax.set(title = "Confusion Matrix for train data")
366
367 # In[14]:
368
369 # Useful for Decision Region Plots #####
370 # define bounds of the domain
371 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
372 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
373
374 # define the x and y scale
375 x1_grid = np.arange(min1, max1, 0.1)
376 x2_grid = np.arange(min2, max2, 0.1)
377
378 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
379
380 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
381 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
382
383 x = np.hstack((c1, c2))
384
385 tq = partial(tqdm, position=0, leave=True)
386
387 net.eval()
388 y_pred = []
389
390 with torch.no_grad():
391     x = torch.from_numpy(x)
392     x = x.float()
393
394     # if torch.cuda.is_available():

```

```

395     # x = x.cuda()
396
397     output,out1,out2 = net(x)
398     prediction = F.softmax(output)
399     prediction = torch.argmax(prediction,1)
400     y_pred = y_pred + list(np.array(prediction.cpu()))
401     surface_plot(x1_grid,x2_grid,out1, out2, output)
402
403 y_pred = np.array(y_pred)
404
405 x3_grid = y_pred.reshape(x1_grid.shape)
406
407 # In[:
408 fig = plt.figure(figsize=(11,11))
409 ax = fig.add_subplot(111)
410 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Pastel1')
411 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
412 # ax.scatter(X_test[:,0],X_test[:,1],marker='x')
413 ax.set_xlabel('x1',fontsize=20)
414 ax.set_ylabel('x2',fontsize=20)
415 ax.set_title('MLFFNN', fontsize=20)
416
417 #####
418 # In[15]:
419
420 # Non-linear SVM
421 from sklearn.svm import SVC
422 from sklearn.multiclass import OneVsRestClassifier
423
424 # In[16]:
425
426 # Gaussian Kernel
427 clf = OneVsRestClassifier(SVC(C=4,kernel='rbf')).fit(X_train, y_train)
428
429
430 y_pred = clf.predict(X_train)
431 train_accuracy = accuracy_score(y_train,y_pred)*100
432 print("train accuracy: " , train_accuracy)
433
434 c_matrix = confusion_matrix(y_train, y_pred)
435
436 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdBu)
437 ax.set(title = "Confusion Matrix for train data")
438
439 y_pred = clf.predict(X_val)
440 val_accuracy = accuracy_score(y_val,y_pred)*100
441 print("validation accuracy: " , val_accuracy)
442
443 y_pred = clf.predict(X_test)
444 test_accuracy = accuracy_score(y_test,y_pred)*100

```

```

445 print("test accuracy: " , test_accuracy)
446
447 c_matrix = confusion_matrix(y_test, y_pred)
448
449 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdBu)
450 ax.set(title = "Confusion Matrix for test data")
451
452 # In[17]:
453
454 # Poly Kernel
455 clf = OneVsRestClassifier(SVC(kernel='poly',degree=11)).fit(X_train, y_train)
456
457 y_pred = clf.predict(X_train)
458 train_accuracy = accuracy_score(y_train,y_pred)*100
459 print("train accuracy: " , train_accuracy)
460
461 c_matrix = confusion_matrix(y_train, y_pred)
462
463 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdBu)
464 ax.set(title = "Confusion Matrix for train data")
465
466 y_pred = clf.predict(X_val)
467 val_accuracy = accuracy_score(y_val,y_pred)*100
468 print("validation accuracy: " , val_accuracy)
469
470 y_pred = clf.predict(X_test)
471 test_accuracy = accuracy_score(y_test,y_pred)*100
472 print("test accuracy: " , test_accuracy)
473
474 c_matrix = confusion_matrix(y_test, y_pred)
475
476 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdBu)
477 ax.set(title = "Confusion Matrix for test data")
478
479
480 # In[18]:
481
482
483 # Useful for Decision Region Plots #####
484 # define bounds of the domain
485 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
486 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
487
488 # define the x and y scale
489 x1_grid = np.arange(min1, max1, 0.1)
490 x2_grid = np.arange(min2, max2, 0.1)
491
492 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
493
494 c1, c2 = x1_grid.flatten(), x1_grid.flatten()

```

