Solução Lista 06

Nome: Julia Xavier
E-mail: julia.xavier@aluno.ufabc.edu.br
Nome: Leonardo Bernardes Lério
E-mail: leonardo.lerio@aluno.ufabc.edu.br

14 novembro, 2024

Exercício 01

a) Explique o que é a medida de desempenho revocação (ou recall) e calcule manualmente este valor de acordo com os valores apresentados na matriz de confusão dada acima.

RESPOSTA: A medida de desempenho revocação (ou recall) é a proporção de casos positivos corretamente identificados em relação ao total de casos positivos reais. Em outras palavras, é a capacidade do modelo em encontrar todos os casos positivos (verdadeiros positivos) dentre todos os casos que realmente são positivos (verdadeiros positivos + falsos negativos).

```
vp = 61
fn = 3

revocacao = vp / (vp+fn)
revocacao
```

```
## 0.953125
```

b) Explique o que é a medida de desempenho precisão (ou precision) e calcule manualmente este valor de acordo com os valores apresentados na matriz de confusão dada acima.

RESPOSTA: A medida de desempenho precisão (ou precision) é a proporção de casos positivos corretamente identificados em relação ao total de casos positivos previstos pelo modelo. Em outras palavras, é a capacidade do modelo de não classificar erroneamente casos negativos como positivos (falsos positivos).

```
fp = 5

precisao = vp / (vp+fp)
precisao
```

```
## 0.9242424242424242
```

c) Calcule manualmente as medidas de desempenho sensibilidade e especificidade usando a matriz de confusão acima.

RESPOSTA: Sensibilidade (ou taxa de verdadeiros positivos) é a proporção de casos positivos corretamente identificados em relação ao total de casos positivos reais. É o mesmo que a revocação calculada na pergunta a).

```
revocacao
```

```
## 0.953125
```

```
vn=136

especificidade = vn/(vn+fp)
especificidade
```

```
## 0.9645390070921985
```

d) Verifique o seu resultado com as funções recall, precision, sensitivity e specificity do pacote yardstick.

Exercício 04

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
```

```
iris = load_iris()
X = iris.data
y = iris.target
X
```

```
## array([[5.1, 3.5, 1.4, 0.2],
##        [4.9, 3. , 1.4, 0.2],
##        [4.7, 3.2, 1.3, 0.2],
##        [4.6, 3.1, 1.5, 0.2],
##        [5. , 3.6, 1.4, 0.2],
##        [5.4, 3.9, 1.7, 0.4],
##        [4.6, 3.4, 1.4, 0.3],
##        [5. , 3.4, 1.5, 0.2],
##        [4.4, 2.9, 1.4, 0.2],
##        [4.9, 3.1, 1.5, 0.1],
##        [5.4, 3.7, 1.5, 0.2],
##        [4.8, 3.4, 1.6, 0.2],
##        [4.8, 3. , 1.4, 0.1],
##        [4.3, 3. , 1.1, 0.1],
##        [5.8, 4. , 1.2, 0.2],
##        [5.7, 4.4, 1.5, 0.4],
##        [5.4, 3.9, 1.3, 0.4],
##        [5.1, 3.5, 1.4, 0.3],
##        [5.7, 3.8, 1.7, 0.3],
##        [5.1, 3.8, 1.5, 0.3],
##        [5.4, 3.4, 1.7, 0.2],
```

