PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Marcos Antonio Lommez Candido Ribeiro[1]

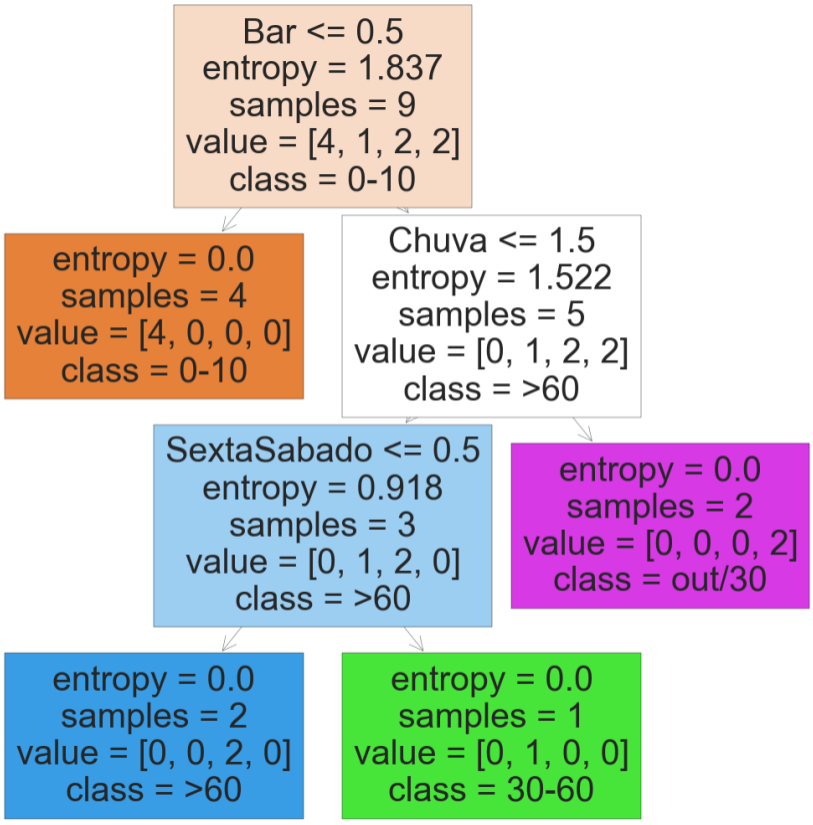
Lista #2

Inteligência Artificial

[1]Aluno de Graduação em Ciência da Computação – tonilommez@hotmail.com

Questão 1)

1) Com a codificação atual dos atributos de entrada, rode-os e plote a árvore de decisão obtida.



2) Altere a codificação dos atributos (preço, cliente e tempo de espera) para um atributo nominal não ordinal e faça os experimentos acima novamente. O que você observa? Houve ganho?

Retirei os atributos pedidos do label_encoder e os adicionei ao onehotencoder para que fossem tratados com a forma que se trata um atributo nominal nao ordinal.

Como resultado a acurácia do modelo caiu de 0,666 para 0,333, além de ter criado diversas colunas desnecessárias (no total 12).

```
label_encoder_Exemplo = LabelEncoder()
label_encoder_Alternativo = LabelEncoder()
label_encoder_Bar = LabelEncoder()
label_encoder_SexSab = LabelEncoder()
label_encoder_fome = LabelEncoder()
label_encoder_chuva = LabelEncoder()
label_encoder_Res = LabelEncoder()
```

[14] ✓ 0.1s

```
X_prev[:,0] = label_encoder_Alternativo.fit_transform(X_prev[:,0])
X_prev[:,1] = label_encoder_Bar.fit_transform(X_prev[:,1])
X_prev[:,2] = label_encoder_SexSab.fit_transform(X_prev[:,2])
X_prev[:,3] = label_encoder_fome.fit_transform(X_prev[:,3])
X_prev[:,4] = label_encoder_chuva.fit_transform(X_prev[:,4])
# X_prev[:,5] = label_encoder_Res.fit_transform(X_prev[:,5])
# X_prev[:,6] = label_encoder_Alternativo.fit_transform(X_prev[:,6])
X_prev[:,7] = label_encoder_Bar.fit_transform(X_prev[:,7])
X_prev[:,8] = label_encoder_Bar.fit_transform(X_prev[:,8])
X_prev[:,9] = label_encoder_SexSab.fit_transform(X_prev[:,9])
# X_prev[:,10] = label_encoder_SexSab.fit_transform(X_prev[:,10])

X_prev
```

