# Data Science com Python

## Módulo 5 - Modelagem Classificação

***Professor: Lucas Roberto Correa***

LEMBRETE: Fazer o import dos datasets usados no ambiente do colab antes de executar os comandos.

## Import dos pacotes

```
# Manipulação dados
import pandas as pd

# Visualização de dados
import seaborn as sns
import matplotlib.pyplot as plt

# Quebrar os dados
from sklearn.model_selection import train_test_split

# Feature selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler

# Modelos de classificação
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

# Validação cruzada
from sklearn.model_selection import cross_val_score

# Metricas
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, auc

# Tuning de hiperparametros
from sklearn.model_selection import GridSearchCV
```

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

## Import dos metadados

link da base: [https://www.kaggle.com/rashmiranu/banking-dataset-classification?select=new_train.csv](https://www.kaggle.com/rashmiranu/banking-dataset-classification?select=new_train.csv)

```
meta = pd.read_excel('metadata.xlsx')
```

```
meta
```

| | Feature | Feature_Type | |
|---|---|---|---|
| **0** | age | numeric | |
| **1** | job | Categorical,nominal | type of job ('admin.','blue-collar','entrepreneur','I employed','services','stude |
| **2** | marital | categorical,nominal | marital status ('divorced','married','single','unknown'; note: |
| **3** | education | categorical,nominal | ('basic.4y','basic.6y','basic.9y','high.school','illiterate','professiona |
| **4** | default | categorical,nominal | ha |
| **5** | housing | categorical,nominal | I |
| **6** | loan | categorical,nominal | h |
| **7** | contact | categorical,nominal | contact co |
| **8** | month | categorical,ordinal | last contact month |
| **9** | dayofweek | categorical,ordinal | last contact da |
| **10** | duration | numeric | last contact duration, in seconds . Important note: this attribute |
| **11** | campaign | numeric | number of contacts performed during this campaign a |
| **12** | pdays | numeric | number of days that passed by after the client was last co mea |
| **13** | previous | numeric | number of contacts performed |
| **14** | poutcome | categorical,nominal | outcome of the previous marketing ca |

## Import da base

```
df = pd.read_csv('new_train.csv', sep=',')
```

```
df.head()
```

| | age | job | marital | education | default | housing | loan | contact | month |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 49 | blue-collar | married | basic.9y | unknown | no | no | cellular | nov |
| **1** | 37 | entrepreneur | married | university.degree | no | no | no | telephone | nov |
| **2** | 78 | retired | married | basic.4y | no | no | no | cellular | jul |
| **3** | 36 | admin. | married | university.degree | no | yes | no | telephone | may |
| **4** | 59 | retired | divorced | university.degree | no | no | no | cellular | jun |

## Feature engineering - Criando novas variáveis

```
df.corr()
```

| | age | duration | campaign | pdays | previous |
|---|---|---|---|---|---|
| **age** | 1.000000 | -0.001841 | 0.003302 | -0.032011 | 0.020670 |
| **duration** | -0.001841 | 1.000000 | -0.075663 | -0.047127 | 0.022538 |
| **campaign** | 0.003302 | -0.075663 | 1.000000 | 0.053795 | -0.079051 |
| **pdays** | -0.032011 | -0.047127 | 0.053795 | 1.000000 | -0.589601 |
| **previous** | 0.020670 | 0.022538 | -0.079051 | -0.589601 | 1.000000 |

```
df['poutcome'].value_counts()
```

```
nonexistent    28416
failure         3429
success         1105
Name: poutcome, dtype: int64
```

```
df['previous'].value_counts()
```

```
0    28416
1     3673
2      606
3      175
4       60
5       14
6        5
```

```
7        1
Name: previous, dtype: int64
```

Criando uma nova variável que traz a escala de dificuldade de contato, baseando-se em poutcome e previous

```
df['difficulty'] = -1 # para desconhecido
df.loc[(df['poutcome'] == 'success') & (df['previous'].between(0,1)), 'difficulty'] = 0 #
df.loc[(df['poutcome'] == 'success') & (df['previous'].between(2,4)), 'difficulty'] = 1 #
df.loc[(df['poutcome'] == 'success') & (df['previous'].between(5,7)), 'difficulty'] = 2 #
df.loc[(df['poutcome'] == 'nonexistent') & (df['previous'] > 7), 'difficulty'] = 3 # para
df.loc[(df['poutcome'] == 'failure'), 'difficulty'] = 4 # para impossivel
```

```
df['difficulty'].value_counts()
```

```
-1    28416
 4     3429
 0      697
 1      391
 2       17
Name: difficulty, dtype: int64
```

## ABT

```
df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y', 'difficulty'],
      dtype='object')
```

```
df.isnull().sum()
```

```
age            0
job            0
marital        0
education      0
default        0
housing        0
loan           0
contact        0
month          0
day_of_week    0
duration       0
campaign       0
pdays          0
previous       0
poutcome       0
y              0
difficulty     0
dtype: int64
```

```
df.columns
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'y', 'difficulty'],
      dtype='object')
```

```
df.dtypes
```

```
age             int64
job            object
marital        object
education      object
default        object
housing        object
loan           object
contact        object
month          object
day_of_week    object
duration        int64
campaign        int64
pdays           int64
previous        int64
poutcome       object
y              object
difficulty      int64
dtype: object
```

Separando as variáveis explicativas da variável resposta

```
explicativas = df.drop(columns=['y'])
```

```
var_resp = df['y']
```

```
var_resp = var_resp.replace('no', 0)
var_resp = var_resp.replace('yes', 1)
```

```
var_resp.head()
```

```
0    0
1    0
2    1
3    0
4    0
Name: y, dtype: int64
```

```
var_resp.value_counts()
```

```
0    29238
1     3712
Name: y, dtype: int64
```

```
type(var_resp)
```

```
pandas.core.series.Series
```

```
explicativas.head()
```

| | age | job | marital | education | default | housing | loan | contact | month |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 49 | blue-collar | married | basic.9y | unknown | no | no | cellular | nov |
| **1** | 37 | entrepreneur | married | university.degree | no | no | no | telephone | nov |
| **2** | 78 | retired | married | basic.4y | no | no | no | cellular | jul |
| **3** | 36 | admin. | married | university.degree | no | yes | no | telephone | may |
| **4** | 59 | retired | divorced | university.degree | no | no | no | cellular | jun |

## Tratamento de enconding das variáveis categóricas

```
expl_cat = explicativas[['job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month','day_of_week','poutcome','difficulty']]
```

```
expl_num = explicativas[['age','duration', 'campaign', 'pdays', 'previous']]
```

```
expl_cat_encoding = pd.get_dummies(expl_cat, prefix_sep='_', columns=expl_cat.columns,
                                    drop_first=True)
```

```
expl_cat_encoding.head()
```

| | job_blue-collar | job_entrepreneur | job_housemaid | job_management | job_retired | job_self employe |
|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | |
| **1** | 0 | 1 | 0 | 0 | 0 | |
| **2** | 0 | 0 | 0 | 0 | 1 | |
| **3** | 0 | 0 | 0 | 0 | 0 | |
| **4** | 0 | 0 | 0 | 0 | 1 | |

```
explicativas_tratada = expl_num.merge(expl_cat_encoding, left_index=True, right_index=True
```

```
explicativas_tratada.shape
```

```
(32950, 52)
```

Resultado a ser considerado na modelagem

```
explicativas_tratada.head()
```

| | age | duration | campaign | pdays | previous | job_blue-collar | job_entrepreneur | job_housem |
|---|---|---|---|---|---|---|---|---|
| **0** | 49 | 227 | 4 | 999 | 0 | 1 | 0 | |
| **1** | 37 | 202 | 2 | 999 | 1 | 0 | 1 | |
| **2** | 78 | 1148 | 1 | 999 | 0 | 0 | 0 | |
| **3** | 36 | 120 | 2 | 999 | 0 | 0 | 0 | |
| **4** | 59 | 368 | 2 | 999 | 0 | 0 | 0 | |

