

```

1  '''
2  Machine Language Programming assignment 2
3  see the assignment description in the README.md file
4  '''
5
6  import numpy as np
7  import pandas as pd
8
9  print("Gaussian Naive Bayes Algorithm")
10 print ("Machine Language Programming assignment 2")
11 print("Author: Larry Bilodeau")
12 print()
13 print()
14
15 # define a function to process the feature labels.
16 def read_and_truncate_file(file_path):
17     feature_labels = []
18     with open(file_path, 'r') as file:
19         for line in file:
20             feature_label = line.split(':')[0]
21             feature_labels.append(feature_label)
22     return feature_labels
23
24 # Load the raw data
25 rawdata = pd.read_csv('../datasets/spambase.data', header=None)
26 print("Data set: spambase.data")
27 print("rawdata shape", rawdata.shape)
28 # load the feature labels for the rawdata
29 feature_labels = read_and_truncate_file('../datasets\\feature_labels.txt')
30
31 #
32 # step 1: split the data into training and testing sets
33 #
34 # Define the labels
35 labels = rawdata.iloc[:, -1]
36
37 # Split the data into training and testing sets
38 # Select 2300 random rows with 40% labels 1 and 60% labels 0
39 test_data = rawdata.sample(n=2300, weights=labels.map({0: 0.6, 1: 0.4}), random_state=42)
40 print("test data shape", test_data.shape)
41
42 # Define the labels for test_data
43 test_targets = test_data.iloc[:, -1]
44 test_data = test_data.iloc[:, :-1]
45
46 # Remove the selected test_data rows from rawdata to get training_data
47 training_data = rawdata.drop(test_data.index)
48 # Define the labels for training_data
49 training_targets = training_data.iloc[:, -1]
50 # trim off the targets from the training data
51 training_data = training_data.iloc[:, :-1]
52 print("training data shape", training_data.shape)
53
54
55
56 #
57 # step 2: create probabilistic models for each class
58 #
59
60 # Determine each of the hypothesis ratios for each hypothesis (H), based on the test data
61 #  $P(H|D) = P(D|H) * P(H) / P(D)$ 
62 hypotheses, frequencies = np.unique(training_targets, return_counts=True)
63 hypothesis_ratios = frequencies / len(test_targets)
64
65 # Calculate the prior probabilities  $P(H)$ 
66 #  $P(H) = P(D|H) * P(H) / P(D)$ 
67

```

```

68 for h, f, r in zip(hypotheses, frequencies, hypothesis_ratios):
69     print()
70     print(f"{'Hypothesis':<15} {'Frequency':<15} {'Ratio':<15}")
71     print(f"{h:<15} {f:<15} {r:<15.2f}")
72     print()
73     print("Prior Probability of each hypothesis")
74     print("hyposis: spam, P(h1):", hypothesis_ratios[1])
75     print("hyposis: not spam, P(h0):", hypothesis_ratios[0])
76     print(" P(+|h1):", 1 - hypothesis_ratios[1], "P(-|h1):", hypothesis_ratios[1])
77     print(" P(+|h0):", 1 - hypothesis_ratios[0], "P(-|h0):", hypothesis_ratios[0])
78
79 # Filter rows for each unique value in training_targets
80 grouped_data = training_data.groupby(training_targets)
81
82 # Calculate the mean and standard deviation for each group
83
84 for clas, group in grouped_data:
85     mean = group.mean()
86     std_dev = group.std()
87     #print(f"\nGroup for target = {clas}")
88     #for col in range(len(mean)):
89     #    print(f"Column {col:<10} Mean: {mean[col]:<15.4f} Standard Deviation:
90     #        {std_dev[col]:<15.4f}")
91
92 # probabilistic Model
93 # build the variance table from the means for each column and the column variance,
94 # where the variance is the square of the standard deviation
95 print()
96 variance_table_data = []
97 for clas, group in grouped_data:
98     mean = group.mean()
99     std_dev = group.std()
100     std_dev[std_dev == 0.0000] = 0.0001
101     variance = std_dev ** 2
102     variance[variance == 0.0000] = 0.0001
103     for i, feature_label in enumerate(feature_labels):
104         print(f"class: {clas:<10} feature: {feature_label:<30} mean: {mean[i]:<10.4f}
105             std_dev: {std_dev[i]:<10.4f} \
106                 variance: {variance[i]:<10.4f}")
107         variance_table_data.append({'Class': clas, 'Feature': feature_label, 'Mean': mean
108             [i], 'Standard Deviation': std_dev[i], \
109                 'Variance': variance[i]})
110
111 variance_table = pd.DataFrame(variance_table_data)
112 variance_table_class0 = variance_table[variance_table['Class'] == 0]
113 variance_table_class1 = variance_table[variance_table['Class'] == 1]
114
115 #print(np.shape(variance_table))
116
117 print("\nVariance Table")
118 print(f"{' ' class 0':<37}{' '
119     class 1':<25}")
120 print(f"{'Feature':<30} {'Spam_mean':<10} {'Spam_variance':<15} {'Not Spam_mean':<13} {'
121     'Not Spam_variance':<16}")
122 row_size = training_data.shape[1] - 1
123 for i in range(row_size):
124     print(f"{variance_table_class0.iloc[i, 1]:<30}", f"{variance_table_class0.iloc[i, 2]-
125         1:<10.4f}", \
126         f"{variance_table_class0.iloc[i, 4]:<15.4f}", f"{variance_table_class1.iloc[i,
127         2]-1:<13.4f}", \
128         f"{variance_table_class1.iloc[i, 4]:<16.4f}")
129
130 print()
131
132 # step 3
133 # Gaussian Naive Bayes Algorithm `
134
135 #calculatethe gaussian probability density function