```
##         [5.1, 3.7, 1.5, 0.4],
##         [4.6, 3.6, 1. , 0.2],
##         [5.1, 3.3, 1.7, 0.5],
##         [4.8, 3.4, 1.9, 0.2],
##         [5. , 3. , 1.6, 0.2],
##         [5. , 3.4, 1.6, 0.4],
##         [5.2, 3.5, 1.5, 0.2],
##         [5.2, 3.4, 1.4, 0.2],
##         [4.7, 3.2, 1.6, 0.2],
##         [4.8, 3.1, 1.6, 0.2],
##         [5.4, 3.4, 1.5, 0.4],
##         [5.2, 4.1, 1.5, 0.1],
##         [5.5, 4.2, 1.4, 0.2],
##         [4.9, 3.1, 1.5, 0.2],
##         [5. , 3.2, 1.2, 0.2],
##         [5.5, 3.5, 1.3, 0.2],
##         [4.9, 3.6, 1.4, 0.1],
##         [4.4, 3. , 1.3, 0.2],
##         [5.1, 3.4, 1.5, 0.2],
##         [5. , 3.5, 1.3, 0.3],
##         [4.5, 2.3, 1.3, 0.3],
##         [4.4, 3.2, 1.3, 0.2],
##         [5. , 3.5, 1.6, 0.6],
##         [5.1, 3.8, 1.9, 0.4],
##         [4.8, 3. , 1.4, 0.3],
##         [5.1, 3.8, 1.6, 0.2],
##         [4.6, 3.2, 1.4, 0.2],
##         [5.3, 3.7, 1.5, 0.2],
##         [5. , 3.3, 1.4, 0.2],
##         [7. , 3.2, 4.7, 1.4],
##         [6.4, 3.2, 4.5, 1.5],
##         [6.9, 3.1, 4.9, 1.5],
##         [5.5, 2.3, 4. , 1.3],
##         [6.5, 2.8, 4.6, 1.5],
##         [5.7, 2.8, 4.5, 1.3],
##         [6.3, 3.3, 4.7, 1.6],
##         [4.9, 2.4, 3.3, 1. ],
##         [6.6, 2.9, 4.6, 1.3],
##         [5.2, 2.7, 3.9, 1.4],
##         [5. , 2. , 3.5, 1. ],
##         [5.9, 3. , 4.2, 1.5],
##         [6. , 2.2, 4. , 1. ],
##         [6.1, 2.9, 4.7, 1.4],
##         [5.6, 2.9, 3.6, 1.3],
##         [6.7, 3.1, 4.4, 1.4],
##         [5.6, 3. , 4.5, 1.5],
##         [5.8, 2.7, 4.1, 1. ],
##         [6.2, 2.2, 4.5, 1.5],
##         [5.6, 2.5, 3.9, 1.1],
##         [5.9, 3.2, 4.8, 1.8],
##         [6.1, 2.8, 4. , 1.3],
##         [6.3, 2.5, 4.9, 1.5],
##         [6.1, 2.8, 4.7, 1.2],
##         [6.4, 2.9, 4.3, 1.3],
```

```
##         [6.6, 3. , 4.4, 1.4],
##         [6.8, 2.8, 4.8, 1.4],
##         [6.7, 3. , 5. , 1.7],
##         [6. , 2.9, 4.5, 1.5],
##         [5.7, 2.6, 3.5, 1. ],
##         [5.5, 2.4, 3.8, 1.1],
##         [5.5, 2.4, 3.7, 1. ],
##         [5.8, 2.7, 3.9, 1.2],
##         [6. , 2.7, 5.1, 1.6],
##         [5.4, 3. , 4.5, 1.5],
##         [6. , 3.4, 4.5, 1.6],
##         [6.7, 3.1, 4.7, 1.5],
##         [6.3, 2.3, 4.4, 1.3],
##         [5.6, 3. , 4.1, 1.3],
##         [5.5, 2.5, 4. , 1.3],
##         [5.5, 2.6, 4.4, 1.2],
##         [6.1, 3. , 4.6, 1.4],
##         [5.8, 2.6, 4. , 1.2],
##         [5. , 2.3, 3.3, 1. ],
##         [5.6, 2.7, 4.2, 1.3],
##         [5.7, 3. , 4.2, 1.2],
##         [5.7, 2.9, 4.2, 1.3],
##         [6.2, 2.9, 4.3, 1.3],
##         [5.1, 2.5, 3. , 1.1],
##         [5.7, 2.8, 4.1, 1.3],
##         [6.3, 3.3, 6. , 2.5],
##         [5.8, 2.7, 5.1, 1.9],
##         [7.1, 3. , 5.9, 2.1],
##         [6.3, 2.9, 5.6, 1.8],
##         [6.5, 3. , 5.8, 2.2],
##         [7.6, 3. , 6.6, 2.1],
##         [4.9, 2.5, 4.5, 1.7],
##         [7.3, 2.9, 6.3, 1.8],
##         [6.7, 2.5, 5.8, 1.8],
##         [7.2, 3.6, 6.1, 2.5],
##         [6.5, 3.2, 5.1, 2. ],
##         [6.4, 2.7, 5.3, 1.9],
##         [6.8, 3. , 5.5, 2.1],
##         [5.7, 2.5, 5. , 2. ],
##         [5.8, 2.8, 5.1, 2.4],
##         [6.4, 3.2, 5.3, 2.3],
##         [6.5, 3. , 5.5, 1.8],
##         [7.7, 3.8, 6.7, 2.2],
##         [7.7, 2.6, 6.9, 2.3],
##         [6. , 2.2, 5. , 1.5],
##         [6.9, 3.2, 5.7, 2.3],
##         [5.6, 2.8, 4.9, 2. ],
##         [7.7, 2.8, 6.7, 2. ],
##         [6.3, 2.7, 4.9, 1.8],
##         [6.7, 3.3, 5.7, 2.1],
##         [7.2, 3.2, 6. , 1.8],
##         [6.2, 2.8, 4.8, 1.8],
##         [6.1, 3. , 4.9, 1.8],
##         [6.4, 2.8, 5.6, 2.1],
```