## Feature selection

Seleção de variáveis usando teste de chi2 para vars categóricas

```
expl_cat_encoding_norm = MinMaxScaler().fit_transform(expl_cat_encoding)

chi_selector = SelectKBest(chi2, k=5)
chi_selector.fit(expl_cat_encoding_norm, var_resp)
chi_support = chi_selector.get_support()
chi_feature = expl_cat_encoding.loc[:, chi_support].columns.tolist()
chi_feature
```

```
['month_mar', 'month_oct', 'poutcome_success', 'difficulty_0', 'difficulty_1']
```

Seleção de variáveis usando Regressão Logistica para vars numéricas

```
rfe_selector = RFE(estimator=LogisticRegression(max_iter=200),
                   n_features_to_select=3,
                   step=1)

rfe_selector.fit(expl_num, var_resp)
rfe_support = rfe_selector.get_support()
rfe_feature = expl_num.loc[:, rfe_support].columns.tolist()
rfe_feature
```

```
['age', 'campaign', 'previous']
```

Base a ser considerada pós feature selection

```
expl_num_feature = expl_num[['age', 'campaign', 'previous']]
```

Note que poderiamos trazer somente a classe selecionada da variável, no entanto, isso é extremamente dificil em produção e ainda traz desvantagens caso uma nova categoria seja adicionada.

```
expl_cat_feature = expl_cat[['job',
 'contact',
 'month',
 'poutcome',
 'difficulty']]
```

```
expl_cat_feature = pd.get_dummies(expl_cat_feature, prefix_sep='_', columns=expl_cat_featu
                                  drop_first=True)
```

```
explicativas_modelagem = expl_num_feature.merge(expl_cat_feature, left_index=True, right_i
```

```
explicativas_modelagem.head()
```

| | age | campaign | previous | job_blue-collar | job_entrepreneur | job_housemaid | job_manageme |
|---|---|---|---|---|---|---|---|
| **0** | 49 | 4 | 0 | 1 | 0 | 0 | |
| **1** | 37 | 2 | 1 | 0 | 1 | 0 | |
| **2** | 78 | 1 | 0 | 0 | 0 | 0 | |
| **3** | 36 | 2 | 0 | 0 | 0 | 0 | |
| **4** | 59 | 2 | 0 | 0 | 0 | 0 | |

## Quebra do dataset entre treino e teste

```
x_treino, x_teste, y_treino, y_teste = train_test_split(explicativas_modelagem,
                                                        var_resp,
                                                        test_size=0.3,
                                                        random_state=42)
```

## O algoritmo

```
tree = DecisionTreeClassifier(random_state=42)
```

```
tree
```

```
DecisionTreeClassifier(random_state=42)
```

```
rf = RandomForestClassifier(n_estimators=400,
                            random_state=42)
```

```
rf
```

```
RandomForestClassifier(n_estimators=400, random_state=42)
```

## cross validation

```
tree_cross = cross_val_score(estimator=tree,
                             X=x_treino,
                             y=y_treino,
                             cv=3,
                             scoring=make_scorer(accuracy_score))
```

```
tree_cross
```

```
array([0.86539212, 0.86147242, 0.8673257 ])
```

```
resultado = cross_val_score(estimator=rf,
                            X=x_treino,
                            y=y_treino,
                            cv=3,
                            scoring=make_scorer(accuracy_score))
```

```
resultado
```

```
array([0.87904799, 0.87656087, 0.88098335])
```

```
tree.fit(x_treino, y_treino)
```

```
DecisionTreeClassifier(random_state=42)
```

```
accuracy_score(y_treino, tree.predict(x_treino))
```

```
0.9526121829611967
```

```
rf.fit(x_treino, y_treino)
```

```
RandomForestClassifier(n_estimators=400, random_state=42)
```

```
accuracy_score(y_treino, rf.predict(x_treino))
```

```
0.9526121829611967
```