```

```

128 def gaussian_pdf(x, mean, variance):
129     if variance == 0:
130         variance = 0.0001
131     return (1 / np.sqrt(2 * np.pi * variance)) * np.exp(-((x - mean) ** 2) / (2 *
132         variance))
133
134 def class_probabilites(data, variance_table_class0, variance_table_class1,
135     hypothesis_ratios):
136     # initialize the probabilities
137     probabilities = []
138     for i in range(len(data)):
139         # initialize the probabilities for each class
140         probabilities_class0 = 0
141         probabilities_class1 = 0
142         for j in range(len(data.iloc[i])):
143             # calculate the probabilities for each class
144             probabilities_class0 += np.log(gaussian_pdf(data.iloc[i, j],
145                 variance_table_class0.iloc[j, 2], variance_table_class0.iloc[j, 4]))
146             probabilities_class1 += np.log(gaussian_pdf(data.iloc[i, j],
147                 variance_table_class1.iloc[j, 2], variance_table_class1.iloc[j, 4]))
148             # calculate the probabilities for each class
149             probabilities_class0 += np.log(hypothesis_ratios[0])
150             probabilities_class1 += np.log(hypothesis_ratios[1])
151             # append the probabilities for each class
152             probabilities.append([probabilities_class0, probabilities_class1])
153     return probabilities
154
155 # calculate the class probabilities for the training data
156 probabilities = class_probabilites(training_data, variance_table_class0,
157     variance_table_class1, hypothesis_ratios)
158 print()
159 print("Training_data class probabilities")
160 print("class 0", "      class 1")
161 for i in range(10):
162     print(f"{probabilities[i][0]:10.4f} {probabilities[i][1]:10.4f}")
163 print()
164
165 # calculate the class probabilities for the test data
166 probabilities = class_probabilites(test_data, variance_table_class0,
167     variance_table_class1, hypothesis_ratios)
168 print("Test_data class probabilities")
169 [print("class 0", "      class 1")]
170 for i in range(10):
171     print(f"{probabilities[i][0]:10.4f} {' ' * 6} {probabilities[i][1]:10.4f}")
172 print()
173
174 # calculate the accuracy, precision, and recall of the model on the test_data and the
175 test_targets
176 def accuracy_precision_recall(test_targets, probabilities):
177     # initialize the accuracy, precision, and recall
178     accuracy = 0
179     precision = 0
180     recall = 0
181     for i in range(len(test_targets)):
182         # determine the predicted class
183         predicted_class = 0 if probabilities[i][0] > probabilities[i][1] else 1
184         # determine the actual class
185         actual_class = test_targets.iloc[i]
186         # increment the accuracy, precision, and recall
187         accuracy += 1 if predicted_class == actual_class else 0
188         precision += 1 if predicted_class == 1 and actual_class == 1 else 0
189         recall += 1 if actual_class == 1 else 0
190     # calculate the accuracy, precision, and recall
191     accuracy /= len(test_targets)
192     precision /= recall
193     recall /= len(test_targets)

```

```

188         return accuracy, precision, recall
189
190     print()
191     print("Accuracy    Precision    Recall")
192     accuracy, precision, recall = accuracy_precision_recall(test_targets, probabilities)
193     print(f"{accuracy:.4f}    {precision:.4f}    {recall:.4f}")
194     print()
195
196     # generate a confusion matrix for the test data given the test_targets
197     def confusion_matrix(test_targets, probabilities):
198         # initialize the confusion matrix
199         confusion_matrix = np.zeros((2, 2))
200         for i in range(len(test_targets)):
201             # determine the predicted class
202             predicted_class = 0 if probabilities[i][0] > probabilities[i][1] else 1
203             # determine the actual class
204             actual_class = test_targets.iloc[i]
205             # increment the confusion matrix
206             confusion_matrix[actual_class][predicted_class] += 1
207         return confusion_matrix
208
209     print()
210     print("Confusion Matrix")
211     print(confusion_matrix(test_targets, probabilities))
212

```