

```

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/11WavsUi1NOM-
8     HTOFKWiYbJUU92NadjRa
9
10 **Implementação de modelo utilizando algoritmo de Árvore de Decisão**
11 <br>
12 <br>
13 **Autor**: Caio Augusto de Souza Mota (*caiomota802@gmail.com*)
14 Data: 09/06/2021
15
16 **Revisor**: Carlos Eduardo Beluzo (*cbeluzo@gmail.com*)
17 <br>
18 <br>
19 *Código adaptado de Diogo Cortiz disponível 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 # BRNeodeath
54 # Recuperando a base de dados do repositório de dados Synapse
55 syn = syna.Synapse()
56 syn.login(input('Synapse User: '), getpass('Passwd: '))
57
58 # Obtendo um ponteiro e baixando os dados
59 dataset = syn.get(entity='syn25575811') # ID do dataset BRNeodeath
60
61 df_ori = pd.read_csv(dataset.path)
62 df_ori.shape
63

```

```

64 # Amostra para testes rápidos
65 df = df_ori.sample(frac=1)
66
67 # Exibindo as 5 primeiras linhas
68 print(df.head(5))
69
70 count_row = df.shape[0] # Número de linhas
71 count_col = df.shape[1] # Número de colunas
72
73 print(count_row)
74 print(count_col)
75
76 # Distribuição entre registros de vivos (negativo = 0) e mortos (positivo = 1)
77 print ('Total de registros negativos: ', df[df['is_neonatal_death'] == 0].shape[0])
78 print ('Total de registros positivos: ', df[df['is_neonatal_death'] == 1].shape[0])
79
80 """# 2. Aplicação de modelos de Machine Learning: Decision Tree
81
82 Precisamos converter o Dataframe para um Array Numpy, que é o tipo de
dados que iremos usar no treinamento.
83
84 Um com as features de entrada, e outro com os labels (etiquetas,
rótulos do registro).
85 """
86
87 # "Features" do modelo
88 nome_features = ['maternal_age', 'tp_maternal_schooling', 'tp_marital_status',
89                   'tp_maternal_race', 'num_live_births',
90                   'num_fetal_losses',
91                   'num_previous_gestations', 'num_normal_labors',
92                   'num Cesarean_labor',
93                   'tp_pregnancy', 'newborn_weight', 'gestaional_week',
94                   'cd_apgar1', 'cd_apgar5', 'hasCongenital_malformation',
95                   'tp_newborn_presentation', 'num_prenatal_appointments',
96                   'tp_labor',
97                   'wasCesarean_before_labor', 'was_labor_induced',
98                   'tp_childbirth_care',
99                   'tp_robson_group']
100
101 target = ['is_neonatal_death']
102
103 # Array de features
104 X = df[nome_features].values
105
106 Y = df[target].values
107
108 """## Criando modelo usando divisão da base em 90%/10% para treino
e teste."""
109
110 # "Split" do dataset para Treino e Teste do modelo
111 # Usando treino com 90% dos dados e 10% dos dados para teste
112 xTreino, xTeste, yTreino, yTeste = train_test_split(X, Y,
113                                                       test_size=0.1,
114                                                       shuffle=False,
115                                                       random_state=7)
116
117 # Instanciando algoritmo
118 algortimo_arvore = DecisionTreeClassifier(criterion='entropy',
119                                             max_depth=5)
120
121 # "Treinamento"/Construção do modelo
122 modelo_90_10 = algortimo_arvore.fit(xTreino, yTreino)
123
124 """## Exibindo a árvore que o modelo montou
125
126 A árvore de decisão pode ser considerada um modelo White Box, ou seja

