

```

1  """
2  Synthetic1 Dataset (Question 2(a))
3  Name: Haolun Cheng
4  EE559 HW4
5  """
6  from __future__ import print_function
7  import copy
8  import csv
9  import random as rm
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from plotDecBoundaries import plotDecBoundaries
13
14
15 class Vector0p(object):
16     @staticmethod
17     def element_multiply(x, y):
18         return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))
19
20     @staticmethod
21     def element_add(x, y):
22         return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))
23
24     @staticmethod
25     def scala_multiply(v, s):
26         return map(lambda e: e * s, v)
27
28
29 class Perceptron(object):
30     def __init__(self, input_num):
31         self.weights = [0.1] * (input_num + 1)
32         self.bias = 0.0
33
34     def __str__(self):
35         return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)
36
37     def train(self, shuffled_set, iteration):
38         JnX = []
39         weight_vectors_list = []
40         weight = np.array(self.weights)
41         wrongpts = 0
42         Jw = 0
43         ZnX = 0
44         for m in range(iteration):
45             for n in range(len(shuffled_set)):
46                 i = (m - 1) * len(shuffled_set) + n
47                 x = np.array(shuffled_set[n][:-1]).astype(float)
48                 if(float(shuffled_set[n][-1]) == 1):
49                     ZnX = 1
50                 else:
51                     ZnX = -1
52                 if(np.dot(weight, ZnX*x) <= 0):
53                     wrongpts += 1
54                     weight += ZnX * x
55                     Jw += (-1) * np.dot(weight, ZnX * x)
56                 if i >= 9500:
57                     JnX.append(Jw)
58                     weight_vectors_list.append(weight)
59             # Two halting conditions

```

```
60         if wrongpts == 0:
61             JnX.append(Jw)
62             weight_vectors_list.append(weight)
63             print("i.1 reached")
64             break
65
66         if i > 10000:
67             print("i.2 reached")
68             break
69
70         wrongpts = 0
71
72         return np.array(JnX).astype(float),
73         np.array(weight_vectors_list).astype(float)
74
75     def calculate_error_rate(self, dataset, minwvalue):
76         dataset_copy = copy.deepcopy(dataset)
77         incorrect_count = 0
78         for line in dataset_copy:
79             input_vec = np.array(line[:-1]).astype(float)
80             if np.dot(input_vec, minwvalue) > 0:
81                 line.append('1')
82             else:
83                 line.append('2')
84
85         for i in dataset_copy:
86             if i[-1] != i[-2]:
87                 incorrect_count += 1
88         return float(incorrect_count) / len(dataset_copy)
89
90 def f(x):
91     return 1 if x > 0 else 0
92
93
94 def read_dataset(data_path):
95     set_as_list = []
96     with open(data_path, "r") as f:
97         dataset = csv.reader(f)
98         for eachLine in dataset:
99             if len(eachLine) != 0:
100                 eachLine.insert(0, 1)
101                 set_as_list += [eachLine]
102     return set_as_list
103
104 def read_for_plot(data_path):
105     input_vecs = []
106     with open(data_path, "r") as f:
107         dataset = csv.reader(f)
108         shuffle_set = np.array(list(dataset))
109         for line in shuffle_set:
110             x, y = line[0], line[1]
111             input_vecs.append((float(x), float(y)))
112     return input_vecs
113
114
115 def train_and_perceptron():
116     p = Perceptron(2)
117     train_list = read_dataset("./synthetic1_train.csv")
118     test_list = read_dataset("./synthetic1_test.csv")
```

```

119
120     # shuffle data points
121     rm.shuffle(train_list)
122     shuffled_train_set = np.array(train_list)
123     rm.shuffle(test_list)
124     shuffled_test_set = np.array(test_list)
125
126     train_datapts = []
127     train_labels = []
128     for line in shuffled_train_set:
129         datap1, label1 = line[1:3], line[-1]
130         train_datapts.append(datap1)