[15] ✓ 0.1s

```
X_prev[:,0:11]
```

[19] ✓ 0.0s

```
array([[0, 1, 0, 0, 1, 'Alguns', 'RRR', 0, 1, 0, '0-10'],
       [9, 1, 0, 0, 1, 'Cheio', 'R', 0, 0, 3, '30-60'],
       [10, 0, 1, 0, 0, 'Alguns', 'R', 0, 0, 1, '0-10'],
       [11, 1, 0, 1, 1, 'Cheio', 'R', 1, 0, 3, 'out/30'],
       [4, 1, 0, 1, 0, 'Cheio', 'RRR', 0, 1, 0, '>60'],
       [5, 0, 1, 0, 1, 'Alguns', 'RR', 1, 1, 2, '0-10'],
       [6, 0, 1, 0, 0, 'Nenhum', 'R', 1, 0, 1, '0-10'],
       [7, 0, 0, 0, 1, 'Alguns', 'RR', 1, 1, 3, '0-10'],
       [8, 0, 1, 1, 0, 'Cheio', 'R', 1, 0, 1, '>60'],
       [1, 1, 1, 1, 1, 'Cheio', 'RRR', 0, 1, 2, 'out/30'],
       [2, 0, 0, 0, 0, 'Nenhum', 'R', 0, 0, 3, '0-10'],
       [3, 1, 1, 1, 1, 'Cheio', 'R', 0, 0, 1, '30-60']], dtype=object)
```

```
onehotencoder_restaurante = ColumnTransformer(
    transformers=[('OneHot', OneHotEncoder(), [5, 6, 10])], remainder='passthrough')
X_prev = onehotencoder_restaurante.fit_transform(X_prev)
X_prev
```

[20] ✓ 0.0s

```
array([[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0, 1, 0, 0, 1,
        0, 1, 0],
       [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 9, 1, 0, 0, 1,
        0, 0, 3],
       [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 10, 0, 1, 0, 0,
        0, 0, 1],
       [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 11, 1, 0, 1, 1,
        1, 0, 3],
       [0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 4, 1, 0, 1, 0,
        0, 1, 0],
       [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 5, 0, 1, 0, 1,
        1, 1, 2],
       [0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 6, 0, 1, 0, 0,
        1, 0, 1],
       [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 7, 0, 0, 0, 1,
        1, 1, 3],
       [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 8, 0, 1, 1, 0,
        1, 0, 1],
       [0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1, 1, 1, 1, 1,
        0, 1, 2],
       [0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2, 0, 0, 0, 0,
        0, 0, 3],
       [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 3, 1, 1, 1, 1,
        0, 0, 1]], dtype=object)
```

Questão 2)

1) Quais as diferenças entre os algoritmos de árvore ID3 e C4.5?

Existem algumas diferenças entre os algoritmos mas a principal gira em torno do problema de características que possuem muitos valores diferentes, como em casos de cpfs onde cada individuo possui um diferente. O ID3 como dito se torna muito sensível a estes dados, por isso o C4.5 modificou a forma como toma suas decisões a partir do calculo de ganho.

Mas não apenas isso como também o C4.5 permite que diversas outras situações sejam tratadas mais adequadamente como valores faltando, valores contínuos (que antes eram apenas categóricos), a poda da arvore para se evitar redundância e também a geração de regras em vez de apenas a arvore. Além disso existem outras questões como o ganho de acurácia e uma significante melhoria no tempo de execução.

2) Como o algoritmo C4.5 lida com os atributos de entrada que são numéricos?

Pela natureza do C4.5 criar regras através de uma sequencia numérica se torna possível que sejam ordenados de acordo com uma categoria (Assim incluindo os nominais ordinais) para então fazer uma regra de que a partir de determinado valor segue por um caminho e por outro valor siga por outro.

Para tratar esses valores primeiro eles são ordenados, valores duplicados são retirados e por fim se procura o ponto de corte que maximiza o ganho.