```
# -*- coding: utf-8 -*-
"""TCC_CaioMota_RegLogic_BRNeoDeath_2016_2018_FINAL.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1J4dedmeN3DC5sLOvvJ0hBmaJu0xch0c3

**Implementação de modelo utilizando algoritmo de Regressão Logística**
<br>
<br>
**Autor**: Caio Augusto de Souza Mota (*caiomota802@gmail.com*)

Data: 09/06/2021

**Revisor**: Carlos Eduardo Beluzo (*cbeluzo@gmail.com*)
<br>
<br>
*Codigo adaptado de Baligh Mnassri disponivel em: https://www.kaggle.com/mnassrib/titanic-logistic-regression-with-python/notebook*

---
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.

# 1. Importação de Bibliotecas, carga de dados e funções
"""

# instalando o Synapse Client
! pip install synapseclient
```

```
29
30 import os
31 import synapseclient as syna
32 from getpass import getpass
33
34 import numpy as np
35 from math import sqrt
36 import pandas as pd
37
38 from sklearn.linear_model import LogisticRegression
39 from sklearn.model_selection import train_test_split
   , cross_val_score, cross_validate, cross_val_predict
   , GridSearchCV, StratifiedKFold
40 from sklearn.feature_selection import RFE, RFECV
41 from sklearn.metrics import roc_curve, plot_roc_curve
   , accuracy_score, auc, log_loss, confusion_matrix,
   classification_report
42
43 import matplotlib.pyplot as plt
44 plt.rc("font", size=14)
45
46 import seaborn as sns
47 sns.set(style="white") #white background style for
   seaborn plots
48 sns.set(style="whitegrid", color_codes=True)
49
50 import warnings
51 warnings.simplefilter(action='ignore')
52
53 """#### Funções auxiliares"""
54
55 # Matrix de Confusão
56 def my_cm(p_cm, p_acc):
57   plt.figure(figsize=(3,3))
58   sns.heatmap(p_cm, annot=True, fmt=".0f", linewidths
   =.9, square=True, cmap='Blues_r')
59   plt.ylabel('Valores reais')
60   plt.xlabel('Valores preditos')
61   plt.title('Acurácia: %2.3f' % p_acc, size=12)
62
63 """## 1.1 Carregando base de dados disponível no
```

```
63 Synapse"""
64
65 # BRNeodeath
66 # Recuperando a base de dados do repositório de
   dados Synapse
67 syn = syna.Synapse()
68 syn.login(input('Sybapse User: '), getpass('Passwd:'
   ))
69
70 # Obtendo um ponteiro e baixando os dados
71 dataset = syn.get(entity='syn22240290') # ID do
   dataset BRNeodeath
72
73 df_ori = pd.read_csv(dataset.path)
74
75 """## 1.2 Divisão da base em conjunto para treino e
   teste do modelo de ML"""
76
77 df = df_ori.sample(frac=1)
78 print("Shape:", df.shape, "\n")
79 df.head()
80
81 # Todas as "Features" da base de dados
82 features = ['tp_birth_place', 'maternal_age', '
   tp_marital_status',
83                 'tp_maternal_education_years', '
   num_live_births','num_fetal_losses',
84                 'tp_pregnancy_duration', '
   tp_pregnancy', 'tp_labor',
85                 'tp_prenatal_appointments', '
   cd_apgar1', 'cd_apgar5',
86                 'newborn_weight', '
   has_congenital_malformation', '
   tp_maternal_skin_color',
87                 'num_gestations', 'num_normal_labors'
   , 'num_cesarean_labors',
88                 'num_gestational_weeks', '
   tp_presentation_newborn', 'tp_childbirth_assistance'
   ,
89                 'tp_fill_form_responsible','
   cd_robson_group']
```

```
 90
 91 target = ['neonatal_death']
 92
 93 df = df.dropna()
 94
 95 # Separação das features e do "target" para serem
    usados no modelo posteriormente