## Analisando overfitting

```
accuracy_score(y_teste, rf.predict(x_teste))
```

```
0.876074860900354
```

## Tuning de hiperparametros

```
# dicionario da random Forest

rf_grid_dc = {
    'n_estimators':[50,100,200],
    'criterion':['gini','entropy'],
    'bootstrap':[True,False],
    'random_state':[42]
}

rf_grid_dc
```

```
{'bootstrap': [True, False],
 'criterion': ['gini', 'entropy'],
 'n_estimators': [50, 100, 200],
 'random_state': [42]}
```

```
rf_grid = GridSearchCV(rf,
                      rf_grid_dc,
                      cv=2)
rf_grid
```

```
GridSearchCV(cv=2,
             estimator=RandomForestClassifier(n_estimators=400,
                                              random_state=42),
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'n_estimators': [50, 100, 200], 'random_state': [42]})
```

```
rf_grid.fit(x_treino, y_treino)
```

```
GridSearchCV(cv=2,
             estimator=RandomForestClassifier(n_estimators=400,
                                              random_state=42),
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'n_estimators': [50, 100, 200], 'random_state': [42]})
```

```
rf_grid.best_params_
```

```
{'bootstrap': True,
 'criterion': 'gini',
 'n_estimators': 200,
 'random_state': 42}
```

```
rf_grid.best_score_
```

```
0.8806850166102556
```

Validando a performance em teste

```
accuracy_score(y_teste, rf_grid.predict(x_teste))
```

```
0.8750632271117855
```

Fazendo o treinamento com tuning de hiperparametros para outro modelo

```
gb = GradientBoostingClassifier(n_estimators=200,
                           random_state=42)
```

```
#dicionario do GB

gb_grid_dc = {
    'min_samples_leaf': [1,5,10],
    'min_samples_split': [1.0,2],
    'max_depth':[1,3,8],
    'n_estimators':[10,20,50,200]
}
```

```
gb_grid = GridSearchCV(gb,
                      gb_grid_dc,
                      cv=2)
gb_grid
```

```
GridSearchCV(cv=2,
             estimator=GradientBoostingClassifier(n_estimators=200,
                                                  random_state=42),
             param_grid={'max_depth': [1, 3, 8], 'min_samples_leaf': [1, 5, 10],
                         'min_samples_split': [1.0, 2],
                         'n_estimators': [10, 20, 50, 200]})
```

```
gb_grid.fit(x_treino, y_treino)
```

```
GridSearchCV(cv=2,
             estimator=GradientBoostingClassifier(n_estimators=200,
                                                  random_state=42),
             param_grid={'max_depth': [1, 3, 8], 'min_samples_leaf': [1, 5, 10],
                         'min_samples_split': [1.0, 2],
                         'n_estimators': [10, 20, 50, 200]})
```

```
gb_grid.best_params_
```

```
{'max_depth': 3,
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'n_estimators': 20}
```

Podemos especificar no Tuning qual a métrica que se quer otimizar

```
gb_grid_roc = GridSearchCV(gb,
                      gb_grid_dc,
                      cv=2,
                          scoring='roc_auc')
```

```
gb_grid_roc.fit(x_treino, y_treino)
```

```
GridSearchCV(cv=2,
             estimator=GradientBoostingClassifier(n_estimators=200,
                                                  random_state=42),
             param_grid={'max_depth': [1, 3, 8], 'min_samples_leaf': [1, 5, 10],
                         'min_samples_split': [1.0, 2],
                         'n_estimators': [10, 20, 50, 200]},
             scoring='roc_auc')
```

```
gb_grid_roc.best_score_
```

```
0.7635106419553951
```

## Métricas

```
from sklearn.metrics import plot_confusion_matrix
```

```
plot_confusion_matrix(gb_gri, X_test, y_test)
```

```
precision_score(y_teste, rf_grid.predict(x_teste))
```

```
0.4428794992175274
```

```
recall_score(y_teste, rf_grid.predict(x_teste))
```

```
0.24354561101549052
```

```
probs = gb_grid_roc.predict_proba(x_teste)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(y_teste, preds)
roc_auc = auc(fpr, tpr)
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```