

BETA BANK

ESTE PROJETO SERÁ DESENVOLVIDO PARA PREVER O POSSÍVEL ÊXODO DE CORRENTISTAS. A INSTITUIÇÃO SOFRE COM A SAÍDA DOS CORRENTISTAS, E DESEJA SABER QUAIS CORRENTISTAS TEM O RISCO MAIOR DE MIGRAR PARA OUTRA ORGANIZAÇÃO.

DESCRIÇÃO DO PROJETO

Os clientes do Beta Bank estão saindo: pouco a pouco, escapulindo todo mês. Os banqueiros descobriram que é mais barato salvar os clientes existentes do que atrair novos.

Precisamos prever se um cliente deixará o banco em breve. Você tem os dados sobre o comportamento passado dos clientes e rescisões de contratos com o banco.

Construa um modelo com o valor máximo possível de F1. Para passar na revisão, você precisa de um valor F1 de pelo menos 0,59 para o conjunto de dados de teste.

Além disso, meça a métrica AUC-ROC e compare-a com o valor F1.

BIBLIOTECAS

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, precision_recall_curve
from sklearn.utils import shuffle
```

LEITURA E AVALIAÇÃO DOS DADOS

In [2]:

```
df = pd.read_csv('/datasets/Churn.csv')
df.head()
```

Out[2]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2.0
1	2	15647311	Hill	608	Spain	Female	41	1.0
2	3	15619304	Onio	502	France	Female	42	8.0
3	4	15701354	Boni	699	France	Female	39	1.0
4	5	15737888	Mitchell	850	Spain	Female	43	2.0

◀ ▶

In [3]:

```
#Informações dos dados
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   RowNumber        10000 non-null   int64  
 1   CustomerId      10000 non-null   int64  
 2   Surname          10000 non-null   object  
 3   CreditScore      10000 non-null   int64  
 4   Geography         10000 non-null   object  
 5   Gender            10000 non-null   object  
 6   Age               10000 non-null   int64  
 7   Tenure            9091 non-null   float64 
 8   Balance           10000 non-null   float64 
 9   NumOfProducts     10000 non-null   int64  
 10  HasCrCard        10000 non-null   int64  
 11  IsActiveMember    10000 non-null   int64  
 12  EstimatedSalary   10000 non-null   float64 
 13  Exited            10000 non-null   int64  
dtypes: float64(3), int64(8), object(3)
memory usage: 1.1+ MB
```

DESCRIÇÃO DAS COLUNAS

RowNumber — índice das strings de dados

CustomerId — identificador exclusivo do cliente

Surname — sobrenome

CreditScore — pontuação de crédito

Geography — país de residência

Gender — gênero

Age — idade

Tenure — tempo de serviço para o cliente

Balance — saldo da conta

NumOfProducts — número de produtos bancários usados pelo cliente

HasCrCard — cliente possui cartão de crédito (1 - sim; 0 - não)

IsActiveMember — cliente ativo (1 - sim; 0 - não)

EstimatedSalary — salário estimado

Exited — o cliente saiu (1 - sim; 0 - não)\

CONCLUSÃO

TEMOS ALGUMAS COLUNAS CATEGÓRICAS:

Geography

Gender

VALORES AUSENTES:

Tenure

VALORES AUSENTES

PARA SUBSTITUIR OS VALORES AUSENTES NA COLUNA, IREMOS UTILIZAR A MEDIANA, POIS ESTE MÉTODO SERÁ A MELHOR REPRESENTAÇÃO DA MÉDIA POR CAUSA DA DISCREPÂNCIA DOS VALORES.

In [4]:

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

Out[4]:

```
1.0    952
2.0    950
8.0    933
3.0    928
5.0    927
7.0    925
4.0    885
9.0    882
6.0    881
10.0   446
0.0    382
Name: Tenure, dtype: int64
```

In [5]:

```
df['Tenure']= df['Tenure'].fillna(df['Tenure'].median())
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   RowNumber        10000 non-null   int64  
 1   CustomerId      10000 non-null   int64  
 2   Surname          10000 non-null   object  
 3   CreditScore      10000 non-null   int64  
 4   Geography         10000 non-null   object  
 5   Gender            10000 non-null   object  
 6   Age               10000 non-null   int64  
 7   Tenure            10000 non-null   float64 
 8   Balance           10000 non-null   float64 
 9   NumOfProducts     10000 non-null   int64  
 10  HasCrCard        10000 non-null   int64  
 11  IsActiveMember    10000 non-null   int64  
 12  EstimatedSalary   10000 non-null   float64 
 13  Exited            10000 non-null   int64  
dtypes: float64(3), int64(8), object(3)
memory usage: 1.1+ MB
```

VALORES DUPLICADOS

In [7]:

```
df.duplicated().sum()
```

Out[7]:

0

DIVISÃO DOS DADOS

PRIMEIRAMENTE, SERÁ REALIZADA A PREPARAÇÃO DOS DADOS PARA MODELAGEM DE APRENDIZADO DE MÁQUINA. OS DADOS SERÃO DIVIDOS EM CONJUNTOS DE TREINAMENTO, VALIDAÇÃO E TESTE

In [8]:

```
df_ohe= pd.get_dummies(df, drop_first=True)
target= df_ohe['Exited']
features= df_ohe.drop(['Exited'], axis=1)

features_train, features_valid_test, target_train, target_valid_test= train_test_split(
    features, target, test_size=0.4, random_state=42)

features_valid, features_test, target_valid, target_test = train_test_split(
    features_valid_test, target_valid_test, test_size=0.5, random_state=42)

print(features_train.shape)
print(features_valid.shape)
print(features_test.shape)
```

```
(6000, 2944)
(2000, 2944)
(2000, 2944)
```

REGRESSÃO LOGÍSTICA

MODELO DE AJUSTE

In [9]:

```
solver_list= ['lbfgs','liblinear']
logistic_regression_columns= ['solver','acc','f1_score']
logistic_regression_list= []

for solver_id in solver_list:
    model= LogisticRegression(random_state=42, solver= solver_id)
    model.fit(features_train, target_train)
    predicted_valid= model.predict(features_valid)
    logistic_regression_list.append(
        [solver_id, accuracy_score(target_valid, predicted_valid),
         f1_score(target_valid,predicted_valid)])

logistic_regression= pd.DataFrame(logistic_regression_list, columns= logistic_regression_
logistic_regression
```

Out[9]:

	solver	acc	f1_score
0	lbfgs	0.81	0.0
1	liblinear	0.81	0.0

AJUSTE DE PESO DE CLASSE

In [10]:

```
logistic_regression_columns= ['solver', 'class_weight', 'f1_score']
logistic_regression_list= []

for solver_id in solver_list:
    for class_weight in ['balanced', None]:
        model= LogisticRegression(random_state=42, solver= solver_id, class_weight= class_weight)
        model.fit(features_train, target_train)
        predicted_valid= model.predict(features_valid)
        logistic_regression_list.append([solver_id, class_weight, f1_score(target_valid, predicted_valid)])

logistic_regression= pd.DataFrame(logistic_regression_list, columns= logistic_regression_columns)
```

Out[10]:

	solver	class_weight	f1_score
0	lbfgs	balanced	0.331135
1	lbfgs	None	0.000000
2	liblinear	balanced	0.406446
3	liblinear	None	0.000000

AMOSTRAGEM

In [11]:

```
features_zeros = features_train[target_train == 0]
features_ones = features_train[target_train == 1]
target_zeros = target_train[target_train == 0]
target_ones = target_train[target_train == 1]

print(features_zeros.shape)
print(features_ones.shape)
print(target_zeros.shape)
print(target_ones.shape)
```

```
(4773, 2944)
(1227, 2944)
(4773,)
(1227,)
```

In [12]:

```
logistic_regression_columns = ['solver', 'repeat', 'f1_score']
logistic_regression_list = []

for solver_id in solver_list:
    for repeat in range(1, 5):
        features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
        target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)
        features_upsampled, target_upsampled = shuffle(features_upsampled,
                                                       target_upsampled,
                                                       random_state=42)
        model = LogisticRegression(random_state=42, solver=solver_id)
        model.fit(features_upsampled, target_upsampled)
        predicted_valid = model.predict(features_valid)
        logistic_regression_list.append([solver_id,
                                         repeat,
                                         f1_score(target_valid, predicted_valid)])
]

logistic_regression = pd.DataFrame(logistic_regression_list, columns=logistic_regression_
logistic_regression
```

Out[12]:

	solver	repeat	f1_score
0	lbfgs	1	0.000000
1	lbfgs	2	0.005222
2	lbfgs	3	0.218054
3	lbfgs	4	0.332689
4	liblinear	1	0.000000
5	liblinear	2	0.005222
6	liblinear	3	0.218054
7	liblinear	4	0.328339

AMOSTRAGEM REDUZIDA

In [13]:

```
logistic_regression_columns = ['solver', 'fraction', 'f1_score']
logistic_regression_list = []

for solver_id in solver_list:
    for fraction in np.arange(0.1, 1, 0.05):
        features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42),
                                           target_zeros.sample(frac=fraction, random_state=42)])
        features_downsampled, target_downsampled = shuffle(features_downsampled,
                                                          target_downsampled,
                                                          random_state=42)
        model = LogisticRegression(random_state=42, solver=solver_id)
        model.fit(features_downsampled, target_downsampled)
        predicted_valid = model.predict(features_valid)
        logistic_regression_list.append([solver_id,
                                         fraction,
                                         f1_score(target_valid, predicted_valid)
                                         ])

logistic_regression = pd.DataFrame(logistic_regression_list, columns=logistic_regression_columns)
```

Out[13]:

	solver	fraction	f1_score
0	lbfgs	0.10	0.319328
1	lbfgs	0.15	0.319328
2	lbfgs	0.20	0.328684
3	lbfgs	0.25	0.332046
4	lbfgs	0.30	0.302676
5	lbfgs	0.35	0.153623
6	lbfgs	0.40	0.079332
7	lbfgs	0.45	0.030227
8	lbfgs	0.50	0.005249
9	lbfgs	0.55	0.000000
10	lbfgs	0.60	0.000000
11	lbfgs	0.65	0.000000
12	lbfgs	0.70	0.000000
13	lbfgs	0.75	0.000000
14	lbfgs	0.80	0.000000
15	lbfgs	0.85	0.000000
16	lbfgs	0.90	0.000000
17	lbfgs	0.95	0.000000
18	liblinear	0.10	0.319328
19	liblinear	0.15	0.319328
20	liblinear	0.20	0.328684
21	liblinear	0.25	0.330097
22	liblinear	0.30	0.301508
23	liblinear	0.35	0.167147
24	liblinear	0.40	0.072340
25	liblinear	0.45	0.030227
26	liblinear	0.50	0.005249
27	liblinear	0.55	0.000000
28	liblinear	0.60	0.000000
29	liblinear	0.65	0.000000
30	liblinear	0.70	0.000000
31	liblinear	0.75	0.000000
32	liblinear	0.80	0.000000
33	liblinear	0.85	0.000000
34	liblinear	0.90	0.000000
35	liblinear	0.95	0.000000

AJUSTE DE LIMIAR

In [14]:

```
logistic_regression_columns = ['solver', 'threshold', 'f1_score']
logistic_regression_list = []

for solver_id in solver_list:
    for threshold in np.arange(0, 1, 0.05):
        model = LogisticRegression(random_state=42, solver=solver_id)
        model.fit(features_train, target_train)
        probabilities_valid = model.predict_proba(features_valid)
        probabilities_one_valid = probabilities_valid[:, 1]
        predicted_valid = probabilities_one_valid > threshold
        logistic_regression_list.append([solver_id,
                                         threshold,
                                         f1_score(target_valid, predicted_valid)
                                         ]))

logistic_regression = pd.DataFrame(logistic_regression_list, columns=logistic_regression_
logistic_regression
```

Out[14]:

	solver	threshold	f1_score
0	lbfgs	0.00	0.319328
1	lbfgs	0.05	0.319328
2	lbfgs	0.10	0.319328
3	lbfgs	0.15	0.330109
4	lbfgs	0.20	0.331186
5	lbfgs	0.25	0.217918
6	lbfgs	0.30	0.052133
7	lbfgs	0.35	0.000000
8	lbfgs	0.40	0.000000
9	lbfgs	0.45	0.000000
10	lbfgs	0.50	0.000000
11	lbfgs	0.55	0.000000
12	lbfgs	0.60	0.000000
13	lbfgs	0.65	0.000000
14	lbfgs	0.70	0.000000
15	lbfgs	0.75	0.000000
16	lbfgs	0.80	0.000000
17	lbfgs	0.85	0.000000
18	lbfgs	0.90	0.000000
19	lbfgs	0.95	0.000000
20	liblinear	0.00	0.319328
21	liblinear	0.05	0.319328
22	liblinear	0.10	0.319328
23	liblinear	0.15	0.332147
24	liblinear	0.20	0.331169
25	liblinear	0.25	0.211285
26	liblinear	0.30	0.055944
27	liblinear	0.35	0.000000
28	liblinear	0.40	0.000000
29	liblinear	0.45	0.000000
30	liblinear	0.50	0.000000
31	liblinear	0.55	0.000000
32	liblinear	0.60	0.000000
33	liblinear	0.65	0.000000
34	liblinear	0.70	0.000000
35	liblinear	0.75	0.000000
36	liblinear	0.80	0.000000

	solver	threshold	f1_score
37	liblinear	0.85	0.000000
38	liblinear	0.90	0.000000

CONCLUSÃO DE REGRESSÃO LOGÍSTICA

APÓS ANÁLISES E UTILIZAÇÃO DE VÁRIAS TÉCNICAS NO MÉTODO DE REGRESSÃO LOGÍSTICA, O LIMITE DESEJADO EM F1 NÃO FOI ATINGIDO. SERÃO TESTADOS OUTROS MODELOS NA TENTATIVA DE ACHAR O QUE MELHOR TRABALHARÁ A QUESTÃO DE DESEQUILÍBRIO DE CLASSES E OBTER UM MELHOR RESULTADO.

ÁRVORE DE DECISÃO

MODELO DE AJUSTE

In [15]:

```
decision_tree_columns = ['depth', 'acc', 'f1_score']
decision_tree_list = []

for depth in range(1, 12):
    model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    model.fit(features_train, target_train)
    predicted_valid = model.predict(features_valid)
    decision_tree_list.append([depth,
                               accuracy_score(target_valid, predicted_valid),
                               f1_score(target_valid, predicted_valid)])
]

decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)

decision_tree
```

Out[15]:

	depth	acc	f1_score
0	1	0.8100	0.000000
1	2	0.8430	0.474916
2	3	0.8475	0.500818
3	4	0.8545	0.471869
4	5	0.8475	0.507270
5	6	0.8540	0.522876
6	7	0.8560	0.515152
7	8	0.8515	0.521739
8	9	0.8490	0.508143
9	10	0.8485	0.516746
10	11	0.8400	0.504644

AJUSTE DE PESO DE CLASSE

In [16]:

```
decision_tree_columns = ['depth', 'class_weight', 'f1_score']
decision_tree_list = []

for depth in range(1, 12):
    for class_weight in ['balanced', None]:
        model = DecisionTreeClassifier(max_depth=depth, class_weight=class_weight, random_state=42)
        model.fit(features_train, target_train)
        predicted_valid = model.predict(features_valid)
        decision_tree_list.append([depth,
                                    class_weight,
                                    f1_score(target_valid, predicted_valid)])
    )

decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)

decision_tree
```

Out[16]:

	depth	class_weight	f1_score
0	1	balanced	0.423143
1	1	None	0.000000
2	2	balanced	0.442942
3	2	None	0.474916
4	3	balanced	0.442942
5	3	None	0.500818
6	4	balanced	0.510436
7	4	None	0.471869
8	5	balanced	0.515895
9	5	None	0.507270
10	6	balanced	0.517787
11	6	None	0.522876
12	7	balanced	0.555925
13	7	None	0.515152
14	8	balanced	0.511294
15	8	None	0.521739
16	9	balanced	0.532741
17	9	None	0.508143
18	10	balanced	0.512035
19	10	None	0.516746
20	11	balanced	0.514286
21	11	None	0.504644

AMOSTRAGEM

In [17]:

```
decision_tree_columns = ['depth', 'repeat', 'f1_score']
decision_tree_list = []

for depth in range(1,12):
    for repeat in range(1, 5):
        features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
        target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)
        features_upsampled, target_upsampled = shuffle(features_upsampled,
                                                       target_upsampled,
                                                       random_state=42)
        model_dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
        model_dt.fit(features_upsampled, target_upsampled)
        predicted_valid = model_dt.predict(features_valid)
        decision_tree_list.append([depth,
                                   repeat,
                                   f1_score(target_valid, predicted_valid)
                                  ])

decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)

decision_tree
```

Out[17]:

	depth	repeat	f1_score
0	1	1	0.000000
1	1	2	0.455366
2	1	3	0.423143
3	1	4	0.423143
4	2	1	0.474916
5	2	2	0.495652
6	2	3	0.476943
7	2	4	0.442942
8	3	1	0.500818
9	3	2	0.483444
10	3	3	0.500645
11	3	4	0.442942
12	4	1	0.471869
13	4	2	0.511556
14	4	3	0.513661
15	4	4	0.510436
16	5	1	0.507270
17	5	2	0.543641
18	5	3	0.542331
19	5	4	0.496622
20	6	1	0.522876
21	6	2	0.540620
22	6	3	0.549280
23	6	4	0.521224
24	7	1	0.515152
25	7	2	0.544959
26	7	3	0.557823
27	7	4	0.548596
28	8	1	0.521739
29	8	2	0.547771
30	8	3	0.541304
31	8	4	0.510814
32	9	1	0.508143
33	9	2	0.553525
34	9	3	0.526066
35	9	4	0.532597
36	10	1	0.516746

	depth	repeat	f1_score
37	10	2	0.544987
38	10	3	0.526316
39	10	4	0.513859
40	11	1	0.504644
41	11	2	0.525907
42	11	3	0.546539
43	11	4	0.524272

AMOSTRA REDUZIDA

In [18]:

```
decision_tree_columns = ['depth', 'fraction', 'f1_score']
decision_tree_list = []

for depth in range(1,12):
    for fraction in np.arange(0, 1, 0.05):
        features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42),
                                          target_zeros.sample(frac=fraction, random_state=42)])
        features_downsampled, target_downsampled = shuffle(features_downsampled,
                                                          target_downsampled,
                                                          random_state=42)
        model_dt = DecisionTreeClassifier(max_depth=depth, random_state=42)
        model_dt.fit(features_downsampled, target_downsampled)
        predicted_valid = model_dt.predict(features_valid)
        decision_tree_list.append([depth,
                                   fraction,
                                   f1_score(target_valid, predicted_valid)])
        
decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)

decision_tree
```

Out[18]:

	depth	fraction	f1_score
0	1	0.00	0.319328
1	1	0.05	0.319328
2	1	0.10	0.319328
3	1	0.15	0.423143
4	1	0.20	0.423143
...
215	11	0.75	0.504132
216	11	0.80	0.502152
217	11	0.85	0.533923
218	11	0.90	0.518409
219	11	0.95	0.497791

220 rows × 3 columns

AJUSTE DE LIMIAR

In [19]:

```
decision_tree_columns = ['depth', 'threshold', 'f1_score']
decision_tree_list = []

for depth in range(1,12):
    for threshold in np.arange(0, 1, 0.05):
        model = DecisionTreeClassifier(max_depth=depth, random_state=42)
        model.fit(features_train, target_train)
        probabilities_valid = model.predict_proba(features_valid)
        probabilities_one_valid = probabilities_valid[:, 1]
        predicted_valid = probabilities_one_valid > threshold
        decision_tree_list.append([depth,
                                    threshold,
                                    f1_score(target_valid, predicted_valid)])
]

decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)

decision_tree
```

Out[19]:

	depth	threshold	f1_score
0	1	0.00	0.319328
1	1	0.05	0.319328
2	1	0.10	0.319328
3	1	0.15	0.442029
4	1	0.20	0.442029
...
215	11	0.75	0.501639
216	11	0.80	0.502479
217	11	0.85	0.502479
218	11	0.90	0.483592
219	11	0.95	0.289738

220 rows × 3 columns

OBSERVAÇÃO

SERÃO EXPLORADOS VÁRIOS HIPERPARÂMETROS PARA ÁRVORE DE DECISÃO E O SEU EFEITO NA PONTUAÇÃO F1. PARA CADA COMBINAÇÃO DE HIPERPARÂMETRO, OCORRERÁ A REDUÇÃO DA AMOSTRA DE CLASSE AFIM DE RESOLVER O EQUILÍBRIO DE CLASSE, O AJUSTE UM MODELO DE ÁRVORE DE DECISÃO AOS DADOS REDUZIDOS E CALCULA O F1 UTILIZANDO O CONJUNTO DE VALIDAÇÃO. O OBJETIVO É IDENTIFICAR A MAIOR PONTUAÇÃO.

In [20]:

```

decision_tree_columns = ['depth', 'class_weight', 'fraction', 'threshold', 'f1_score']
decision_tree_list = []

for depth in range(1, 12):
    for class_weight in ['balanced', None]:
        for fraction in np.arange(0, 1, 0.1):
            for threshold in np.arange(0, 1, 0.1):
                features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42),
                                                   target_zeros.sample(frac=fraction, random_state=42)])
                target_downsampled = pd.concat([target_zeros.sample(frac=fraction, random_state=42),
                                                 shuffle(features_downsampled,
                                                         target_downsampled,
                                                         random_state=42)])
                model = DecisionTreeClassifier(max_depth=depth, class_weight=class_weight)
                model.fit(features_downsampled, target_downsampled)
                probabilities_valid = model.predict_proba(features_valid)
                if probabilities_valid.shape[1] == 1:
                    probabilities_one_valid = probabilities_valid.ravel()
                else:
                    probabilities_one_valid = probabilities_valid[:, 1]
                predicted_valid = probabilities_one_valid > threshold

                decision_tree_list.append([depth,
                                           class_weight,
                                           fraction,
                                           threshold,
                                           f1_score(target_valid, predicted_valid)
                                         ])

