Classificação de risco de crédito usando Métodos Ensemble (II)

Objetivo

Nesta segunda etapa, o objetivo será avaliar o quanto a acurácia dos modelos de classificação com **métodos ensemble** melhorará, através do balanceamento das classes da base de dados e do ajustamento de hiperparâmetros dos algortimos selecionados. Idealmente, esses experimentos visam encontrar acurácia em teste = ou > a 90%.

Algoritmos de classificação e de ajustes

- Random Forest
- Extremely Randomized Forest
- Gradient Boosting

Os ajustes serão feitos com:

- Randomized Searsch

Bibliotecas e funções

In [1]:

```
import pandas as pd
import numpy as np
import sklearn as sk
import matplotlib as mpl
from matplotlib import pyplot as plt

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
```

```
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE

import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

Base de dados e sua separação em treino e teste

In [2]:
```
credit_data = pd.read_excel('credit.xls', skiprows = 1)
```

In [3]:
```
# Variáveis da base de dados
print(credit_data.columns)
```

```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default payment next month'],
      dtype='object')
```

In [4]:
```
# Quantidades de observações e de variáveis
print(credit_data.shape)
```

```
(30000, 25)
```

Variável-target

In [5]:
```
target = 'default payment next month'
y = np.asarray(credit_data[target])
```

Variáveis explicativas

In [6]:
```
features = credit_data.columns.drop(['ID', target])
X = np.asarray(credit_data[features])
```

Bases para treino e para teste

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 10)
```

Modelo 1 - Random Forest

Construção do modelo com 50 árvores de decisão (n_estimators)

In [8]:
```
random = RandomForestClassifier(n_estimators = 50)
random.fit(X_train, y_train)
print("Acurácia em treino = ", random.score(X_train, y_train))

random_pred = random.predict(X_test)
print("Acurácia em teste = ", accuracy_score(y_test, random_pred))
print(confusion_matrix(y_test, random_pred))
```

```
Acurácia em treino =  0.9989523809523809
Acurácia em teste =  0.8148888888888889
[[6600  382]
 [1284  734]]
```

Comentário: a acurácia em treino de 99,9% mostra que existe *overfitting* nesse modelo. Isso é reforçado pelo resultado obtido com a base de teste = 81,49% (diferença de 18.4% entre eles).

In [9]:
```
# Resultados da predição
print(classification_report(y_test, random_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      6982
           1       0.66      0.36      0.47      2018

    accuracy                           0.81      9000
   macro avg       0.75      0.65      0.68      9000
weighted avg       0.80      0.81      0.79      9000
```

Comentário: um dos problemas dessa base de dados é o desbalanceamento de suas classes, com 6982 dados para a classe "0" e somente 2018 para a "1". Isso fez com que o modelo aprendesse mais sobre os clientes que *não* atrasaram o pagamento (precisão=84% e sensibilidade=95%) do que

sobre os *inadimplentes* (precisão=66% e sensibilidade=36%).

Balanceamento das classes na base de dados com a função SMOTE

In [10]:
```
sm = SMOTE(random_state = 1)
X, y = sm.fit_resample(X, y)
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 10)
```

Recalculando o modelo de Random Forest com as classes balanceadas

In [12]:
```
random1 = RandomForestClassifier(n_estimators = 50)
random1.fit(X_train, y_train)
print("Acurácia em treino = ", random1.score(X_train, y_train))

random1_pred = random1.predict(X_test)
print("Acurácia em teste = ", accuracy_score(y_test, random1_pred))
print(confusion_matrix(y_test, random1_pred))
```

```
Acurácia em treino =  0.9990522486165887
Acurácia em teste =  0.8307297239460731
[[5947 1019]
 [1354 5699]]
```

In [13]:
```
# Resultados da predição
print(classification_report(y_test, random1_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83      6966
           1       0.85      0.81      0.83      7053

    accuracy                           0.83     14019
   macro avg       0.83      0.83      0.83     14019
weighted avg       0.83      0.83      0.83     14019
```

Comentário: agora, as classes estão balanceadas, com 6966 dados na classe "0" (precisão=81% e sensibilidades=85%) e 7053 na classe "1" (precisão=85% e sensibilidade=81%).

Otimizando o modelo de Random Forest com o algoritmo Randomized Search

O objetivo será conseguir acurárias em treino e teste com valores próximos um do outro, para que o modelo possa ser considerado generalizável para dados de novos clientes.