131         train_labels.append(label1)
132
133     test_datapts = []
134     test_labels = []
135     for line in shuffled_test_set:
136         datap2, label2 = line[1:3], line[-1]
137         test_datapts.append(datap2)
138         test_labels.append(label2)
139
140     Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
141     minJvalue = min(Jvalue)
142     minPos = 0
143     for i in range(Jvalue.shape[0]):
144         if Jvalue[i] == minJvalue:
145             minPos = i
146             break
147     minwvalue = weight_vectors[minPos]
148     error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
149     error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
150     print(f"The best weight vector omega (w) is {minwvalue}")
151     print(f"The criterion function value is {minJvalue}")
152     print(f"The error rate of training set is {error_rate_train_set}")
153     print(f"The error rate of test set is {error_rate_test_set}")
154
155     train_data_points = read_for_plot("./synthetic1_train.csv")
156     plt.scatter(np.array(train_data_points)[: ,0], np.array(train_data_points)
157 [:,1])
158     plt.xlabel('Feature1')
159     plt.ylabel('Feature2')
160     plt.title('Feature Plot of all Elements')
161     plt.show()
162
163     training = np.array(train_datapts).astype(float)
164     label_train = np.array(train_labels).astype(float)
165     weight_vector = minwvalue[1:]
166     plotDecBoundaries(training, label_train, weight_vector)
167     return p
168
169 if __name__ == '__main__':
170     and_perception = train_and_perceptron()

```

```

1  """
2  Synthetic2 Dataset (Question 2(b))
3  Name: Haolun Cheng
4  EE559 HW4
5  """
6  from __future__ import print_function
7  import copy
8  import csv
9  import random as rm
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from plotDecBoundaries import plotDecBoundaries
13
14
15 class Vector0p(object):
16     @staticmethod
17     def element_multiply(x, y):
18         return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))
19
20     @staticmethod
21     def element_add(x, y):
22         return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))
23
24     @staticmethod
25     def scala_multiply(v, s):
26         return map(lambda e: e * s, v)
27
28
29 class Perceptron(object):
30     def __init__(self, input_num):
31         self.weights = [0.1] * (input_num + 1)
32         self.bias = 0.0
33
34     def __str__(self):
35         return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)
36
37     def train(self, shuffled_set, iteration):
38         JnX = []
39         weight_vectors_list = []
40         weight = np.array(self.weights)
41         wrongpts = 0
42         Jw = 0
43         ZnX = 0
44         for m in range(iteration):
45             for n in range(len(shuffled_set)):
46                 i = (m - 1) * len(shuffled_set) + n
47                 x = np.array(shuffled_set[n][:-1]).astype(float)
48                 if(float(shuffled_set[n][-1]) == 1):
49                     ZnX = 1
50                 else:
51                     ZnX = -1
52                 if(np.dot(weight, ZnX*x) <= 0):
53                     wrongpts += 1
54                     weight += ZnX * x
55                     Jw += (-1) * np.dot(weight, ZnX * x)
56                 if i >= 9500:
57                     JnX.append(Jw)
58                     weight_vectors_list.append(weight)
59             # Two halting conditions

```

```
60         if wrongpts == 0:
61             JnX.append(Jw)
62             weight_vectors_list.append(weight)
63             print("i.1 reached")
64             break
65
66         if i > 10000:
67             print("i.2 reached")
68             break
69
70         wrongpts = 0
71
72     return np.array(JnX).astype(float),
73     np.array(weight_vectors_list).astype(float)
74
75 def calculate_error_rate(self, dataset, minwvalue):
76     dataset_copy = copy.deepcopy(dataset)
77     incorrect_count = 0
78     for line in dataset_copy:
79         input_vec = np.array(line[:-1]).astype(float)
80         if np.dot(input_vec, minwvalue) > 0:
81             line.append('1')
82         else:
83             line.append('2')
84
85     for i in dataset_copy:
86         if i[-1] != i[-2]:
87             incorrect_count += 1
88     return float(incorrect_count) / len(dataset_copy)
89
90 def f(x):
91     return 1 if x > 0 else 0
92
93
94 def read_dataset(data_path):
95     set_as_list = []
96     with open(data_path, "r") as f:
97         dataset = csv.reader(f)
98         for eachLine in dataset:
99             if len(eachLine) != 0:
100                 eachLine.insert(0, 1)
101                 set_as_list += [eachLine]
102     return set_as_list
103
104 def read_for_plot(data_path):
105     input_vecs = []
106     with open(data_path, "r") as f:
107         dataset = csv.reader(f)
108         shuffle_set = np.array(list(dataset))
109         for line in shuffle_set:
110             x, y = line[0], line[1]
111             input_vecs.append((float(x), float(y)))
112     return input_vecs
113
114
115 def train_and_perceptron():
116     p = Perceptron(2)
117     train_list = read_dataset("./synthetic2_train.csv")
118     test_list = read_dataset("./synthetic2_test.csv")
```

```
119
120     # shuffle data points
121     rm.shuffle(train_list)
122     shuffled_train_set = np.array(train_list)
123     rm.shuffle(test_list)
124     shuffled_test_set = np.array(test_list)
125
126     train_datapts = []
127     train_labels = []
128     for line in shuffled_train_set:
129         datap1, label1 = line[1:3], line[-1]
130         train_datapts.append(datap1)
131         train_labels.append(label1)
132
133     test_datapts = []
134     test_labels = []
135     for line in shuffled_test_set:
136         datap2, label2 = line[1:3], line[-1]
137         test_datapts.append(datap2)
138         test_labels.append(label2)
139
140     Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
141     minJvalue = min(Jvalue)
142     minPos = 0
143     for i in range(Jvalue.shape[0]):
144         if Jvalue[i] == minJvalue:
145             minPos = i
146             break
147     minwvalue = weight_vectors[minPos]
148     error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
149     error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
150     print(f"The best weight vector omega (w) is {minwvalue}")
151     print(f"The criterion function value is {minJvalue}")
152     print(f"The error rate of training set is {error_rate_train_set}")
153     print(f"The error rate of test set is {error_rate_test_set}")
154
155     train_data_points = read_for_plot("./synthetic2_train.csv")
156     plt.scatter(np.array(train_data_points)[: ,0], np.array(train_data_points)
157 [:,1])
158     plt.xlabel('Feature1')
159     plt.ylabel('Feature2')
160     plt.title('Feature Plot of all Elements')
161     plt.show()
162
163     training = np.array(train_datapts).astype(float)
164     label_train = np.array(train_labels).astype(float)
165     weight_vector = minwvalue[1:]
166     plotDecBoundaries(training, label_train, weight_vector)
167     return p
168
169 if __name__ == '__main__':
170     and_perception = train_and_perceptron()
```

```

1  """
2  Wine Dataset (Question 2(c))
3  Name: Haolun Cheng
4  EE559 HW4
5  """
6  from __future__ import print_function
7  import copy
8  import csv
9  import random as rm
10 import numpy as np
11
12
13 class VectorOp(object):
14     @staticmethod
15     def element_multiply(x, y):
16         return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))
17
18     @staticmethod
19     def element_add(x, y):
20         return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))
21
22     @staticmethod
23     def scala_multiply(v, s):
24         return map(lambda e: e * s, v)
25
26
27 class Perceptron(object):
28     def __init__(self, input_num):
29         self.weights = [0.1] * (input_num + 1)
30         self.bias = 0.0
31
32     def __str__(self):
33         return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)
34
35     def train(self, shuffled_set, iteration):
36         JnX = []
37         weight_vectors_list = []
