


Join GitHub today

Dismiss

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▼

[Find file](#)[Copy path](#)[ml02450](#) / [toolbox_extended.py](#) **Thomas Nilsson** Added toolbox

dbd6e67 on Dec 29, 2018

[0 contributors](#)[Raw](#) [Blame](#) [History](#)

181 lines (138 sloc) | 4.29 KB

```
1 import numpy as np
2
3 def adaboost(delta, rounds):
4     # Initial weights
5     delta = np.array(delta)
6     n = len(delta)
7     weights = np.ones(n) / n
8
9     # Run all rounds
10    for i in range(rounds):
11        eps = np.mean(delta == 1)
12        alpha = 0.5 * np.log((1 - eps) / eps)
13        s = np.array([-1 if d == 0 else 1 for d in delta])
14
15        # Calculate weight vector and normalize it
16        weights = weights.T * np.exp(s * alpha)
17        weights /= np.sum(weights)
18
19        # Print resulting weights
20        for i, w in enumerate(weights):
21            print('w[%i]: %f' % (i, w))
22
23    ### NAIVE BAYES PROB
24    def naive_bayes(y, x, *obs):
25        y = np.array(y)
26        classes = set(y)
27        X = np.array(obs).T
28        N, M = X.shape
29        C = len(classes)
30        priors = np.zeros(C)
31
32        # Class priors
33        for i, c in enumerate(classes):
34            priors[i] = sum(y == c) / N
35
36        # Probs
37        probs = np.zeros((C, M))
38        for i, c in enumerate(classes):
39            for j in range(M):
40                probs[i, j] = sum((X[:, j] == x[j]) & (y == c)) / sum(y == c)
41
42        # Joint probs
43        joint = np.prod(probs, axis=1)
44
45        # Naive bayes
46        return (joint * priors) / sum(joint * priors)
47
48
```

```

49 # ### Confusion Matrix
50 def confusion_matrix(matrix=None, tp=None, fn=None, tn=None, fp=None):
51     if matrix:
52         [tp, fn], [fp, tn] = matrix
53
54     print("TP:", tp, "FN:", fn, "TN:", tn, "FP:", fp)
55
56     n = tp + fn + tn + fp
57     accuracy = (tp + tn) / n
58     error = 1 - accuracy
59     recall = tp / (tp + fn)
60     prec = tp / (tp + fp)
61     fpr = fp / (fp + tn)
62     tpr = tp / (tp + fn)
63
64     print('Accuracy:', accuracy)
65     print('Error rate:', error)
66     print('Recall:', recall)
67     print('Precision:', prec)
68     print('FPR:', fpr)
69     print('TPR:', tpr)
70
71 ### SUPPORT
72 def supp(A):
73     A = np.array(A)
74     return sum(A.all(axis=0)) / len(A[0])
75
76 ### CONFIDENCE
77 def conf(A, B):
78     AB = np.concatenate((A, B))
79     return supp(AB) / supp(A)
80
81 ### LIFT
82 def lift(A, B): return conf(A, B) / supp(B)
83
84 ### DENSITY FOR ARD
85 def density(d):
86     return 1 / d.mean()
87
88 ### SIMILIARITY MEASURES
89 def sim(x, y):
90     f11 = sum((x == 1) & (y == 1))
91     f10 = sum((x == 1) & (y == 0))
92     f01 = sum((x == 0) & (y == 1))
93     f00 = sum((x == 0) & (y == 0))
94     return f11, f10, f01, f00
95
96
97 def SMC(x, y):
98     f11, f10, f01, f00 = sim(x, y)
99     M = len(x)
100     return (f11 + f00) / M
101
102
103 def J(x, y):
104     f11, f10, f01, f00 = sim(x, y)
105     return f11 / (f11 + f10 + f01)
106
107
108 def cos(x, y):
109     f11, f10, f01, f00 = sim(x, y)
110     return f11 / (np.linalg.norm(x) * np.linalg.norm(y))
111
112
113 def EJ(x, y):
114     a = x.T * y
115     b = np.linalg.norm(x) ** 2 + np.linalg.norm(y) ** 2 - a
116     return a / b
117
118
119 # Impurity measures
120 def gini(v): return 1 - ((v / sum(v)) ** 2).sum()
121

```

```
122 def class_error(v): return 1 - v[np.argmax(v)] / v.sum()
123
124 def kmeans3_main(data, centroids):
125     c1, c2, c3 = centroids
126     dif1, dif2, dif3 = data - c1, data - c2, data - c3
127     cat1, cat2, cat3 = [], [], []
128
129     for i in range(0, len(data)):
130         if abs(dif1[i]) <= abs(dif2[i]) and abs(dif1[i]) <= abs(dif3[i]):
131             cat1.append(data[i])
132         elif abs(dif2[i]) <= abs(dif1[i]) and abs(dif2[i]) <= abs(dif3[i]):
133             cat2.append(data[i])
134         elif abs(dif3[i]) <= abs(dif1[i]) and abs(dif3[i]) <= abs(dif2[i]):
135             cat3.append(data[i])
136         else:
137             print("ERROR")
138
139     # Print clusterings
140     print(cat1, cat2, cat3)
141
142     # Return new centroids
143     return np.array([np.mean(cat1), np.mean(cat2), np.mean(cat3)])
144
145
146 def kmeans3(data, centroids):
147     current = np.array(centroids)
148     old = np.zeros(3)
149     while np.any(current != old):
150         old = current
151         current = kmeans3_main(data, current)
152     print("terminated!\ncentroids:", current)
153
154
155 def kmeans2_main(data, centroids):
156     c1, c2 = centroids
157     dif1, dif2 = data - c1, data - c2
158     cat1, cat2 = [], []
159
160     for i in range(0, len(data)):
161         if abs(dif1[i]) <= abs(dif2[i]):
162             cat1.append(data[i])
163         elif abs(dif2[i]) <= abs(dif1[i]):
164             cat2.append(data[i])
165         else:
166             print("ERROR")
167
168     # Print clusterings
169     print(cat1, cat2)
170
171     # Return new centroids
172     return np.array([np.mean(cat1), np.mean(cat2)])
173
174
175 def kmeans2(data, centroids):
176     current = np.array(centroids)
177     old = np.zeros(2)
178     while np.any(current != old):
179         old = current
180         current = kmeans2_main(data, current)
181     print("terminated!\ncentroids:", current)
```