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

```python
import numpy as np

def adaboost(delta, rounds):
    # Initial weights
    delta = np.array(delta)
    n = len(delta)
    weights = np.ones(n) / n

    # Run all rounds
    for i in range(rounds):
        eps = np.mean(delta == 1)
        alpha = 0.5 * np.log((1 - eps) / eps)
        s = np.array([-1 if d == 0 else 1 for d in delta])

        # Calculate weight vector and normalize it
        weights = weights.T * np.exp(s * alpha)
        weights /= np.sum(weights)

        # Print resulting weights
    for i, w in enumerate(weights):
        print('w[%i]: %f' % (i, w))

### NAIVE BAYES PROB
def naive_bayes(y, x, *obs):
    y = np.array(y)
    classes = set(y)
    X = np.array(obs).T
    N, M = X.shape
    C = len(classes)
    priors = np.zeros(C)

    # Class priors
    for i, c in enumerate(classes):
        priors[i] = sum(y == c) / N

    # Probs
    probs = np.zeros((C, M))
    for i, c in enumerate(classes):
        for j in range(M):
            probs[i, j] = sum((X[:, j] == x[j]) & (y == c)) / sum(y == c)

    # Joint probs
    joint = np.prod(probs, axis=1)

    # Naive bayes
    return (joint * priors) / sum(joint * priors)


```

```python
# ### Confusion Matrix
def confusion_matrix(matrix=None, tp=None, fn=None, tn=None, fp=None):
    if matrix:
        [tp, fn], [fp, tn] = matrix

    print("TP:", tp, "FN:", fn, "TN:", tn, "FP:", fp)

    n = tp + fn + tn + fp
    accuracy = (tp + tn) / n
    error = 1 - accuracy
    recall = tp / (tp + fn)
    prec = tp / (tp + fp)
    fpr = fp / (fp + tn)
    tpr = tp / (tp + fn)

    print('Accuracy:', accuracy)
    print('Error rate:', error)
    print('Recall:', recall)
    print('Precision:', prec)
    print('FPR:', fpr)
    print('TPR:', tpr)

### SUPPORT
def supp(A):
    A = np.array(A)
    return sum(A.all(axis=0)) / len(A[0])

### CONFIDENCE
def conf(A, B):
    AB = np.concatenate((A, B))
    return supp(AB) / supp(A)

### LIFT
def lift(A, B): return conf(A, B) / supp(B)

### DENSITY FOR ARD
def density(d):
    return 1 / d.mean()

### SIMILIARITY MEASURES
def sim(x, y):
    f11 = sum((x == 1) & (y == 1))
    f10 = sum((x == 1) & (y == 0))
    f01 = sum((x == 0) & (y == 1))
    f00 = sum((x == 0) & (y == 0))
    return f11, f10, f01, f00


def SMC(x, y):
    f11, f10, f01, f00 = sim(x, y)
    M = len(x)
    return (f11 + f00) / M


def J(x, y):
    f11, f10, f01, f00 = sim(x, y)
    return f11 / (f11 + f10 + f01)


def cos(x, y):
    f11, f10, f01, f00 = sim(x, y)
    return f11 / (np.linalg.norm(x) * np.linalg.norm(y))


def EJ(x, y):
    a = x.T * y
    b = np.linalg.norm(x) ** 2 + np.linalg.norm(y) ** 2 - a
    return a / b


# Impurity measures
def gini(v): return 1 - ((v / sum(v)) ** 2).sum()

```

```python
def class_error(v): return 1 - v[np.argmax(v)] / v.sum()

def kmeans3_main(data, centroids):
    c1, c2, c3 = centroids
    dif1, dif2, dif3 = data - c1, data - c2, data - c3
    cat1, cat2, cat3 = [], [], []

    for i in range(0, len(data)):
        if abs(dif1[i]) <= abs(dif2[i]) and abs(dif1[i]) <= abs(dif3[i]):
            cat1.append(data[i])
        elif abs(dif2[i]) <= abs(dif1[i]) and abs(dif2[i]) <= abs(dif3[i]):
            cat2.append(data[i])
        elif abs(dif3[i]) <= abs(dif1[i]) and abs(dif3[i]) <= abs(dif2[i]):
            cat3.append(data[i])
        else:
            print("ERROR")

    # Print clusterings
    print(cat1, cat2, cat3)

    # Return new centroids
    return np.array([np.mean(cat1), np.mean(cat2), np.mean(cat3)])


def kmeans3(data, centroids):
    current = np.array(centroids)
    old = np.zeros(3)
    while np.any(current != old):
        old = current
        current = kmeans3_main(data, current)
    print("terminated!\ncentroids:", current)


def kmeans2_main(data, centroids):
    c1, c2 = centroids
    dif1, dif2 = data - c1, data - c2
    cat1, cat2 = [], []

    for i in range(0, len(data)):
        if abs(dif1[i]) <= abs(dif2[i]):
            cat1.append(data[i])
        elif abs(dif2[i]) <= abs(dif1[i]):
            cat2.append(data[i])
        else:
            print("ERROR")

    # Print clusterings
    print(cat1, cat2)

    # Return new centroids
    return np.array([np.mean(cat1), np.mean(cat2)])


def kmeans2(data, centroids):
    current = np.array(centroids)
    old = np.zeros(2)
    while np.any(current != old):
        old = current
        current = kmeans2_main(data, current)
    print("terminated!\ncentroids:", current)
```