

```
1 # -*- coding: utf-8 -*-
2 """
   TCC_CaioMota_ArvoreDecisao_BRNeoDeath_2016_2018_FINAL
   .ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/
   11WavsUi1N0M-HT0FKWiYbJU92NadjRa
8
9 **Implementação de modelo utilizando algoritmo de
   Árvore de Decisão**
10 <br>
11 <br>
12 **Autor**: Caio Augusto de Souza Mota (*caiomota802@
   gmail.com*)
13
14 Data: 09/06/2021
15
16 **Revisor**: Carlos Eduardo Beluzo (*cbeluzo@gmail.
   com*)
17 <br>
18 <br>
19 *Codigo adaptado de Diogo Cortiz disponivel em: https
   ://github.com/diogocortiz/Curso-IA-para-todos/tree/
   master/ArvoreDecis%C3%A3o*
20
21 ---
22 Este código é parte do Trabalho de Conclusão de Curso
   apresentado como exigência parcial para obtenção do
   diploma do Curso de Tecnologia em Análise e
   Desenvolvimento de Sistemas do Instituto Federal de
   Educação, Ciência e Tecnologia de São Paulo Câmpus
   Campinas.
23
24 # 1. Importação de Bibliotecas, carga de dados e
   funções
25 """
26
27 # Instalando o Synapse Client
```

```
28 ! pip install synapseclient
29
30 # Util
31 import itertools
32 import synapseclient as syna
33 from getpass import getpass
34
35 # Data
36 import numpy as np
37 import pandas as pd
38
39 from sklearn.tree import DecisionTreeClassifier,
    export_graphviz
40 from sklearn import preprocessing
41 from sklearn.model_selection import train_test_split
    , cross_val_score, cross_validate, cross_val_predict
42 from sklearn.externals.six import StringIO
43 from sklearn.metrics import accuracy_score,
    classification_report, precision_score, recall_score
    , confusion_matrix, precision_recall_curve, roc_curve
    , auc, log_loss
44
45 # Images
46 import matplotlib.pyplot as plt
47 import pydotplus
48 import matplotlib.image as mpimg
49 from IPython.display import Image
50
51 """## 1.1 Carregando base de dados disponível no
    Synapse"""
52
53 # Recuperando a base de dados do repositório de dados
    Synapse
54 syn = syna.Synapse()
55 syn.login(input('Sybapse User: '), getpass('Passwd: '
    ))
56
57 # Obtendo um ponteiro e baixando os dados
58 dataset = syn.get(entity='syn22240290') # ID do
    dataset SPNeodeath
59
```

```

60 df_ori = pd.read_csv(dataset.path)
61 df_ori.shape
62
63 # Amostra para testes rápidos
64 df = df_ori.sample(frac=1)
65 df = df.dropna()
66 # Exibindo as 5 primeiras linhas
67 print(df.head(5))
68
69 count_row = df.shape[0] # Número de linhas
70 count_col = df.shape[1] # Número de colunas
71
72 print(count_row)
73 print(count_col)
74
75 # Distribuição entre registros de vivos (negativo =
    0) e mortos (positivo = 1)
76 print ('Total de registros negativos: ', df[df['
    neonatal_death'] == 0 ].shape[0])
77 print ('Total de registros positivos: ', df[df['
    neonatal_death'] == 1 ].shape[0])
78
79 """# 2. Aplicação de modelos de Machine Learning:
    Decision Tree
80
81 ---
82
83 Precisamos converter o Dataframe para um Array Numpy
    , que é o tipo de dados que iremos usar no
    treinamento.
84
85 Um com as features de entrada, e outro com os labels
    (etiquetas, rótulos do registro).
86 """
87
88 # "Features" do modelo
89 nome_features = ['tp_birth_place', 'maternal_age', '
    tp_marital_status',
90                  'tp_maternal_education_years', '
    num_live_births', 'num_fetal_losses',
91                  'tp_pregnancy_duration', '