38         weight = np.array(self.weights)
39         wrongpts = 0
40         Jw = 0
41         ZnX = 0
42         for m in range(iteration):
43             for n in range(len(shuffled_set)):
44                 i = (m - 1) * len(shuffled_set) + n
45                 x = np.array(shuffled_set[n][:-1]).astype(float)
46                 if(float(shuffled_set[n][-1]) == 1):
47                     ZnX = 1
48                 else:
49                     ZnX = -1
50                 if(np.dot(weight, ZnX*x) <= 0):
51                     wrongpts += 1
52                     weight += ZnX * x
53                     Jw += (-1) * np.dot(weight, ZnX * x)
54                 if i >= 9500:
55                     JnX.append(Jw)
56                     weight_vectors_list.append(weight)
57             # Two halting conditions
58             if wrongpts == 0:
59                 JnX.append(Jw)

```

```
60         weight_vectors_list.append(weight)
61         print("i.1 reached")
62         break
63
64     if i > 10000:
65         print("i.2 reached")
66         break
67
68     wrongpts = 0
69
70     return np.array(JnX).astype(float),
71     np.array(weight_vectors_list).astype(float)
72
73 def calculate_error_rate(self, dataset, minwvalue):
74     dataset_copy = copy.deepcopy(dataset)
75     incorrect_count = 0
76     for line in dataset_copy:
77         input_vec = np.array(line[:-1]).astype(float)
78         if np.dot(input_vec, minwvalue) > 0:
79             line.append('1')
80         else:
81             line.append('2')
82
83     for i in dataset_copy:
84         if i[-1] != i[-2]:
85             incorrect_count += 1
86     return float(incorrect_count) / len(dataset_copy)
87
88 def f(x):
89     return 1 if x > 0 else 0
90
91
92 def read_dataset(data_path):
93     set_as_list = []
94     with open(data_path, "r") as f:
95         dataset = csv.reader(f)
96         for eachLine in dataset:
97             if eachLine[-1] != '3':
98                 if len(eachLine) != 0:
99                     eachLine.insert(0, 1)
100             set_as_list += [eachLine]
101     return set_as_list
102
103 def read_for_plot(data_path):
104     input_vecs = []
105     with open(data_path, "r") as f:
106         dataset = csv.reader(f)
107         shuffle_set = np.array(list(dataset))
108         for line in shuffle_set:
109             x, y = line[0], line[1]
110             input_vecs.append((float(x), float(y)))
111     return input_vecs
112
113
114 def train_and_perceptron():
115     p = Perceptron(13)
116     train_list = read_dataset("./wine_train.csv")
117     test_list = read_dataset("./wine_test.csv")
118
```



```
119     # shuffle data points
120     rm.shuffle(train_list)
121     shuffled_train_set = np.array(train_list)
122     rm.shuffle(test_list)
123     shuffled_test_set = np.array(test_list)
124
125     train_datapts = []
126     train_labels = []
127     for line in shuffled_train_set:
128         datap1, label1 = line[1:3], line[-1]
129         train_datapts.append(datap1)
130         train_labels.append(label1)
131
132     test_datapts = []
133     test_labels = []
134     for line in shuffled_test_set:
135         datap2, label2 = line[1:3], line[-1]
136         test_datapts.append(datap2)
137         test_labels.append(label2)
138
139     Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
140     minJvalue = min(Jvalue)
141     minPos = 0
142     for i in range(Jvalue.shape[0]):
143         if Jvalue[i] == minJvalue:
144             minPos = i
145             break
146     minwvalue = weight_vectors[minPos]
147     error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
148     error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
149     print(f"The best weight vector omega (w) is {minwvalue}")
150     print(f"The criterion function value is {minJvalue}")
151     print(f"The error rate of training set is {error_rate_train_set}")
152     print(f"The error rate of test set is {error_rate_test_set}")
153     return p
154
155
156 if __name__ == '__main__':
157     and_perception = train_and_perceptron()
```