```
# -*- 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='syn25575811') # 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 = ['maternal_age', 'tp_maternal_schooling'
   , 'tp_marital_status',
83                 'tp_maternal_race', 'num_live_births'
   ,'num_fetal_losses',
84                 'num_previous_gestations', '
   num_normal_labors', 'num_cesarean_labor',
85                 'tp_pregnancy', 'newborn_weight', '
   gestaional_week',
86                 'cd_apgar1', 'cd_apgar5', '
   has_congenital_malformation',
87                 'tp_newborn_presentation', '
   num_prenatal_appointments', 'tp_labor',
88                 'was_cesarean_before_labor', '
   was_labor_induced', 'tp_childbirth_care',
89                 'tp_robson_group']
90
91 target = ['is_neonatal_death']
92
```

```
 93 # Separação das features e do "target" para serem
    usados no modelo posteriormente
 94 xArray =  df[features]
 95 yArray =  df[target]
 96
 97 # "Split" do dataset para Treino e Teste do modelo
    usando 90% para o treino e 10% para teste
 98 xTreino, xTeste, yTreino, yTeste = train_test_split(
    xArray, yArray, test_size=0.1 ,random_state=42)
 99
100 print(xTreino.shape)
101 print(xTeste.shape)
102 print(yTreino.shape)
103 print(yTeste.shape)
104
105 """# 2. Aplicação de modelos de Machine Learning:
    Logistic Regression
106
107 ---
108
109 ## Usando particionamento 90/10
110 """
111
112 logreg = LogisticRegression()
113
114 logreg.fit(xTreino, yTreino)
115 y_pred = logreg.predict(xTeste)
116 y_pred_proba = logreg.predict_proba(xTeste)[:, 1]
117
118 [fpr, tpr, thr] = roc_curve(yTeste, y_pred_proba)
119
120 print('Treino/Teste resultados divididos %s:' %
    logreg.__class__.__name__)
121 print("Accuracy is %2.3f" % accuracy_score(yTeste,
    y_pred))
122 print("Log_loss is %2.3f" % log_loss(yTeste,
    y_pred_proba))
123 print("AUC is %2.3f" % auc(fpr, tpr))
124
125 """#### Curva ROC"""
126
```

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

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

```
185
186 X.reset_index(inplace=True)
187 X.drop(columns={'index'}, inplace=True)
188
189 y.reset_index(inplace=True)
190 y.drop(columns={'index'}, inplace=True)
191
192 n_samples, n_features = X.shape
193
194 # ##################################################
    ############################
195 # Classification and ROC analysis
196
197 # Run classifier with cross-validation and plot ROC
    curves
198 cv = StratifiedKFold(n_splits=10)
199 classifier = LogisticRegression()
200
201 tprs = []
202 aucs = []
203 mean_fpr = np.linspace(0, 1, 100)
204
205 fig, ax = plt.subplots()
206 fig.set_figwidth(10)
207 fig.set_figheight(10)
208
209 for i, (train, test) in enumerate(cv.split(X, y)):
210     classifier.fit(X[X.index.isin(train)], y[y.index
    .isin(train)])
211     viz = plot_roc_curve(classifier, X[X.index.isin(
    test)], y[y.index.isin(test)],
212                          name='ROC fold {}'.format(i
    ),
213                          alpha=0.3, lw=1, ax=ax)
214     interp_tpr = np.interp(mean_fpr, viz.fpr, viz.
    tpr)
215     interp_tpr[0] = 0.0
216     tprs.append(interp_tpr)
217     aucs.append(viz.roc_auc)
218
219 ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color=
```

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

```
ClassRep_KFold.csv')

"""## RFE (Eliminação recursiva de features)

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.

### Criando um modelo com número de features otimizado usando RFECV

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.

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:

- 'tp_maternal_schooling'
- 'gestaional_week'
- 'cd_apgar1'
- 'cd_apgar5'
- 'has_congenital_malformation'
- 'tp_labor'
"""

# Crie o objeto RFE e calcule uma pontuação com validação cruzada.
```

```
275 # A pontuação de "precisão" é proporcional ao número
     de classificações corretas
276 rfecv = RFECV(estimator=LogisticRegression(), step=1
    , cv=10, scoring='accuracy')
277 rfecv.fit(xArray, yArray)
278
279 print("Número ideal de Features: %d" % rfecv.
    n_features_)
280 print('Features Selecionadas: %s' % list(xArray.
    columns[rfecv.support_]))
281
282 # Número do lote das Features VS. pontuações de
    validação cruzada
283 plt.figure(figsize=(7,5))
284 plt.xlabel("Número de Features selecionadas")
285 plt.ylabel("Pontuação de validação cruzada (nb de
    classificações corretas)")
286 plt.plot(range(1, len(rfecv.grid_scores_) + 1),
    rfecv.grid_scores_)
287 plt.show()
288
289 """## Modelo Final: Usar apenas variáveis
    selecionadas e aplicar o GridSearchCV
290
291 No RFECV vc esta esclhendo mehlores features de sua
    base, no gridSearch vc esta escolhendo parametros do
     algorítmo.
292 """
293
294 xArray_RFE = xArray[list(xArray.columns[rfecv.
    support_])]
295 xArray_RFE.head()
296
297 param_grid = {'C': np.arange(1e-05, 3, 0.1)}
298 scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc'
    , 'Log_loss': 'neg_log_loss'}
299
300 gs = GridSearchCV(LogisticRegression(),
    return_train_score=True,
301                   param_grid=param_grid, scoring=
    scoring, cv=10, refit='Accuracy')
```

```
302
303 gs.fit(xArray_RFE, yArray)
304 results = gs.cv_results_
305
306 print('='*20)
307 print("Best params: " + str(gs.best_estimator_))
308 print("Best params: " + str(gs.best_params_))
309 print('Best score:', gs.best_score_)
310 print('='*20)
311
312 plt.figure(figsize=(10, 5))
313 plt.title("GridSearchCV evaluating using multiple
    scorers simultaneously",fontsize=16)
314
315 plt.xlabel("Inverse of regularization strength: C")
316 plt.ylabel("Score")
317 plt.grid()
318
319 ax = plt.axes()
320 ax.set_xlim(0, param_grid['C'].max())
321 ax.set_ylim(0.0, 1.2)
322
323 X_axis = np.array(results['param_C'].data, dtype=
    float)
324
325 for scorer, color in zip(list(scoring.keys()), ['g'
    , 'k', 'b']):
326     for sample, style in (('train', '--'), ('test',
    '-')):
327         sample_score_mean = -results['mean_%s_%s'
     % (sample, scorer)] if scoring[scorer]=='
    neg_log_loss' else results['mean_%s_%s' % (sample,
    scorer)]
328         sample_score_std = results['std_%s_%s' % (
    sample, scorer)]
329         ax.fill_between(X_axis, sample_score_mean -
    sample_score_std,
330                         sample_score_mean +
    sample_score_std,
331                         alpha=0.1 if sample == 'test
    ' else 0, color=color)
```

```
332         ax.plot(X_axis, sample_score_mean, style, color=color,
333                 alpha=1 if sample == 'test' else 0.7,
334                 label="%s (%s)" % (scorer, sample))
335
336     best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
337     best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' % scorer][best_index]
338
339     ax.plot([X_axis[best_index], ] * 2, [0, best_score],
340             linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)
341
342     ax.annotate("%0.2f" % best_score,
343                 (X_axis[best_index], best_score + 0.005))
344
345 plt.legend(loc="best")
346 plt.grid('off')
347 plt.show()
348
349 """#### MODELO FINAL: Usando modelo sugerido pelo Grid Search com variáveis selecionadas"""
350
351 logReg_GS = LogisticRegression(C=1e-05, class_weight=None, dual=False, fit_intercept=True,
352                    intercept_scaling=1, l1_ratio=None, max_iter=100,
353                    multi_class='auto', n_jobs=None, penalty='l2',
354                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
355                    warm_start=False)
356
357 scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}
358
```

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

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

```
424                     label=r'$\pm$ 1 std. dev.')
425
426 ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
427        title="Receiver operating characteristic
    example")
428 ax.legend(loc="lower right")
429 plt.show()
430
431 plt.savefig('ROC_KFold_Final.png', dpi=300)
432
433 """#### Matriz de confusão"""
434
435 y_pred = cross_val_predict(logReg_GS, xArray_RFE,
    yArray, cv=10)
436 acc = modelo_Final['test_accuracy'].mean()
437 my_cm(confusion_matrix(yArray, y_pred), acc)
438 plt.savefig('CM_Final.png', dpi=300)
439
440 """#### Relatório de Classificação"""
441
442 print(classification_report(yArray, y_pred))
443 report = classification_report(yArray, y_pred,
    output_dict=True)
444 round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_Final.csv')
```