

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	2
1.3	Linear SVM	6
2	Dataset 1B	6
2.1	MLFFNN	6
2.2	Non-Linear SVM	13
3	Dataset 2A	13
3.1	MLFFNN	13
3.2	Gaussian-kernel SVM	14

All codes excluding the modules are converted to .py files from IPython Notebooks

1 Dataset 1A

1.1 Perceptron

1.2 MLFFNN

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

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Assignment 3 - 1A (MLFFNN)
5  #
6  # Team members:
7  # - N Sowmya Manojna (BE17B007)
8  # - Thakkar Riya Anandbhai (PH17B010)
9  # - Chaithanya Krishna Moorthy (PH17B011)
10
11 # ## Importing Essential Libraries
12
13 # In[1]:
14
15
16 import numpy as np
17 import pandas as pd
18 import seaborn as sns
19 from sklearn.pipeline import Pipeline
20 from sklearn.metrics import confusion_matrix
21 from sklearn.neural_network import MLPClassifier
22 from sklearn.preprocessing import StandardScaler
23 from sklearn.model_selection import GridSearchCV
24 from sklearn.model_selection import train_test_split
25 from sklearn.model_selection import StratifiedShuffleSplit
26
27 import matplotlib.pyplot as plt
28 plt.rcParams["font.size"] = 18
29 plt.rcParams["axes.grid"] = True
30 plt.rcParams["figure.figsize"] = 12,8
31 plt.rcParams['font.serif'] = "Cambria"
32 plt.rcParams['font.family'] = "serif"
33
34 get_ipython().run_line_magic('load_ext', 'autoreload')
35 get_ipython().run_line_magic('autoreload', '2')
36
37 import warnings
38 warnings.filterwarnings("ignore")
39
40 from gridsearch import GridSearch1A
41
42
43 # ## Reading the data, Splitting it
44
45 # In[2]:
46
47
48 # Get the data
49 column_names = ["x1", "x2", "y"]
50 df = pd.read_csv("../datasets/1A/train.csv", names=column_names)
51 df_test = pd.read_csv("../datasets/1A/dev.csv", names=column_names)
52 display(df.head())
53
54 # Split dev into test and validation
55 df_val, df_test = train_test_split(df_test, test_size=0.3, random_state=42)
56 display(df_val.head())
57 display(df_test.head())
```

```

58
59
60 # In[3]:
61
62
63 X_train = df.drop("y", axis=1).to_numpy()
64 y_train = df["y"].to_numpy().astype("int")
65
66 X_val = df_val.drop("y", axis=1).to_numpy()
67 y_val = df_val["y"].to_numpy().astype("int")
68
69 X_test = df_test.drop("y", axis=1).to_numpy()
70 y_test = df_test["y"].to_numpy().astype("int")
71
72
73 # ## Training the Model
74
75 # In[4]:
76
77
78 parameters = {"hidden_layer_sizes": [5, 8, 10, 15], "activation": ["logistic", "tanh", "...
              relu"],
              "solver": ["lbfgs", "sgd", "adam"], "batch_size": [100, ...
              200],
              "alpha": [0, 0.0001], "learning_rate": ["constant", "adaptive"...
              , "invscaling"],
              }
79
80 mlp = MLPClassifier(random_state=1)
81
82 clf = GridSearch1A(mlp, parameters)
83 clf.fit(X_train, y_train, X_val, y_val)
84 result_df = pd.DataFrame(clf.cv_results_)
85 result_df.to_csv("../parameter_search/1A_MLFFNN_train_val.csv")
86 result_df.head()
87
88
89 # In[5]:
90
91
92 print("Best Parameters Chosen:")
93 for i in clf.best_params_:
94     print("    - ", i, ": ", clf.best_params_[i], sep="")
95
96 best_mlp = MLPClassifier(random_state=1, **clf.best_params_)
97 best_mlp.fit(X_train, y_train)
98
99
100 # ## Best Model Predictions
101
102 # In[6]:
103
104
105 y_pred = best_mlp.predict(X_train)
106 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
107 conf_mat = confusion_matrix(y_train, y_pred)
108 plt.figure()
109 sns.heatmap(conf_mat, annot=True)
110 plt.title("1A - Train Confusion Matrix (MLFFNN)")
111 plt.xlabel("Predicted Class")
112 plt.ylabel("Actual Class")
113 plt.savefig("images/1A_MLFFNN_train_confmat.png")
114 plt.show()
115
116 y_val_pred = best_mlp.predict(X_val)
117 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
118 val_conf_mat = confusion_matrix(y_val, y_val_pred)
119 plt.figure()
120 sns.heatmap(val_conf_mat, annot=True)
121 plt.title("1A - Validation Confusion Matrix (MLFFNN)")
122 plt.xlabel("Predicted Class")
123 plt.ylabel("Actual Class")

```

```

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

```

The helper functions used are as follows:

```

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     self.params_list = params_list
29
30     def fit(self, X_train, y_train, X_val, y_val):
31         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
32
33         self.params_ = defaultdict(list)
34         self.acc_list_ = []
35         self.val_acc_list_ = []
36         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, \

```

```

93                                     "alpha":a})
94     self.params_list = params_list
95
96     def fit(self, X_train, y_train, X_val, y_val):
97         self.cv_results_ = pd.DataFrame(columns=self.params_keys)
98
99         self.params_ = defaultdict(list)
100        self.acc_list_ = []
101        self.val_acc_list_ = []
102        self.t_inv_list_ = []
103
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        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)], ...
               "activation":["logistic", "relu"], "batch_size":[50, ...

```



```

    100, 200], "early_stopping":[True, False],                "learning_rate":["...
    constant", "adaptive", "invscaling"],                  "alpha":[0.01, 0.001]
78         }
79
80 mlp = MLPClassifier(random_state=1)
81
82 clf = GridSearch1B(mlp, parameters)
83 clf.fit(X_train, y_train, X_val, y_val)
84 result_df = pd.DataFrame(clf.cv_results_)
85 result_df.to_csv("../parameter_search/1B_MLFFNN_train_val.csv")
86 result_df.head(10)
87
88
89 # In[5]:
90
91
92 print("Best Parameters Chosen:")
93 for i in clf.best_params_:
94     print("    - ", i, ": ", clf.best_params_[i], sep="")
95
96 best_mlp = MLPClassifier(random_state=1, **clf.best_params_)
97 best_mlp.fit(X_train, y_train)
98
99
100 # ## Best Model Predictions
101
102 # In[6]:
103
104
105 y_pred = best_mlp.predict(X_train)
106 print("Accuracy:", 100*np.sum(y_pred==y_train)/y_train.size)
107 conf_mat = confusion_matrix(y_train, y_pred)
108 plt.figure(figsize=(8,6))
109 sns.heatmap(conf_mat, annot=True)
110 plt.title("1B - Train Confusion Matrix (MLFFNN)")
111 plt.xlabel("Predicted Class")
112 plt.ylabel("Actual Class")
113 plt.savefig("images/1B_MLFFNN_train_confmat.png")
114 plt.show()
115
116 y_val_pred = best_mlp.predict(X_val)
117 print("Validation Accuracy:", 100*np.sum(y_val_pred==y_val)/y_val.size)
118 val_conf_mat = confusion_matrix(y_val, y_val_pred)
119 plt.figure(figsize=(8,6))
120 sns.heatmap(val_conf_mat, annot=True)
121 plt.title("1B - Validation Confusion Matrix (MLFFNN)")
122 plt.xlabel("Predicted Class")
123 plt.ylabel("Actual Class")
124 plt.savefig("images/1B_MLFFNN_val_confmat.png")
125 plt.show()
126
127 y_test_pred = best_mlp.predict(X_test)
128 print("Test Accuracy:", 100*np.sum(y_test_pred==y_test)/y_test.size)
129 test_conf_mat = confusion_matrix(y_test, y_test_pred)
130 plt.figure(figsize=(8,6))
131 sns.heatmap(test_conf_mat, annot=True)
132 plt.title("1B - Test Confusion Matrix (MLFFNN)")
133 plt.xlabel("Predicted Class")
134 plt.ylabel("Actual Class")
135 plt.savefig("images/1B_MLFFNN_test_confmat.png")
136 plt.show()
137
138
139 # ## Visualising the decision boundaries
140
141 # In[7]:
142
143
144 h = 0.02

```

```

145 x_min, x_max = X_train[:,0].min() - .5, X_train[:,0].max() + .5
146 y_min, y_max = X_train[:,1].min() - .5, X_train[:,1].max() + .5
147
148 xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
149 Z_pro = np.argmax(best_mlp.predict_proba(np.c_[xx.ravel(), yy.ravel()]), axis=1)
150 Z_pro = Z_pro.reshape(xx.shape)
151
152 color_list = ["springgreen", "gold", "palevioletred", "royalblue"]
153 plt.figure(figsize=(12,8))
154 plt.title("1B - Decision Region Plot (MLFFNN)")
155 plt.contourf(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=0.1)
156 plt.contour(xx, yy, Z_pro, np.unique(Z_pro).size-1, colors=color_list, alpha=1)
157 plt.scatter(X_train[:,0], X_train[:,1], c=[color_list[i] for i in y_train])
158 plt.xlabel("X1")
159 plt.ylabel("X2")
160 plt.savefig("images/1B_MLFFNN_Decision_Plot.png")
161 plt.show()
162
163
164 # ## Visualising Neuron Responses
165
166 # In[8]:
167
168
169 def get_values(weights, biases, X_train):
170     ip = X_train.T
171     h1 = weights[0].T @ ip + biases[0].reshape(-1,1)
172     a1 = np.maximum(0, h1)
173     h2 = weights[1].T @ a1 + biases[1].reshape(-1,1)
174     a2 = np.maximum(0, h2)
175     h3 = weights[2].T @ a2 + biases[2].reshape(-1,1)
176     pred = np.exp(h3)/np.sum(np.exp(h3))
177
178     return a1, a2, pred
179
180
181 # In[9]:
182
183
184 from matplotlib import cm
185 from mpl_toolkits import mplot3d
186 from mpl_toolkits.mplot3d import axes3d
187 grid = np.c_[xx.ravel(), yy.ravel()]
188
189 for epochs in [1, 5, 20, 100]:
190     mlp = MLPClassifier(random_state=1, max_iter=epochs, **clf.best_params_)
191     mlp.fit(X_train, y_train)
192
193     weights = mlp.coefs_
194     biases = mlp.intercepts_
195
196     a1, a2, op = get_values(weights, biases, grid)
197     a1 = a1.reshape(a1.shape[0], *xx.shape)
198     a2 = a2.reshape(a2.shape[0], *xx.shape)
199     op = op.reshape(op.shape[0], *xx.shape)
200
201
202     for i in range(a1.shape[0]):
203         fig = plt.figure(figsize=(8,8))
204         ax = plt.axes(projection="3d")
205
206         # ax.contour3D(xx, yy, a1[i,:], 500)
207         ax.contourf(xx, yy, a1[i,:], 500, cmap=cm.CMRmap)
208         ax.set_xlabel("X1")
209         ax.set_ylabel("X2")
210         ax.set_zlabel("HL1-Neuron "+str(i+1));
211         ax.set_title("Epoch: " + str(epochs) + "; Surface for Layer 1, Neuron "+str(...
212                       i+1))
213     plt.tight_layout()

```

```

213     plt.savefig("images/1B_MLFFNN_E"+str(epochs)+"_HL1_N"+str(i+1)+".png")
214     plt.show()
215
216     for i in range(a2.shape[0]):
217         fig = plt.figure(figsize=(8,8))
218         ax = plt.axes(projection="3d")
219
220         # ax.contour3D(xx, yy, a2[i,:], 500)
221         ax.contourf(xx, yy, a2[i,:], 500, cmap=cm.CMRmap)
222         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[ ]:

```

The helper functions used are as follows:

```

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

```

```

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

```

2.2 Non-Linear SVM

3 Dataset 2A

3.1 MLFFNN

The helper function used is as follows:

```

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)

```

3.2 Gaussian-kernel SVM