decision_tree = pd.DataFrame(decision_tree_list, columns=decision_tree_columns)
decision_tree

```

Out[20]:

	depth	class_weight	fraction	threshold	f1_score
0	1	balanced	0.0	0.0	0.319328
1	1	balanced	0.0	0.1	0.319328
2	1	balanced	0.0	0.2	0.319328
3	1	balanced	0.0	0.3	0.319328
4	1	balanced	0.0	0.4	0.319328
...
2195	11	None	0.9	0.5	0.518409
2196	11	None	0.9	0.6	0.517751
2197	11	None	0.9	0.7	0.509924
2198	11	None	0.9	0.8	0.490196
2199	11	None	0.9	0.9	0.476027

2200 rows × 5 columns

CONCLUSÃO DA ÁRVORE DE DECISÃO

APÓS AS TÉCNICAS ABORDADAS NA ÁRVORE DE DECISÃO, NÃO CONSEGUIMOS ATINGIR O LIMITE DE PONTUAÇÃO F1 ESTIPULADO. SERÁ ABORDADO UM NOVO MODELO, AFIM DE ATINGIR A PONTUAÇÃO ADEQUADA.

RANDOM FOREST

MODELO DE AJUSTE

In [21]:

```
random_forest_columns = ['estimator', 'acc', 'f1_score']
random_forest_list = []

for estimator in range(10, 101, 10):
    model = RandomForestClassifier(n_estimators=estimator, random_state=42)
    model.fit(features_train, target_train)
    predicted_valid = model.predict(features_valid)
    random_forest_list.append([estimator,
                               accuracy_score(target_valid, predicted_valid),
                               f1_score(target_valid, predicted_valid)])
]

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
random_forest
```

Out[21]:

	estimator	acc	f1_score
0	10	0.8490	0.412451
1	20	0.8565	0.436149
2	30	0.8580	0.447471
3	40	0.8585	0.448343
4	50	0.8605	0.460348
5	60	0.8600	0.457364
6	70	0.8590	0.455598
7	80	0.8585	0.458891
8	90	0.8595	0.462715
9	100	0.8610	0.467433

AJUSTE DE PESO DE CLASSE

In [22]:

```
random_forest_columns = ['estimator', 'class_weight', 'f1_score']
random_forest_list = []

for estimator in range(10, 101, 10):
    for class_weight in ['balanced', None]:
        model = RandomForestClassifier(n_estimators=estimator, class_weight=class_weight,
                                       model.fit(features_train, target_train)
        predicted_valid = model.predict(features_valid)
        random_forest_list.append([estimator,
                                   class_weight,
                                   f1_score(target_valid, predicted_valid)
                                  ])

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
```

Out[22]:

	estimator	class_weight	f1_score
0	10	balanced	0.386847
1	10	None	0.412451
2	20	balanced	0.402344
3	20	None	0.436149
4	30	balanced	0.413255
5	30	None	0.447471
6	40	balanced	0.405512
7	40	None	0.448343
8	50	balanced	0.412574
9	50	None	0.460348
10	60	balanced	0.417154
11	60	None	0.457364
12	70	balanced	0.405512
13	70	None	0.455598
14	80	balanced	0.426357
15	80	None	0.458891
16	90	balanced	0.424125
17	90	None	0.462715
18	100	balanced	0.428571
19	100	None	0.467433

AMOSTRAGEM

In []:

```
random_forest_columns = ['estimator', 'repeat', 'f1_score']
random_forest_list = []

for estimator in range(10, 201, 10):
    for repeat in range(1, 5):
        features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
        target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)
        features_upsampled, target_upsampled = shuffle(features_upsampled,
                                                       target_upsampled,
                                                       random_state=42)
        model = RandomForestClassifier(n_estimators=estimator, random_state=42)
        model.fit(features_upsampled, target_upsampled)
        predicted_valid = model.predict(features_valid)
        random_forest_list.append([estimator,
                                   repeat,
                                   f1_score(target_valid, predicted_valid)
                                  ])

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
random_forest
```

AMOSTRA REDUZIDA

In []:

```
random_forest_columns = ['estimator', 'fraction', 'f1_score']
random_forest_list = []

for estimator in range(10, 201, 10):
    for fraction in np.arange(0, 1, 0.05):
        features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42),
                                         target_zeros.sample(frac=fraction, random_state=42)])
        target_downsampled = pd.concat([target_zeros.sample(frac=fraction, random_state=42),
                                         target_ones.sample(frac=(1-fraction), random_state=42)])
        features_downsampled, target_downsampled = shuffle(features_downsampled,
                                                          target_downsampled,
                                                          random_state=42)
        model = RandomForestClassifier(n_estimators=estimator, random_state=42)
        model.fit(features_downsampled, target_downsampled)
        predicted_valid = model.predict(features_valid)
        random_forest_list.append([estimator,
                                   fraction,
                                   f1_score(target_valid, predicted_valid)
                                  ])