In [14]:
```
# Parâmetros a serem testados
param_dist = {"max_depth": [4, 6, 8, 10],
              "max_features": [10, 16, 18, 20],
              "min_samples_split": [2, 4, 8, 10, 16, 18],
              "min_samples_leaf": [2, 3, 4, 5, 6, 8],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Combinando parâmetros
rs_search = RandomizedSearchCV(random1, param_distributions = param_dist, n_iter = 10, return_train_score = True)
rs_search.fit(X_train, y_train)

# Identificando o melhor estimador
bestrs = rs_search.best_estimator_
print (bestrs)
```

```
RandomForestClassifier(max_depth=10, max_features=10, min_samples_leaf=4,
                       min_samples_split=16, n_estimators=50)
```

Comentário: dentre os parâmetros apresentados ao novo modelo, o melhor foi construído com as configurações acima.

In [15]:
```
# Nova predição usando melhor estimador
rs_pred = bestrs.predict(X_test)
print("Acurácia em treino = ", rs_search.score(X_train, y_train))
print("Acurácia em teste = ", accuracy_score(y_test, rs_pred))
print(confusion_matrix(y_test, rs_pred))
```

```
Acurácia em treino =  0.8154024886116971
Acurácia em teste =  0.786789357300806
[[5724 1242]
 [1747 5306]]
```

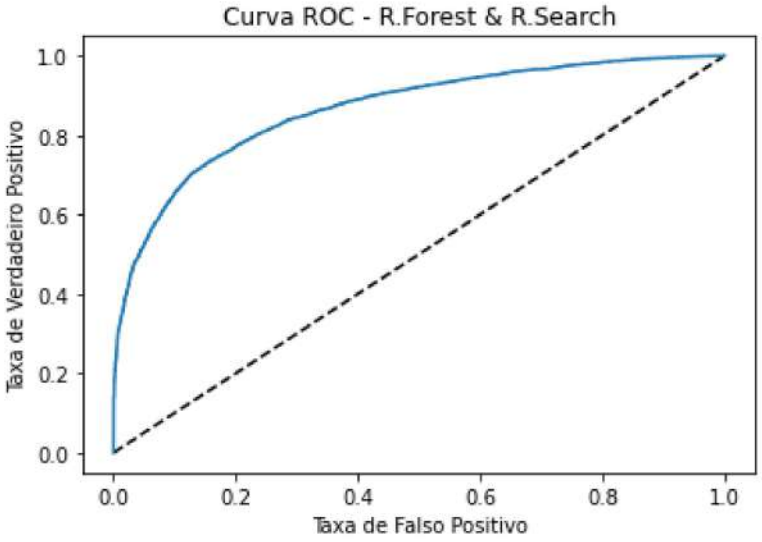
Comentário: as acurácias desse modelo, onde treino = 82% e teste = 79%, ficaram mais equilibrados do que as vistas no modelo anterior.

In [16]:
```
# Resultados da predição
print(classification_report(y_test, rs_pred))
```

	precision	recall	f1-score	support
0	0.77	0.82	0.79	6966
1	0.81	0.75	0.78	7053
accuracy			0.79	14019
macro avg	0.79	0.79	0.79	14019
weighted avg	0.79	0.79	0.79	14019

In [17]:

```
# Visualização dos % de Verdadeiros Positivos x Falsos Negativos através da Curva ROC
credito_pred_prob0 = bestrs.predict_proba(X_test)[:,1]

fpr0, tpr0, thresholds = roc_curve(y_test, credito_pred_prob0)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr0, tpr0)
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC - R.Forest & R.Search')
plt.show()
```



Modelo 2 - Extremely Randomized Forest (Extra Trees)

In [18]:
```
extratrees = ExtraTreesClassifier(n_estimators = 50)
extratrees.fit(X_train, y_train)
print("Acurácia em treino = ", extratrees.score(X_train, y_train))

extra_pred = extratrees.predict(X_test)
print("Acurácia em Teste:", accuracy_score(y_test, extra_pred))
print(confusion_matrix(y_test, extra_pred))
```

```
Acurácia em treino =  0.9991133938671314
Acurácia em Teste: 0.8365789285969042
[[5956 1010]
 [1281 5772]]
```

In [19]:
```
# Resultados da predição
print(classification_report(y_test, extra_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      6966
           1       0.85      0.82      0.83      7053

    accuracy                           0.84     14019
   macro avg       0.84      0.84      0.84     14019
weighted avg       0.84      0.84      0.84     14019
```

Comentário: as acurácias do primeiro modelo usando Extra Trees mostra que ele precisa ter os hiperparâmetros otimizados.