```

```
91 tp_pregnancy', 'tp_labor',
92         'tp_prenatal_appointments', '
    cd_apgar1', 'cd_apgar5',
93         'newborn_weight', '
    has_congenital_malformation', '
    tp_maternal_skin_color',
94         'num_gestations', 'num_normal_labors'
    , 'num_cesarean_labors',
95         'num_gestational_weeks', '
    tp_presentation_newborn', 'tp_childbirth_assistance'
    ,
96         'tp_fill_form_responsible', '
    cd_robson_group']
97
98 target = ['neonatal_death']
99
100 # Array de features
101 X = df[nome_features].values
102
103 Y = df[target].values
104
105 """### Criando modelo usando divisão da base em 90
    %/10% para treino e teste."""
106
107 # "Split" do dataset para Treino e Teste do modelo
108 # Usando treino com 90% dos dados e 10% dos dados
    para teste
109 xTreino, xTeste, yTreino, yTeste = train_test_split(
    X, Y,
110     test_size=0.1,
111     shuffle=False,
112     random_state=7)
113
114 # Instanciando algortimo
115 algortimo_arvore = DecisionTreeClassifier(criterion=
    'entropy', max_depth=5)
116
117 # "Treinamento"/Construção do modelo
```

```
118 modelo_90_10 = algortimo_arvore.fit(xTreino, yTreino
119 )
120 """### Exibindo a árvore que o modelo montou
121
122 A árvore de decisão pode ser considerada um modelo
123 White Box, ou seja, um modelo que podemos entender
124 melhor o que ele aprendeu e como ele decide. Podemos
125 mostrar a árvore para isso.
126
127 """
128
129 importances = modelo_90_10.feature_importances_
130 indices = np.argsort(importances)[::-1]
131 print("Importância de cada variável para o modelo:")
132
133 for f in range(X.shape[1]):
134     print("%d. feature %d (%f)" % (f + 1, indices[f],
135 importances[indices[f]]))
136
137 f, ax = plt.subplots(figsize=(10, 5))
138 plt.title("Importância de cada variável para o
139 modelo", fontsize = 14)
140 plt.bar(range(X.shape[1]), importances[indices],
141 color="b",
142 align="center")
143 plt.xticks(range(X.shape[1]), indices)
144 plt.xlim([-1, X.shape[1]])
145 plt.ylabel("Importância", fontsize = 12)
146 plt.xlabel("Features (índice)", fontsize = 12)
147
148 plt.savefig("importance_features_tree.png", dpi=300)
149
150 plt.show()
151
152
153 # 0 - 'maternal_age'
154 # 1 - 'tp_maternal_schooling'
155 # 2 - 'tp_marital_status'
156 # 3 - 'tp_maternal_race'
157 # 4 - 'num_live_births'
158 # 5 - 'num_fetal_losses'
159 # 6 - 'num_previous_gestations'
```

```
153 # 7 - 'num_normal_labors'
154 # 8 - 'num_cesarean_labor'
155 # 9 - 'tp_pregnancy'
156 # 10 - 'newborn_weight'
157 # 11 - 'gestational_week'
158 # 12 - 'cd_apgar1',
159 # 13 - 'cd_apgar5'
160 # 14 - 'has_congenital_malformation'
161 # 15 - 'tp_newborn_presentation'
162 # 16 - 'num_prenatal_appointments'
163 # 17 - 'tp_labor'
164 # 18 - 'was_cesarean_before_labor'
165 # 19 - 'was_labor_induced'
166 # 20 - 'tp_childbirth_care'
167 # 21 - 'tp_robson_group'
168
169 # Montando imagem da árvore de decisão
170 dot_data = StringIO()
171 export_graphviz(modelo_90_10, out_file=dot_data,
172                 filled=True, feature_names=nome_features,
173                 class_names=['Vivo', 'Morto'],
174                 rounded=True, special_characters=True,
175                 node_ids=True, proportion=True,
176                 max_depth=5, rotate=True)
177
178 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
179 Image(graph.create_png())
180 graph.write_png("arvore.png")
181 Image('arvore.png',)
182
183 # Montando imagem da árvore de decisão
184 dot_data = StringIO()
185 export_graphviz(modelo_90_10, out_file=dot_data,
186                 filled=True, feature_names=nome_features,
187                 class_names=['Vivo', 'Morto'],
188                 rounded=True, special_characters=True,
189                 node_ids=True, proportion=True,
190                 max_depth=2)
191
192 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

