

```

1 import random as rnd
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import numpy as np
5 from scipy import stats
6
7
8 class SimpleBayesClassifier:
9
10     def __init__(self, n_pos, n_neg):
11         """
12         Initializes the SimpleBayesClassifier with prior probabilities.
13
14         Parameters:
15         n_pos (int): The number of positive samples.
16         n_neg (int): The number of negative samples.
17
18         Returns:
19         None: This method does not return anything as it is a constructor.
20         """
21
22         self.n_pos = n_pos
23         self.n_neg = n_neg
24         self.prior_pos = n_pos / (n_pos + n_neg)
25         self.prior_neg = n_neg / (n_pos + n_neg)
26
27     def fit_params(self, x, y, n_bins=10):
28         """
29         Computes histogram-based parameters for each feature in the dataset.
30
31         Parameters:
32         x (np.ndarray): The feature matrix, where rows are samples and columns are
33         features.
34         y (np.ndarray): The target array, where each element corresponds to the
35         label of a sample.
36         n_bins (int): Number of bins to use for histogram calculation.
37
38         Returns:
39         (stay_params, leave_params): A tuple containing two lists of tuples,
40         one for 'stay' parameters and one for 'leave' parameters.
41         Each tuple in the list contains the bins and edges of the histogram for a
42         feature.
43         """
44
45         self.stay_params = []
46         self.leave_params = []
47
48         # INSERT CODE HERE
49         for i in range(x.shape[1]): # for each attributes in x_train
50             stay = x[y == 0, i]
51             stay = stay[~np.isnan(stay)]
52             Shist, Sedges = np.histogram(stay, n_bins)
53             Sedges[0] = -np.inf
54             Sedges[-1] = np.inf
55             Shist = Shist / np.sum(Shist)
56             self.stay_params.append((Shist, Sedges))
57
58             leave = x[y == 1, i]
59             leave = leave[~np.isnan(leave)]
60             Lhist, Ledges = np.histogram(leave, n_bins)

```

```

58         Ledges[0] = -np.inf
59         Ledges[-1] = np.inf
60         Lhist = Lhist / np.sum(Lhist)
61         self.leave_params.append((Lhist, Ledges))
62
63     return self.stay_params, self.leave_params
64
65     def predict(self, x, thresh=0):
66         """
67         Predicts the class labels for the given samples using the non-parametric
68         model.
69
70         Parameters:
71         x (np.ndarray): The feature matrix for which predictions are to be made.
72         thresh (float): The threshold for log probability to decide between classes.
73
74         Returns:
75         result (list): A list of predicted class labels (0 or 1) for each sample in
76         the feature matrix.
77         """
78
79         y_pred = []
80
81         # INSERT CODE HERE
82         init = np.log(self.prior_pos) - np.log(self.prior_neg)
83
84         for i in range(x.shape[0]):
85             LH = init
86             for j in range(x.shape[1]):
87                 if np.isnan(x[i][j]):
88                     continue
89                 leave_index = (
90                     np.searchsorted(self.leave_params[j][1], x[i][j], side="right")
91                     - 1
92                 )
93                 stay_index = (
94                     np.searchsorted(self.stay_params[j][1], x[i][j], side="right") -
95                     1
96                 )
97                 leave_value = self.leave_params[j][0][leave_index]
98                 if leave_value == 0:
99                     leave_value += 1e-6
100                 stay_value = self.stay_params[j][0][stay_index]
101                 if stay_value == 0:
102                     stay_value += 1e-6
103
104                 LH += np.log(leave_value) - np.log(stay_value)
105
106             if LH > thresh:
107                 y_pred.append(1)
108             else:
109                 y_pred.append(0)
110
111         return y_pred
112
113     def fit_gaussian_params(self, x, y):
114         """
115         Computes mean and standard deviation for each feature in the dataset.

```

```

114     Parameters:
115     x (np.ndarray): The feature matrix, where rows are samples and columns are
features.
116     y (np.ndarray): The target array, where each element corresponds to the
label of a sample.
117
118     Returns:
119     (gaussian_stay_params, gaussian_leave_params): A tuple containing two lists
of tuples,
120     one for 'stay' parameters and one for 'leave' parameters.
121     Each tuple in the list contains the mean and standard deviation for a
feature.
122     """
123
124     self.gaussian_stay_params = [(0, 0) for _ in range(x.shape[1])]
125     self.gaussian_leave_params = [(0, 0) for _ in range(x.shape[1])]
126
127     # INSERT CODE HERE
128     for i in range(x.shape[1]): # for each feature: calculate the parameter
129         stay = x[y == 0, i]
130         stay = stay[~np.isnan(stay)]
131         stay_mean = np.mean(stay)
132         stay_std = max(np.std(stay), 1e-6)
133         self.gaussian_stay_params[i] = (stay_mean, stay_std)
134
135         leave = x[y == 1, i]
136         leave = leave[~np.isnan(leave)]
137         leave_mean = np.mean(leave)
138         leave_std = max(np.std(leave), 1e-6)
139         self.gaussian_leave_params[i] = (leave_mean, leave_std)
140
141     return self.gaussian_stay_params, self.gaussian_leave_params
142
143     def gaussian_predict(self, x, thresh=0):
144         """
145         Predicts the class labels for the given samples using the parametric model.
146
147         Parameters:
148         x (np.ndarray): The feature matrix for which predictions are to be made.
149         thresh (float): The threshold for log probability to decide between classes.
150
151         Returns:
152         result (list): A list of predicted class labels (0 or 1) for each sample in
the feature matrix.
153         """
154
155         y_pred = []
156
157         # INSERT CODE HERE
158         for i in range(x.shape[0]):
159             predict = np.log(self.prior_pos) - np.log(self.prior_neg)
160             for j in range(x.shape[1]):
161                 if np.isnan(x[i][j]):
162                     continue
163
164                 log_leave = max(
165                     stats.norm(
166                         self.gaussian_leave_params[j][0],
167                         self.gaussian_leave_params[j][1],
168

```

```
169         1e-9,
170     )
171     log_stay = max(
172         stats.norm(
173             self.gaussian_stay_params[j][0],
174             self.gaussian_stay_params[j][1]
175         ).pdf(x[i][j]),
176         1e-9,
177     )
178     predict += np.log(log_leave) - np.log(log_stay)
179
180     if predict > thresh:
181         y_pred.append(1)
182     else:
183         y_pred.append(0)
184
185     return np.array(y_pred)
186
```