```
##        [7.2, 3. , 5.8, 1.6],
##        [7.4, 2.8, 6.1, 1.9],
##        [7.9, 3.8, 6.4, 2. ],
##        [6.4, 2.8, 5.6, 2.2],
##        [6.3, 2.8, 5.1, 1.5],
##        [6.1, 2.6, 5.6, 1.4],
##        [7.7, 3. , 6.1, 2.3],
##        [6.3, 3.4, 5.6, 2.4],
##        [6.4, 3.1, 5.5, 1.8],
##        [6. , 3. , 4.8, 1.8],
##        [6.9, 3.1, 5.4, 2.1],
##        [6.7, 3.1, 5.6, 2.4],
##        [6.9, 3.1, 5.1, 2.3],
##        [5.8, 2.7, 5.1, 1.9],
##        [6.8, 3.2, 5.9, 2.3],
##        [6.7, 3.3, 5.7, 2.5],
##        [6.7, 3. , 5.2, 2.3],
##        [6.3, 2.5, 5. , 1.9],
##        [6.5, 3. , 5.2, 2. ],
##        [6.2, 3.4, 5.4, 2.3],
##        [5.9, 3. , 5.1, 1.8]])
```

```
y
```

```
## array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
##        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
##        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
##        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
##        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
#Conversao setosa=1, Outras=0
y_setosa = np.where(y == 0, 1, 0)
y_setosa
```

```
## array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
##        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
##        1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
##        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Perceptron  implementação
class Perceptron:
    def __init__(self, learning_rate=0.01, n_epochs=100):
        self.learning_rate = learning_rate
        self.n_epochs = n_epochs
```

```
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_epochs):
            for xi, target in zip(X, y):
                linear_output = np.dot(xi, self.weights) + self.bias
                predicted = self.activation(linear_output)
                update = self.learning_rate * (target - predicted)
                self.weights += update * xi
                self.bias += update

    def activation(self, x):
        return 1 if x >= 0 else 0

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.array([self.activation(x) for x in linear_output])

perceptron = Perceptron(learning_rate=0.01, n_epochs=100)
```

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
acuracias = []

for train_index, test_index in kf.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y_setosa[train_index], y_setosa[test_index]

    perceptron.fit(X_train, y_train)
    y_pred = perceptron.predict(X_test)

    acuracia = np.mean(y_pred == y_test)
    acuracias.append(acuracia)
```

```
print("Acuracia média:", np.mean(acuracias))
```

```
## Acuracia média: 0.9933333333333334
```