 96 xArray =  df[features]
 97 yArray =  df[target]
 98
 99 # "Split" do dataset para Treino e Teste do modelo
    usando 90% para o treino e 10% para teste
100 xTreino, xTeste, yTreino, yTeste = train_test_split(
    xArray, yArray, test_size=0.1 ,random_state=42)
101
102 print(xTreino.shape)
103 print(xTeste.shape)
104 print(yTreino.shape)
105 print(yTeste.shape)
106
107 """# 2. Aplicação de modelos de Machine Learning:
    Logistic Regression
108
109 ---
110
111 ## Usando particionamento 90/10
112 """
113
114 logreg = LogisticRegression()
115
116 logreg.fit(xTreino, yTreino)
117 y_pred = logreg.predict(xTeste)
118 y_pred_proba = logreg.predict_proba(xTeste)[:, 1]
119
120 [fpr, tpr, thr] = roc_curve(yTeste, y_pred_proba)
121
122 print('Treino/Teste resultados divididos %s:' %
    logreg.__class__.__name__)
123 print("Accuracy is %2.3f" % accuracy_score(yTeste,
    y_pred))
124 print("Log_loss is %2.3f" % log_loss(yTeste,
```

```
124 y_pred_proba))
125 print("AUC is %2.3f" % auc(fpr, tpr))
126
127 """#### Curva ROC"""
128
129 idx = np.min(np.where(tpr > 0.95))
130
131 plt.figure()
132 plt.plot(fpr, tpr, color='coral', label='Curva ROC (
    area = %0.3f)' % auc(fpr, tpr))
133 plt.plot([0, 1], [0, 1], 'k--')
134 plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--',
    color='blue')
135 plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--',
    color='blue')
136 plt.xlim([0.0, 1.0])
137 plt.ylim([0.0, 1.05])
138 plt.xlabel('Taxa de falso positivo (1 -
    especificidade)', fontsize=14)
139 plt.ylabel('Taxa de verdadeiro positivo (recall)',
    fontsize=14)
140 plt.title('Curva de característica de operação do
    receptor (ROC)')
141 plt.legend(loc="lower right")
142 plt.show()
143
144 plt.savefig('ROC_90_10.png', dpi=300)
145
146 print("Usando um limite de %.3f " % thr[idx] + "
    garante uma sensibilidade de %.3f " % tpr[idx] +
147       "e uma especificidade de %.3f" % (1-fpr[idx
    ]) +
148       ", ou seja, uma taxa de falsos positivos de %.
    2f%%." % (np.array(fpr[idx])*100))
149
150 """#### Matrix de Confusão"""
151
152 acc = accuracy_score(yTeste, y_pred)
153 my_cm(confusion_matrix(yTeste, y_pred), acc)
154 plt.savefig('CM_90_10.png', dpi=300)
155
```

```
156 """#### Relatório de Classificação"""
157
158 print(classification_report(yTeste, y_pred))
159 report = classification_report(yTeste, y_pred,
    output_dict=True)
160 round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_90_10.csv')
161
162 """## Cross validation com k = 10
163
164 **No Trabalho incluir apenas o relatório de
    classificação para comparar com o resultado sem
    cross-validation, não precisar incluir matrix de
    conf.**
165 """
166
167 # Regressão logística de validação cruzada de 10
    vezes
168 logreg = LogisticRegression()
169 scoring = {'accuracy': 'accuracy', 'log_loss': '
    neg_log_loss', 'auc': 'roc_auc'}
170
171 modelo_KFold = cross_validate(logreg, xArray, yArray
    , cv=10,
172                               scoring=list(scoring.values
    ()),
173                               return_train_score=False)
174
175 print('K-fold cross-validation results:')
176 for sc in range(len(scoring)):
177     print("%s: %.3f (+/-%.3f)" % (list(scoring.keys
    ())[sc], -modelo_KFold['test_%s' % list(scoring.
    values())[sc]].mean()
178                                    if list(scoring.
    values())[sc]=='neg_log_loss'
179                                    else modelo_KFold['
    test_%s' % list(scoring.values())[sc]].mean(),
180                                    modelo_KFold['test_%s
    ' % list(scoring.values())[sc]].std()))
181
182 """#### CURVA ROC para CROSS VALIDATION"""
```

```
183
184 #https://scikit-learn.org/stable/auto_examples/
    model_selection/plot_roc_crossval.html
185 X = xArray.copy()
186 y = yArray.copy()
187
188 X.reset_index(inplace=True)
189 X.drop(columns={'index'}, inplace=True)
190
191 y.reset_index(inplace=True)
192 y.drop(columns={'index'}, inplace=True)
193
194 n_samples, n_features = X.shape
195
196 # ##############################################################
    ############################
197 # Classification and ROC analysis
198
199 # Run classifier with cross-validation and plot ROC
    curves
200 cv = StratifiedKFold(n_splits=10)
201 classifier = LogisticRegression()
202
203 tprs = []
204 aucs = []
205 mean_fpr = np.linspace(0, 1, 100)
206
207 fig, ax = plt.subplots()
208 fig.set_figwidth(10)
209 fig.set_figheight(10)
210
211 for i, (train, test) in enumerate(cv.split(X, y)):
212     classifier.fit(X[X.index.isin(train)], y[y.index
    .isin(train)])
213     viz = plot_roc_curve(classifier, X[X.index.isin(
    test)], y[y.index.isin(test)],
214                          name='ROC fold {}'.format(i
    ),
215                          alpha=0.3, lw=1, ax=ax)
216     interp_tpr = np.interp(mean_fpr, viz.fpr, viz.
    tpr)
```

```
217     interp_tpr[0] = 0.0
218     tprs.append(interp_tpr)
219     aucs.append(viz.roc_auc)
220 
221 ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color=
    'r',
222         label='Chance', alpha=.8)
223 
224 mean_tpr = np.mean(tprs, axis=0)
225 mean_tpr[-1] = 1.0
226 mean_auc = auc(mean_fpr, mean_tpr)
227 std_auc = np.std(aucs)
228 ax.plot(mean_fpr, mean_tpr, color='b',
229         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)'
     % (mean_auc, std_auc),
230         lw=2, alpha=.8)
231 
232 std_tpr = np.std(tprs, axis=0)
233 tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
234 tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
235 ax.fill_between(mean_fpr, tprs_lower, tprs_upper,
    color='grey', alpha=.2,
236                 label=r'$\pm$ 1 std. dev.')
237 
238 ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
239        title="Receiver operating characteristic
    example")
240 ax.legend(loc="lower right")
241 plt.show()
242 
243 plt.savefig('ROC_KFold_10.png', dpi=300)
244 
245 """#### Matrix de Confusão"""
246 
247 y_pred = cross_val_predict(logreg, xArray, yArray,
    cv=10)
248 acc = modelo_KFold['test_accuracy'].mean()
249 my_cm(confusion_matrix(yArray, y_pred), acc)
250 plt.savefig('CM_KFold.png', dpi=300)
251 
252 """#### Relatório de Classificação"""
```

```
253
254 print(classification_report(yArray, y_pred))
255 report = classification_report(yArray, y_pred,
    output_dict=True)
256 round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_KFold.csv')
257
258 """## RFE (Eliminação recursiva de features)
259
260 A ideia da eliminação recursiva de feature (RFE) é
    selecionar features considerando recursivamente
    conjuntos cada vez menores de features, isso ocorre
    da seguinte forma. Primeiro, o estimador é treinado
    no conjunto inicial de features e a importância de
    cada feature é obtida por meio de um atributo "coef_
    " ou por meio de um atributo "feature_importances_
    ". Em seguida, os features menos importantes são
    removidos do conjunto atual de features. Esse
    procedimento é repetido recursivamente no conjunto
    removido até que o número desejado de features a
    serem selecionados seja finalmente alcançado.
261
262 ### Criando um modelo com número de features
    otimizado usando RFECV
263
264 RFECV executa RFE em um loop de validação cruzada
    para encontrar o número ideal ou o melhor número de
    features. No codigo abaixo temos uma eliminação de
    feature recursiva aplicada na regressão logística
    com ajuste automático do número de features
    selecionados com validação cruzada.
265
266 Como resultado o codigo trouxe 6 features que é o
    numero ideia de features para o modelo e trouxe
    também as melhores features sendo elas:
267
268 - 'tp_maternal_schooling'
269 - 'gestaional_week'
270 - 'cd_apgar1'
271 - 'cd_apgar5'
272 - 'has_congenital_malformation'
```

```
- 'tp_labor'
"""

# Crie o objeto RFE e calcule uma pontuação com validação cruzada.
# A pontuação de "precisão" é proporcional ao número de classificações corretas
rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=10, scoring='accuracy')
rfecv.fit(xArray, yArray)

print("Número ideal de Features: %d" % rfecv.n_features_)
print('Features Selecionadas: %s' % list(xArray.columns[rfecv.support_]))

# Número do lote das Features VS. pontuações de validação cruzada
plt.figure(figsize=(7,5))
plt.xlabel("Número de Features selecionadas")
plt.ylabel("Pontuação de validação cruzada (nb de classificações corretas)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()

"""## Modelo Final: Usar apenas variáveis selecionadas e aplicar o GridSearchCV

No RFECV vc esta esclhendo mehlores features de sua base, no gridSearch vc esta escolhendo parametros do algorítmo.
"""

xArray_RFE = xArray[list(xArray.columns[rfecv.support_])]
xArray_RFE.head()

param_grid = {'C': np.arange(1e-05, 3, 0.1)}
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
```

```
301
302 gs = GridSearchCV(LogisticRegression(),
    return_train_score=True,
303                    param_grid=param_grid, scoring=
    scoring, cv=10, refit='Accuracy')
304
305 gs.fit(xArray_RFE, yArray)
306 results = gs.cv_results_
307
308 print('='*20)
309 print("Best params: " + str(gs.best_estimator_))
310 print("Best params: " + str(gs.best_params_))
311 print('Best score:', gs.best_score_)
312 print('='*20)
313
314 plt.figure(figsize=(10, 5))
315 plt.title("GridSearchCV evaluating using multiple
    scorers simultaneously",fontsize=16)
316
317 plt.xlabel("Inverse of regularization strength: C")
318 plt.ylabel("Score")
319 plt.grid()
320
321 ax = plt.axes()
322 ax.set_xlim(0, param_grid['C'].max())
323 ax.set_ylim(0.0, 1.2)
324
325 X_axis = np.array(results['param_C'].data, dtype=
    float)
326
327 for scorer, color in zip(list(scoring.keys()), ['g'
    , 'k', 'b']):
328     for sample, style in (('train', '--'), ('test',
    '-')):
329         sample_score_mean = -results['mean_%s_%s'
     % (sample, scorer)] if scoring[scorer]=='
    neg_log_loss' else results['mean_%s_%s' % (sample,
    scorer)]
330         sample_score_std = results['std_%s_%s' % (
    sample, scorer)]
331         ax.fill_between(X_axis, sample_score_mean -
```

```
331 sample_score_std,
332                          sample_score_mean +
    sample_score_std,
333                          alpha=0.1 if sample == 'test
    ' else 0, color=color)
334         ax.plot(X_axis, sample_score_mean, style,
    color=color,
335                 alpha=1 if sample == 'test' else 0.7
    ,
336                 label="%s (%s)" % (scorer, sample))
337
338     best_index = np.nonzero(results['rank_test_%s'
     % scorer] == 1)[0][0]
339     best_score = -results['mean_test_%s' % scorer][
    best_index] if scoring[scorer]=='neg_log_loss' else
    results['mean_test_%s' % scorer][best_index]
340
341     ax.plot([X_axis[best_index], ] * 2, [0,
    best_score],
342             linestyle='-.', color=color, marker='x'
    , markeredgewidth=3, ms=8)
343
344     ax.annotate("%0.2f" % best_score,
345                 (X_axis[best_index], best_score + 0.
    005))
346
347 plt.legend(loc="best")
348 plt.grid('off')
349 plt.show()
350
351 """#### MODELO FINAL: Usando modelo sugerido pelo
    Grid Search com variáveis selecionadas"""
352
353 logReg_GS = LogisticRegression(C=1e-05, class_weight
    =None, dual=False, fit_intercept=True,
354                    intercept_scaling=1, l1_ratio=
    None, max_iter=100,
355                    multi_class='auto', n_jobs=None,
    penalty='l2',
356                    random_state=None, solver='lbfgs'
    , tol=0.0001, verbose=0,
```

```
357                     warm_start=False)
358 
359 scoring = {'accuracy': 'accuracy', 'log_loss': '
    neg_log_loss', 'auc': 'roc_auc'}
360 
361 modelo_Final = cross_validate(logReg_GS, xArray_RFE
    , yArray, cv=10,
362                           scoring=list(scoring.values
    ()),
363                           return_train_score=False)
364 
365 print('K-fold cross-validation results:')
366 for sc in range(len(scoring)):
367     print("%s: %.3f (+/-%.3f)" % (list(scoring.keys
    ())[sc], -modelo_Final['test_%s' % list(scoring.
    values())[sc]].mean()
368                                    if list(scoring.
    values())[sc]=='neg_log_loss'
369                                    else modelo_Final['
    test_%s' % list(scoring.values())[sc]].mean(),
370                                    modelo_Final['test_%s
    ' % list(scoring.values())[sc]].std()))
371 
372 """#### Curva ROC"""
373 
374 #https://scikit-learn.org/stable/auto_examples/
    model_selection/plot_roc_crossval.html
375 X = xArray_RFE.copy()
376 y = yArray.copy()
377 
378 X.reset_index(inplace=True)
379 X.drop(columns={'index'}, inplace=True)
380 
381 y.reset_index(inplace=True)
382 y.drop(columns={'index'}, inplace=True)
383 
384 n_samples, n_features = X.shape
385 
386 # ##############################################################
    ############################
387 # Classification and ROC analysis
```

```
388
389 # Run classifier with cross-validation and plot ROC
    curves
390 cv = StratifiedKFold(n_splits=10)
391 classifier = logReg_GS
392
393 tprs = []
394 aucs = []
395 mean_fpr = np.linspace(0, 1, 100)
396
397 fig, ax = plt.subplots()
398 fig.set_figwidth(10)
399 fig.set_figheight(10)
400
401 for i, (train, test) in enumerate(cv.split(X, y)):
402     classifier.fit(X[X.index.isin(train)], y[y.index
    .isin(train)])
403     viz = plot_roc_curve(classifier, X[X.index.isin(
    test)], y[y.index.isin(test)],
404                          name='ROC fold {}'.format(i
    ),
405                          alpha=0.3, lw=1, ax=ax)
406     interp_tpr = np.interp(mean_fpr, viz.fpr, viz.
    tpr)
407     interp_tpr[0] = 0.0
408     tprs.append(interp_tpr)
409     aucs.append(viz.roc_auc)
410
411 ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color=
    'r',
412         label='Chance', alpha=.8)
413
414 mean_tpr = np.mean(tprs, axis=0)
415 mean_tpr[-1] = 1.0
416 mean_auc = auc(mean_fpr, mean_tpr)
417 std_auc = np.std(aucs)
418 ax.plot(mean_fpr, mean_tpr, color='b',
419         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)'
     % (mean_auc, std_auc),
420         lw=2, alpha=.8)
421
```

```
std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper,
    color='grey', alpha=.2,
                    label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic
    example")
ax.legend(loc="lower right")
plt.show()

plt.savefig('ROC_KFold_Final.png', dpi=300)

"""#### Matriz de confusão"""

y_pred = cross_val_predict(logReg_GS, xArray_RFE,
    yArray, cv=10)
acc = modelo_Final['test_accuracy'].mean()
my_cm(confusion_matrix(yArray, y_pred), acc)
plt.savefig('CM_Final.png', dpi=300)

"""#### Relatório de Classificação"""

print(classification_report(yArray, y_pred))
report = classification_report(yArray, y_pred,
    output_dict=True)
round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_Final.csv')
```