```

```

121 , um modelo que podemos entender melhor o que ele aprendeu e como ele
122 decide. Podemos mostrar a árvore para isso.
123 """
124 importances = modelo_90_10.feature_importances_
125 indices = np.argsort(importances)[::-1]
126 print("Importância de cada variável para o modelo:")
127
128 for f in range(X.shape[1]):
129     print("%d. feature %d (%f)" % (f + 1, indices[f], importances[
130         indices[f]]))
130 f, ax = plt.subplots(figsize=(10, 5))
131 plt.title("Importância de cada variável para o modelo", fontsize = 14
132 )
132 plt.bar(range(X.shape[1]), importances[indices],
133         color="b",
134         align="center")
135 plt.xticks(range(X.shape[1]), indices)
136 plt.xlim([-1, X.shape[1]])
137 plt.ylabel("Importância", fontsize = 12)
138 plt.xlabel("Features (índice)", fontsize = 12)
139
140 plt.savefig("importance_features_tree.png", dpi=300)
141
142 plt.show()
143
144 #Índice das features
145 # 1 - 'tp_birth_place,
146 # 2 - 'maternal_age'
147 # 3 - 'tp_marital_status'
148 # 4 - 'tp_maternal_education_years'
149 # 5 - 'num_live_births'
150 # 6 - 'num_fetal_losses'
151 # 7 - 'tp_pregnancy_duration'
152 # 8 - 'tp_pregnancy'
153 # 9 - 'tp_labor'
154 # 10 - 'tp_prenatal_appointments'
155 # 11 - 'cd_apgar1'
156 # 12 - 'cd_apgar5'
157 # 13 - 'newborn_weight'
158 # 14 - 'has_congenital_malformation'
159 # 15 - 'tp_maternal_skin_color'
160 # 16 - 'num_gestations'
161 # 17 - 'num_normal_labors'
162 # 18 - 'num Cesarean_labors'
163 # 19 - 'num_gestational_weeks'
164 # 20 - 'tp_presentation_newborn'
165 # 21 - 'tp_childbirth_assistance'
166 # 22 - 'tp_fill_form_responsible'
167 # 23 - 'cd_robson_group'
168
169 # Montando imagem da árvore de decisão
170 dot_data = StringIO()
171 export_graphviz(modelo_90_10, out_file=dot_data, filled=True,
172 feature_names=nome_features,
173 class_names=['Vivo', 'Morto'], rounded=True,
174 special_characters=True,
175 node_ids=True, proportion=True, max_depth=5, rotate=True)
176 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
177 Image(graph.create_png())
178 graph.write_png("arvore.png")
179 Image('arvore.png')
180
181 # Montando imagem da árvore de decisão
182 dot_data = StringIO()
183 export_graphviz(modelo_90_10, out_file=dot_data, filled=True,
184 feature_names=nome_features,
185 class_names=['Vivo', 'Morto'], rounded=True,
186 special_characters=True,

```

```

File - /Users/carlosbeluzo/Downloads/tcc_caiomota_arvoredecisao_brneodeath_2016_2018_final.py
184                     node_ids=True, proportion=True, max_depth=2)
185
186 graph = pydotplus.graph_from_dot_data(dot_data.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__.
200      __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)' %
208           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 )
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.
223         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
229 CLASSE
230 - RECALL: DOS POSSÍVEIS DATAPoints PERTECENTES A UMA DETERMINADA
231 CLASSE
232 """
233
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',
242                                           max_depth=3)
242 scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc':
243             '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[
251         'test_%s' % list(scoring.values())[sc]].mean()
252             if list(scoring.values())[sc]==''
253             neg_log_loss'
254             else modelo2['test_%s' % list(scoring.
255             values())[sc]].mean(),
256             values())[sc]).std()))
257 """
258 """
259 algortimo_arvore = DecisionTreeClassifier(criterion='entropy',
260 max_depth=3)
261 modelo2_predicts = cross_val_predict(algortimo_arvore, X, Y, cv=10)
262 cm_modelo2 = confusion_matrix(Y, modelo2_predicts)
263 print(cm_modelo2)
264 # generate report
265 print(classification_report(Y, modelo2_predicts))
```