```

495 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
496
497 x = np.hstack((c1,c2))
498
499 y_pred = clf.predict(x)
500
501 x3_grid = y_pred.reshape(x1_grid.shape)
502
503
504 fig = plt.figure(figsize=(11,11))
505 ax = fig.add_subplot(111)
506 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Pastel1')
507 ax.scatter(X_train[:,0],X_train[:,1],marker='x',label='Training Points')
508 for i in range(0,3):
509     support_vectors = clf.estimators_[i].support_vectors_
510     ax.scatter(support_vectors[:,0],support_vectors[:,1],marker='x',
511               label='Support Vectors '+str(i))
512
513 ax.set_xlabel('x1',fontsize=20)
514 ax.set_ylabel('x2',fontsize=20)
515 ax.set_title('Support Vector Classifier', fontsize=20)
516 ax.legend()

```

---

## III Static Pattern Classification on Real World Dataset 2A

### III.1 Python Code

---

```
1  # In[1]:
2  # Import Relevant Libraries
3  import numpy as np
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  from sklearn.metrics import confusion_matrix
7  from mlxtend.plotting import plot_confusion_matrix
8  from sklearn.metrics import accuracy_score
9
10 import torch
11 from tqdm import tqdm
12
13 import torch.nn as nn
14 import torch.nn.functional as F
15 import torch.optim as optim
16 from functools import partial
17
18 #####
19
20 # In[2]:
21 #####
22 # train data
23 coast_train_data = np.array(pd.read_csv('datasets/Dataset_2a/coast
24                                         /train.csv'))[:,1:]
25 forest_train_data = np.array(pd.read_csv('datasets/Dataset_2a/forest
26                                           /train.csv'))[:,1:]
27 mountain_train_data = np.array(pd.read_csv('datasets/Dataset_2a/mountain
28                                             /train.csv'))[:,1:]
29 country_train_data = np.array(pd.read_csv('datasets/Dataset_2a/opencountry
30                                            /train.csv'))[:,1:]
31 street_train_data = np.array(pd.read_csv('datasets/Dataset_2a/street
32                                           /train.csv'))[:,1:]
33
34 # populate class vector
35 target_0 = np.zeros((coast_train_data.shape[0],1))
36 target_1 = np.ones((forest_train_data.shape[0],1))
37 target_2 = np.ones((mountain_train_data.shape[0],1))*2
38 target_3 = np.ones((country_train_data.shape[0],1))*3
39 target_4 = np.ones((street_train_data.shape[0],1))*4
40
41 X_train = np.array(np.concatenate((coast_train_data,forest_train_data,
42                                     mountain_train_data,country_train_data,street_train_data),
43                                   axis=0),dtype=float)
44 y_train = np.array(np.concatenate((target_0,target_1,target_2,
45                                     target_3,target_4),axis=0),dtype=float)
46 y_train = np.reshape(y_train,(np.size(y_train,)))
```

```

47
48 # test & val data
49 coast_test_data = np.array(pd.read_csv('datasets/Dataset_2a/coast
50                                         /dev.csv'))[:,1:]
51 forest_test_data = np.array(pd.read_csv('datasets/Dataset_2a/forest
52                                         /dev.csv'))[:,1:]
53 mountain_test_data = np.array(pd.read_csv('datasets/Dataset_2a/mountain
54                                         /dev.csv'))[:,1:]
55 country_test_data = np.array(pd.read_csv('datasets/Dataset_2a/opencountry
56                                         /dev.csv'))[:,1:]
57 street_test_data = np.array(pd.read_csv('datasets/Dataset_2a/street
58                                         /dev.csv'))[:,1:]
59
60 # populate class vector
61 target_0 = np.zeros((coast_test_data.shape[0],1))
62 target_1 = np.ones((forest_test_data.shape[0],1))
63 target_2 = np.ones((mountain_test_data.shape[0],1))*2
64 target_3 = np.ones((country_test_data.shape[0],1))*3
65 target_4 = np.ones((street_test_data.shape[0],1))*4
66
67 X_dev = np.array(np.concatenate((coast_test_data,forest_test_data,
68                                  mountain_test_data,country_test_data,
69                                  street_test_data),axis=0),dtype=float)
70 y_dev = np.array(np.concatenate((target_0,target_1,
71                                  target_2,target_3,target_4),axis=0),dtype=float)
72 y_dev = np.reshape(y_dev,(np.size(y_dev),))
73 #####
74
75 # In[3]:
76 # Split training data
77 # length of data for fit
78 train_len = int(np.shape(X_train)[0])
79
80 index1 = np.random.choice([True,False],size=np.shape(X_dev)[0])
81 index2 = ~index1
82
83 X_val = X_dev[index1,:]
84 X_test = X_dev[index2,:]
85
86 y_val = y_dev[index1]
87 y_test = y_dev[index2]
88
89 val_len = int(np.shape(X_val)[0])
90 test_len = int(np.shape(X_test)[0])
91
92
93 #####
94
95 # In[4]:
96

```



```

97  # Build MLFFNN with 2 hidden layers using pytorch
98
99  class Net(nn.Module):
100      def __init__(self):
101          super(Net, self).__init__()
102
103          # in_features = input dimension out_features -> to be tuned
104          self.fc1 = nn.Linear(in_features=24, out_features=500)
105
106          # out_features -> to be tuned
107          self.fc2 = nn.Linear(in_features=500, out_features=250)
108
109          # out_features = number of classes to be predicted
110          self.fc3 = nn.Linear(in_features=250, out_features=5)
111
112      def forward(self, x):
113          x = F.relu(self.fc1(x))
114          out1 = x
115          x = F.relu(self.fc2(x))
116          out2 = x
117          x = self.fc3(x)
118          return x, out1, out2
119
120  # In[5]:
121
122  # Initialize neural network class
123  net = Net()
124
125  ## transfer the model to GPU if available
126  # if torch.cuda.is_available():
127  #     print("using GPU")
128  #     net = net.cuda()
129
130  # In[6]:
131
132  #####
133  # Define a Loss function and optimizer
134
135  # Both variables are to be tuned
136  num_epochs = 500          # desired number of training epochs.
137  learning_rate = 0.001
138
139  # loss function and optimizers can be changed
140  criterion = nn.CrossEntropyLoss()
141  optimizer = optim.Adam(net.parameters(), lr=learning_rate,
142                          weight_decay=5e-4)
143
144  num_params = np.sum([p.nelement() for p in net.parameters()])
145  print(num_params, ' parameters')
146

```

```

147 # In[7]:
148 # create dataloader for loading data
149
150 class myDataset(torch.utils.data.Dataset):
151
152     #Characterizes a dataset for PyTorch'
153     def __init__(self,X,y,total_samples):
154         #'Initialization'
155         self.X = X
156         self.y = y
157         self.total_samples = total_samples
158
159     def __len__(self):
160         #'Denotes the total number of samples'
161         return self.total_samples
162
163     def __getitem__(self, index):
164         #'Generates one sample of data'
165         # Select sample
166         # Load data and get label
167
168         x_data = self.X[index,:]
169         y_data = self.y[index]
170
171         return x_data,y_data
172
173     # batch size can be changed to make epochs faster
174     params = {'batch_size': 32,
175              'shuffle': False,
176              'num_workers': 0}
177
178     # training dataset
179     training_set = myDataset(X_train,y_train,train_len)
180
181     training_generator = torch.utils.data.DataLoader(training_set,
182                                                       **params)
183
184     # validation dataset
185     validation_set = myDataset(X_val,y_val,val_len)
186     validation_generator = torch.utils.data.DataLoader(validation_set,
187                                                         **params)
188
189     # test dataset
190     test_set = myDataset(X_test,y_test,test_len)
191
192     test_generator = torch.utils.data.DataLoader(test_set,
193                                                    **params)
194
195 # In[8]:
196

```

```

197 def validation(model, loader):
198     total_loss = 0
199     accuracy = []
200     tq = partial(tqdm, position=0, leave=True)
201
202     model.eval()
203     with torch.no_grad():
204         for X, y in tq(loader):
205             X = X.float()
206             y = y.long()
207
208             # if torch.cuda.is_available():
209             #     X = X.cuda()
210             #     y = y.cuda()
211
212             prediction, _, _ = model(X)
213
214             loss = criterion(prediction, y)
215
216             prediction = F.softmax(prediction)
217
218             acc = np.mean(np.array((torch.argmax(prediction, 1) == y))) * 100
219
220
221             total_loss += loss.item()
222             accuracy.append(acc)
223
224             # print('Validation Accuracy: ', np.mean(np.array(accuracy)))
225     return total_loss / len(loader), np.mean(np.array(accuracy))
226
227
228 # In[9]:
229
230 def test(model, loader):
231     y_pred = []
232     accuracy = []
233     tq = partial(tqdm, position=0, leave=True)
234
235     model.eval()
236     with torch.no_grad():
237         for X, y in tq(loader):
238             X = X.float()
239             y = y.long()
240
241             # if torch.cuda.is_available():
242             #     X = X.cuda()
243             #     y = y.cuda()
244
245             prediction, _, _ = model(X)
246

```

```

247         prediction = F.softmax(prediction)
248
249         acc = np.mean(np.array((torch.argmax(prediction,1) == y)))*100
250
251         accuracy.append(acc)
252
253         prediction = torch.argmax(prediction,1)
254         y_pred = y_pred + list(np.array(prediction))
255
256         print('Test Accuracy: ', np.mean(np.array(accuracy)))
257         return y_pred,np.mean(np.array(accuracy))
258
259
260     # In[10]:
261
262     tq = partial(tqdm, position=0, leave=True)
263
264     print('Start Training')
265     train_loss_list = []
266     train_accuracy_list = []
267
268     validation_loss_list = []
269     validation_accuracy_list = []
270
271     for epoch in range(0,num_epochs):
272         print('epoch ', epoch + 1)
273         loss = 0
274         train_accuracy = []
275
276         for X,y in tq(training_generator):
277
278             X = X.float()
279             y = y.long()
280             # y = torch.squeeze(y,1)
281
282             # if torch.cuda.is_available():
283             #     X = X.cuda()
284             #     y = y.cuda()
285
286             optimizer.zero_grad()
287
288             output,out1,out2 = net(X)
289             loss = criterion(output,y)
290
291             loss.backward()
292             optimizer.step()
293
294             loss += loss.item()
295
296         prediction = F.softmax(output)

```

297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346

```
accuracy = np.mean(np.array((torch.argmax(prediction,1) == y)))*100
train_accuracy.append(accuracy)

train_loss_list.append([loss/len(training_generator)])
# print(loss/len(training_generator))

train_accuracy_list.append(np.mean(np.array(train_accuracy)))

val_loss, val_accuracy = validation(net, validation_generator)

validation_loss_list.append(val_loss)
validation_accuracy_list.append(val_accuracy)

# if epoch == 0 or epoch == 4 or epoch == 19 or epoch == 99 or epoch == num_epochs
#     torch.save(net.state_dict(), 'models/model-'+str(epoch)+'.pth')

print(train_accuracy_list[-1])
print(validation_accuracy_list[-1])

# In[]:
net = Net()

# Import saved Model
net.load_state_dict(torch.load('models/model-499.pth'))
net.eval()

# plt.plot(train_loss_list, label='Training Loss')
plt.plot(validation_loss_list, label='Validation Loss')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.title('Loss curve')
plt.legend()
plt.plot()

# In[12]:

plt.plot(train_accuracy_list, label='Training Accuracy')
plt.plot(validation_accuracy_list, label='Validation Accuracy')
plt.xlabel('Number of Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy curve')
plt.legend()
plt.show()

# In[13]:
```

```

347 y_pred, test_accuracy = test(net,test_generator)
348 c_matrix = confusion_matrix(y_test, y_pred)
349
350 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7)
351                                ,cmap=plt.cm.RdBu)
352 ax.set(title = "Confusion Matrix for test data")
353
354 y_pred, train_accuracy = test(net,training_generator)
355
356 c_matrix = confusion_matrix(y_train, y_pred)
357
358 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7)
359                                ,cmap=plt.cm.RdBu)
360 ax.set(title = "Confusion Matrix for train data")
361
362 #####
363 # In[15]:
364
365 # Non-linear SVM
366 from sklearn.svm import SVC
367 from sklearn.multiclass import OneVsRestClassifier
368
369 # Gaussian Kernel
370 clf = OneVsRestClassifier(SVC(C=15,kernel='rbf')).fit(X_train, y_train)
371 y_pred = clf.predict(X_train)
372 train_accuracy = accuracy_score(y_train,y_pred)*100
373 print("train accuracy: " , train_accuracy)
374
375 c_matrix = confusion_matrix(y_train, y_pred)
376
377 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),
378                                cmap=plt.cm.RdBu)
379 ax.set(title = "Confusion Matrix for train data")
380 y_pred = clf.predict(X_val)
381 val_accuracy = accuracy_score(y_val,y_pred)*100
382 print("validation accuracy: " , val_accuracy)
383
384 y_pred = clf.predict(X_test)
385 test_accuracy = accuracy_score(y_test,y_pred)*100
386 print("test accuracy: " , test_accuracy)
387
388 c_matrix = confusion_matrix(y_test, y_pred)
389
390 afig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),
391                                cmap=plt.cm.RdBu)
392 ax.set(title = "Confusion Matrix for test data")

```

---