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
random_forest
```

AJUSTE DE LIMIAR

In []:

```
random_forest_columns = ['estimator', 'threshold', 'f1_score']
random_forest_list = []

for estimator in range(10, 201, 10):
    for threshold in np.arange(0, 1, 0.1):
        model = RandomForestClassifier(n_estimators=estimator, random_state=42)
        model.fit(features_train, target_train)
        probabilities_valid = model.predict_proba(features_valid)
        probabilities_one_valid = probabilities_valid[:, 1]
        predicted_valid = probabilities_one_valid > threshold
        random_forest_list.append([estimator,
                                    threshold,
                                    f1_score(target_valid, predicted_valid)])
)

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
random_forest
```

OBSERVAÇÃO

NOVAMENTE VAMOS COMBINAR HIPERPARÂMETROS E VERIFICAR O EFEITO NA PONTUAÇÃO F1 NO MÉTODO DE FLORESTA ALEATÓRIA.

In []:

```

random_forest_columns = ['estimator', 'class_weight', 'fraction', 'threshold', 'f1_score']
random_forest_list = []

for estimator in range(10, 201, 10):
    for class_weight in ['balanced', None]:
        for fraction in np.arange(0, 1, 0.05):
            for threshold in np.arange(0, 1, 0.1):
                features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42),
                                                   target_zeros.sample(frac=fraction, random_state=42)])
                target_downsampled = pd.concat([target_zeros.sample(frac=fraction, random_state=42),
                                                 features_downsampled])
                features_downsampled, target_downsampled = shuffle(features_downsampled,
                                                               target_downsampled,
                                                               random_state=42)
                model = RandomForestClassifier(n_estimators=estimator, class_weight=class_weight)
                model.fit(features_downsampled, target_downsampled)
                probabilities_valid = model.predict_proba(features_valid)
                if probabilities_valid.shape[1] == 1:
                    probabilities_one_valid = probabilities_valid.ravel()
                else:
                    probabilities_one_valid = probabilities_valid[:, 1]
                predicted_valid = probabilities_one_valid > threshold

                random_forest_list.append([estimator,
                                           class_weight,
                                           fraction,
                                           threshold,
                                           f1_score(target_valid, predicted_valid)])
)

random_forest = pd.DataFrame(random_forest_list, columns=random_forest_columns)
random_forest

```

CONCLUSÃO DA FLORESTA ALEATÓRIA

CONSEGUIMOS CONQUISTAR O OBJETIVO DE PONTUAÇÃO F1 NO MÉTODO DE FLORESTA ALEATÓRIA POR MEIO DA AMOSTRA REDUZIDA. UMA PONTUAÇÃO DE 0.5945. PORTANTO, ESTE É O MELHOR MODELO PARA UTILIZAR NO CONJUNTO DE TESTE.

PARÂMETROS DO MELHOR MODELO:\

n_estimators: 160
 class_weight: balanced
 downsampling fraction: 0.85
 threshold: 0.4\

VALOR RESULTANTE DE F1_SCORE: 0.594816

A FLORESTA ALEATÓRIA TEM ALGUMAS DESVANTAGENS. A REDUÇÃO DA AMOSTRA PODE RESULTAR EM PERDA DE INFORMAÇÕES E PODEM SER COMPUTACIONALMENTE CARAS, PODENDO NÃO SER VIÁVEL PARA GRANDES CONJUNTOS DE DADOS.

TESTE DO MODELO

In []:

```

fraction = 0.85
threshold = 0.4

features_downsampled = pd.concat([features_zeros.sample(frac=fraction, random_state=42)])
target_downsampled = pd.concat([target_zeros.sample(frac=fraction, random_state=42)] + [
    features_downsampled, target_downsampled = shuffle(features_downsampled,
                                                       target_downsampled,
                                                       random_state=42)
model = RandomForestClassifier(n_estimators=160, class_weight='balanced', random_state=42)
model.fit(features_downsampled, target_downsampled)
probabilities_test = model.predict_proba(features_test)
if probabilities_test.shape[1] == 1:
    probabilities_one_test = probabilities_test.ravel()
else:
    probabilities_one_test = probabilities_test[:, 1]
predicted_test = probabilities_one_test > threshold

f1_score(target_test, predicted_test)

```

CURVA PR

SERÁ UTILIZADO A CURVA PR, POIS TEMOS NESTE PROJETO CLASSES DESBALANCEADAS, A CLASSE NEGATIVA É SUPERIOR A CLASSE POSITIVA. A CURVA PR AJUDA NA AVALIAÇÃO DE DESEMPENHO DO CLASSIFICADOR PARA A CLASSE POSITIVA, SEPARADAMENTE DA CLASSE NEGATIVA, O QUE É IMPORTANTE PARA CONJUNTO DE DADOS DESBALANCEADOS.

In []:

```

precision, recall, thresholds = precision_recall_curve(target_test, probabilities_test[:, 1])

plt.figure(figsize=(6, 6))
plt.step(recall, precision, where='post')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall Curve')
plt.show()

```

CURVA ROC

SERÁ GERADO UMA CURVA ROC PARA AVALIAR O DESEMPENHO DE UM MODELO DE CLASSIFICAÇÃO BINÁRIO.

In []:

```
fpr, tpr, thresholds = roc_curve(target_test, probabilities_one_test)

plt.figure()
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.show()
```

AUC-ROC

A CURVA ROC É UM GRÁFICO DE VERDEDEIROS POSITIVOS (TPR) EM RELAÇÃO À TAXA DE FALSOS POSITIVOS (FPR). O AUC-ROC É UM VALOR QUE QUANTIFICA O DESEMPENHO GERAL DO CLASSIFICADOR. UM AUC-ROC DE 1, INDICA UM DESEMPENHO DE CLASSIFICAÇÃO PERFEITO, ENQUANTO UM AUC-ROC DE VALOR 0.5, INDICA ALEATORIEDADE.

ALÉM DA PONTUAÇÃO F1, O AUC-ROC É UMA MÉTRICA IMPORTANTE NESTE PROJETO, POIS REVELA COM MAIS CLAREZA A CAPACIDADE DO CLASSIFICADOR DE DISTINGUIR AS AMOSTRAS POSITIVAS E NEGATIVAS.

In []:

```
auc_roc = roc_auc_score(target_test, probabilities_one_test)

print(auc_roc)
```

CONCLUSÃO GERAL

O OBJETIVO DESTE PROJETO É CRIAR UM MODELO CAPAZ DE PREVER SE UM CLIENTE SAIRÁ OU NÃO DA INSTITUIÇÃO BANCÁRIA.

ALGUMAS DAS BIBLIOTECAS FORAM IMPORTADAS CONFORME A NECESSIDADE, E OUTRAS FORAM PREVIAMENTE IMPORTADAS. FORAM REALIZADOS OS ESTUDOS DOS DADOS E ASSIM FORAM IDENTIFICADOS VALORES AUSENTES E OS TRATAMOS DE FORMA ADEQUADA. LOGO APÓS, IDENTIFICAMOS AS VARIÁVEIS CATEGÓRICAS.

O CONJUNTO DE DADOS FOI BALANCEADO UTILIZANDO A AMOSTRA REDUZIDA PARA EXEMPLIFICAR O PROBLEMA DE DESEQUILÍBRIO DE CLASSESS.

TRÊS MODELOS FORAM UTILIZADOS:

REGRESSÃO LOGÍSTICA

ÁRVORE DE DECISÃO

FLORESTA ALEATÓRIA

OS HIPERPARÂMETROS FORAM AJUSTADOS. O MELHOR MODELO (FLORESTA ALEATÓRIA) FOI SELECIONADO BASEADO NO VALOR DA PONTUAÇÃO F1, APLICADO NO CONJUNTO DE VALIDAÇÃO, E O TESTAMOS NO CONJUNTO DE TESTE.

PARA MELHOR AVALIAÇÃO DO DESEMPENHO DO MODELO, UTILIZAMOS A ANÁLISE DA CURVA DE RECUPERAÇÃO DE PRECISÃO E A CURVA ROC.

FLORESTA ALEATÓRIA

ESTIMADORES : 160 PESO DE CLASSE: balanced FRAÇÃO DE AMOSTRA REDUZIDA: 0.85 LIMIAR: 0.4
F1 CONJUNTO DE VALIDAÇÃO: 0.590 F1 CONJUNTO DE TESTE: 0.642

A PONTUAÇÃO F1 ATINGIU O OBJETIVO DE 0.590 NO MODELO DE FLORESTA ALEATÓRIA.

A CURVA ROC MOSTRA QUE O MODELO TEVE UMA BOA TAXA DE VERDADEIROS POSITIVOS E UMA BAIXA TAXA DE FALSOS POSITIVOS. A PONTUAÇÃO AUC-ROC É 0.863.

ESTE MODELO PODE SER UTILIZADO PARA BUSCAR CLIENTES QUE CORREM ALTO RISCO DE

In []:

