```
# -*- coding: utf-8 -*-
"""TCC_CaioMota_RegLogic_SRNeoDeath_2012_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
```

```
Synapse"""

# SPNeodeath
# Recuperando a base de dados do repositório de dados Synapse
syn = syna.Synapse()
syn.login(input('Sybapse User: '), getpass('Passwd:'))

# Obtendo um ponteiro e baixando os dados
dataset = syn.get(entity='syn22240290') # ID do dataset SPNeodeath

df_ori = pd.read_csv(dataset.path)

"""## 1.2 Divisão da base em conjunto para treino e teste do modelo de ML"""

df = df_ori.sample(frac=1)

"""# REMOVENDO OS REGISTROS NOS QUAIS PELO MENOS UM CAMPO ESTÁ EM BRANCO (NAN) """
df = df.dropna()

print("Shape:", df.shape, "\n")
df.head()

# Todas as "Features" da base de dados
features = ['tp_birth_place', 'maternal_age', 'tp_marital_status',
                'tp_maternal_education_years', 'num_live_births','num_fetal_losses',
                'tp_pregnancy_duration', 'tp_pregnancy', 'tp_labor',
                'tp_prenatal_appointments', 'cd_apgar1', 'cd_apgar5',
                'newborn_weight', 'has_congenital_malformation', 'tp_maternal_skin_color',
                'num_gestations', 'num_normal_labors', 'num_cesarean_labors',
```

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

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

```
my_cm(confusion_matrix(yTeste, y_pred), acc)
plt.savefig('CM_90_10.png', dpi=300)

"""#### Relatório de Classificação"""

print(classification_report(yTeste, y_pred))
report = classification_report(yTeste, y_pred, output_dict=True)
round(pd.DataFrame(report).transpose(),2).to_csv('ClassRep_90_10.csv')

"""## Cross validation com k = 10

**No Trabalho incluir apenas o relatório de classificação para comparar com o resultado sem cross-validation, não precisar incluir matrix de conf.**
"""

# Regressão logística de validação cruzada de 10 vezes
logreg = LogisticRegression()
scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelo_KFold = cross_validate(logreg, xArray, yArray, cv=10,
                              scoring=list(scoring.values()),
                              return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print("%s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -modelo_KFold['test_%s' % list(scoring.values())[sc]].mean()
                                  if list(scoring.values())[sc]=='neg_log_loss'
                                  else modelo_KFold['test_%s' % list(scoring.values())[sc]].mean(),
                                  modelo_KFold['test_%s
```

```
' % list(scoring.values())[sc]].std()))

"""#### CURVA ROC para CROSS VALIDATION"""

#https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html
X = xArray.copy()
y = yArray.copy()

X.reset_index(inplace=True)
X.drop(columns={'index'}, inplace=True)

y.reset_index(inplace=True)
y.drop(columns={'index'}, inplace=True)

n_samples, n_features = X.shape

# ############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = StratifiedKFold(n_splits=10)
classifier = LogisticRegression()

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
fig.set_figwidth(10)
fig.set_figheight(10)

for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X[X.index.isin(train)], y[y.index.isin(train)])
    viz = plot_roc_curve(classifier, X[X.index.isin(test)], y[y.index.isin(test)],
                         name='ROC fold {}'.format(i),
```

```
217                             alpha=0.3, lw=1, ax=ax)
218     interp_tpr = np.interp(mean_fpr, viz.fpr, viz.
    tpr)
219     interp_tpr[0] = 0.0
220     tprs.append(interp_tpr)
221     aucs.append(viz.roc_auc)
222
223 ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color=
    'r',
224         label='Chance', alpha=.8)
225
226 mean_tpr = np.mean(tprs, axis=0)
227 mean_tpr[-1] = 1.0
228 mean_auc = auc(mean_fpr, mean_tpr)
229 std_auc = np.std(aucs)
230 ax.plot(mean_fpr, mean_tpr, color='b',
231         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)'
     % (mean_auc, std_auc),
232         lw=2, alpha=.8)
233
234 std_tpr = np.std(tprs, axis=0)
235 tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
236 tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
237 ax.fill_between(mean_fpr, tprs_lower, tprs_upper,
    color='grey', alpha=.2,
238                 label=r'$\pm$ 1 std. dev.')
239
240 ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
241        title="Receiver operating characteristic
    example")
242 ax.legend(loc="lower right")
243 plt.show()
244
245 plt.savefig('ROC_KFold_10.png', dpi=300)
246
247 """#### Matrix de Confusão"""
248
249 y_pred = cross_val_predict(logreg, xArray, yArray,
    cv=10)
250 acc = modelo_KFold['test_accuracy'].mean()
251 my_cm(confusion_matrix(yArray, y_pred), acc)
```

```
252 plt.savefig('CM_KFold.png', dpi=300)
253
254 """#### Relatório de Classificação"""
255
256 print(classification_report(yArray, y_pred))
257 report = classification_report(yArray, y_pred,
    output_dict=True)
258 round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_KFold.csv')
259
260 """## RFE (Eliminação recursiva de features)
261
262 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.
263
264 ### Criando um modelo com número de features
    otimizado usando RFECV
265
266 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.
267
268 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:
269
270 - 'tp_maternal_schooling'
271 - 'gestaional_week'
```

```
- 'cd_apgar1'
- 'cd_apgar5'
- '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()

```

```
301 param_grid = {'C': np.arange(1e-05, 3, 0.1)}
302 scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc'
    , 'Log_loss': 'neg_log_loss'}
303
304 gs = GridSearchCV(LogisticRegression(),
    return_train_score=True,
305                    param_grid=param_grid, scoring=
    scoring, cv=10, refit='Accuracy')
306
307 gs.fit(xArray_RFE, yArray)
308 results = gs.cv_results_
309
310 print('='*20)
311 print("Best params: " + str(gs.best_estimator_))
312 print("Best params: " + str(gs.best_params_))
313 print('Best score:', gs.best_score_)
314 print('='*20)
315
316 plt.figure(figsize=(10, 5))
317 plt.title("GridSearchCV evaluating using multiple
    scorers simultaneously",fontsize=16)
318
319 plt.xlabel("Inverse of regularization strength: C")
320 plt.ylabel("Score")
321 plt.grid()
322
323 ax = plt.axes()
324 ax.set_xlim(0, param_grid['C'].max())
325 ax.set_ylim(0.0, 1.2)
326
327 X_axis = np.array(results['param_C'].data, dtype=
    float)
328
329 for scorer, color in zip(list(scoring.keys()), ['g'
    , 'k', 'b']):
330     for sample, style in (('train', '--'), ('test',
    '-')):
331         sample_score_mean = -results['mean_%s_%s'
     % (sample, scorer)] if scoring[scorer]=='
    neg_log_loss' else results['mean_%s_%s' % (sample,
    scorer)]
```

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

```
357 penalty='l2',
358                     random_state=None, solver='lbfgs'
    , tol=0.0001, verbose=0,
359                     warm_start=False)
360 
361 scoring = {'accuracy': 'accuracy', 'log_loss': '
    neg_log_loss', 'auc': 'roc_auc'}
362 
363 modelo_Final = cross_validate(logReg_GS, xArray_RFE
    , yArray, cv=10,
364                           scoring=list(scoring.values
    ()),
365                           return_train_score=False)
366 
367 print('K-fold cross-validation results:')
368 for sc in range(len(scoring)):
369     print("%s: %.3f (+/-%.3f)" % (list(scoring.keys
    ())[sc], -modelo_Final['test_%s' % list(scoring.
    values())[sc]].mean()
370                                    if list(scoring.
    values())[sc]=='neg_log_loss'
371                                    else modelo_Final['
    test_%s' % list(scoring.values())[sc]].mean(),
372                                    modelo_Final['test_%s
    ' % list(scoring.values())[sc]].std()))
373 
374 """#### Curva ROC"""
375 
376 #https://scikit-learn.org/stable/auto_examples/
    model_selection/plot_roc_crossval.html
377 X = xArray_RFE.copy()
378 y = yArray.copy()
379 
380 X.reset_index(inplace=True)
381 X.drop(columns={'index'}, inplace=True)
382 
383 y.reset_index(inplace=True)
384 y.drop(columns={'index'}, inplace=True)
385 
386 n_samples, n_features = X.shape
387 
```

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

```
421  % (mean_auc, std_auc),
422         lw=2, alpha=.8)
423
424 std_tpr = np.std(tprs, axis=0)
425 tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
426 tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
427 ax.fill_between(mean_fpr, tprs_lower, tprs_upper,
    color='grey', alpha=.2,
428                 label=r'$\pm$ 1 std. dev.')
429
430 ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
431        title="Receiver operating characteristic
    example")
432 ax.legend(loc="lower right")
433 plt.show()
434
435 plt.savefig('ROC_KFold_Final.png', dpi=300)
436
437 """#### Matriz de confusão"""
438
439 y_pred = cross_val_predict(logReg_GS, xArray_RFE,
    yArray, cv=10)
440 acc = modelo_Final['test_accuracy'].mean()
441 my_cm(confusion_matrix(yArray, y_pred), acc)
442 plt.savefig('CM_Final.png', dpi=300)
443
444 """#### Relatório de Classificação"""
445
446 print(classification_report(yArray, y_pred))
447 report = classification_report(yArray, y_pred,
    output_dict=True)
448 round(pd.DataFrame(report).transpose(),2).to_csv('
    ClassRep_Final.csv')
```