```
186 getvalue())
187 Image(graph.create_png())
188 graph.write_png("arvore_short.png")
189 Image('arvore_short.png',)
190
191 """### Curva ROC
192
193 """
194
195 y_pred = modelo_90_10.predict(xTeste)
196 y_pred_proba = modelo_90_10.predict_proba(xTeste
   )[: , 1]
197 [fpr, tpr, thr] = roc_curve(yTeste, y_pred_proba)
198
199 print('Treino/Teste resultados para %s:' %
    modelo_90_10.__class__.__name__)
200 print("Accuracy = %2.3f" % accuracy_score(yTeste,
    y_pred))
201 print("Log_loss = %2.3f" % log_loss(yTeste,
    y_pred_proba))
202 print("AUC = %2.3f" % auc(fpr, tpr))
203
204 idx = np.min(np.where(tpr > 0.95))
205
206 #plt.figure(figsize=(13, 13))
207 plt.plot(fpr, tpr, color='coral', label='Curva ROC (
    area = %0.3f)' % auc(fpr, tpr))
208 plt.plot([0, 1], [0, 1], 'k--')
209 plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--',
    color='blue')
210 plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--',
    color='blue')
211 plt.xlim([0.0, 1.0])
212 plt.ylim([0.0, 1.05])
213 plt.xlabel('Taxa de falso positivo (1 -
    especificidade)', fontsize=14)
214 plt.ylabel('Taxa de verdadeiro positivo (recall)',
    fontsize=14)
215 plt.title('Curva ROC')
216 plt.legend(loc="lower right")
217
```

```

218 plt.savefig("ROC.png", dpi=300)
219 print("\n*Nota: Usando um limite de %.3f " % thr[idx
    ] +
220       "garante uma sensibilidade de %.3f " % tpr[idx
    ] +
221       "\ne uma especificidade de %.3f" % (1-fpr[idx
    ])) +
222       ", ou seja, uma taxa de falsos positivos de %.
    2f%%.\n" % (np.array(fpr[idx])*100))
223
224 plt.show()
225
226 """Métricas do modelo usando divisão 90/10
227
228 - PRECISÃO: DAS CLASSIFICAÇÕES QUE O MODELO FEZ PARA
    UMA DETERMINADA CLASSE
229
230 - RECALL: DOS POSSÍVEIS DATAPPOINTS PERTENCENTES A UMA
    DETERMINADA CLASSE
231 """
232
233 Y_predicoes = modelo_90_10.predict(xTeste)
234
235 print("ACURÁCIA DA ÁRVORE: ", accuracy_score(yTeste
    , Y_predicoes))
236 print(classification_report(yTeste, Y_predicoes))
237
238 """ # 3. Criando modelo usando validação cruzada"""
239
240 # k = 10 na divisão da base para treino e teste
241 algortimo_arvore = DecisionTreeClassifier(criterion=
    'entropy', max_depth=3)
242 scoring = {'accuracy': 'accuracy', 'log_loss': '
    neg_log_loss', 'auc': 'roc_auc'}
243
244 modelo2 = cross_validate(algortimo_arvore, X, Y, cv=
    10,
245                          scoring=list(scoring.values
    ()),
246                          return_train_score=False)
247

```



```
248 print('Resultado usando K-fold = 10 no cross-
validation:\n')
249 for sc in range(len(scoring)):
250     print("%s: %.3f (+/-%.3f)" % (list(scoring.keys
    ())[sc], -modelo2['test_%s' % list(scoring.values
    ())[sc]].mean()
251                                     if list(scoring.
    values())[sc]=='neg_log_loss'
252                                     else modelo2['test_%s
    ' % list(scoring.values())[sc]].mean(),
253                                     modelo2['test_%s' %
    list(scoring.values())[sc]].std()))
254
255 """Matriz de confusão
256
257 """
258
259 algortimo_arvore = DecisionTreeClassifier(criterion=
    'entropy', max_depth=3)
260 modelo2_predicts = cross_val_predict(
    algortimo_arvore, X, Y, cv=10)
261 cm_modelo2 = confusion_matrix(Y, modelo2_predicts)
262 print(cm_modelo2)
263
264 # generate report
265 print(classification_report(Y, modelo2_predicts))
```