Otimizando os hiperparâmetros com Randomized Search

In [20]:
```
# Parâmetros a serem testados
param_dist_et = {"max_depth": [4, 6, 8, 10],
                 "max_features": [15, 16, 18, 20],
                 "min_samples_split": [2, 4, 8, 10, 16, 18, 20],
                 "min_samples_leaf": [2, 3, 4, 6, 8, 10],
                 "bootstrap": [True, False]}

# Combinando parâmetros
rs_extrees = RandomizedSearchCV(extratrees, param_distributions = param_dist_et, n_iter = 20, return_train_score = True)
rs_extrees.fit(X_train, y_train)
```

```
# Melhor estimador
bestrset = rs_extrees.best_estimator_
print (bestrset)
```

```
ExtraTreesClassifier(max_depth=10, max_features=16, min_samples_leaf=2,
                     min_samples_split=16, n_estimators=50)
```

In [21]:

```
# Nova predição
rs_et_pred = bestrset.predict(X_test)
print("Acurácia em treino = ", rs_extrees.score(X_train, y_train))
print("Acurácia em teste = ", accuracy_score(y_test, rs_et_pred))
print(confusion_matrix(y_test, rs_et_pred))
```

```
Acurácia em treino =  0.7805496958023785
Acurácia em teste =  0.7693130751123475
[[5726 1240]
 [1994 5059]]
```

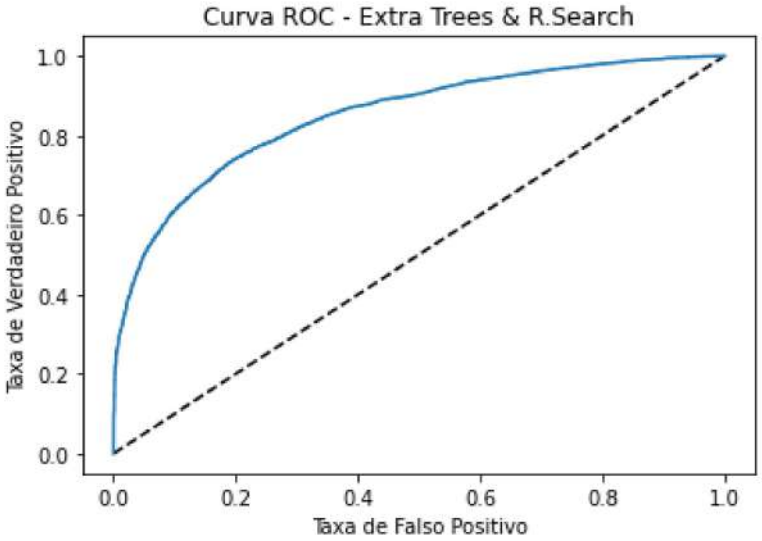
In [22]:

```
# Resultados da predição
print(classification_report(y_test, rs_et_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.82      0.78      6966
           1       0.80      0.72      0.76      7053

    accuracy                           0.77     14019
   macro avg       0.77      0.77      0.77     14019
weighted avg       0.77      0.77      0.77     14019
```

In [23]:

```
# Visualização dos % de Verdadeiros Positivos x Falsos Negativos através da Curva ROC
credito_pred_prob2 = bestrset.predict_proba(X_test)[:,1]

fpr2, tpr2, thresholds = roc_curve(y_test, credito_pred_prob2)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr2, tpr2)
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC - Extra Trees & R.Search')
plt.show()
```



Modelo 3 - Gradient Boosting

In [24]:

```
gboost = GradientBoostingClassifier(n_estimators = 50)
gboost.fit(X_train, y_train)
print("Acurácia em treino =", gboost.score(X_train, y_train))

gboost_pred = gboost.predict(X_test)
print("Acurácia em Teste:", accuracy_score(y_test, gboost_pred))
print(confusion_matrix(y_test, gboost_pred))
```

```
Acurácia em treino = 0.7706747378397383
Acurácia em Teste: 0.7730223268421428
[[5664 1302]
 [1880 5173]]
```

In [25]:

```
# Resultados da predição
print(classification_report(y_test, gboost_pred))
```

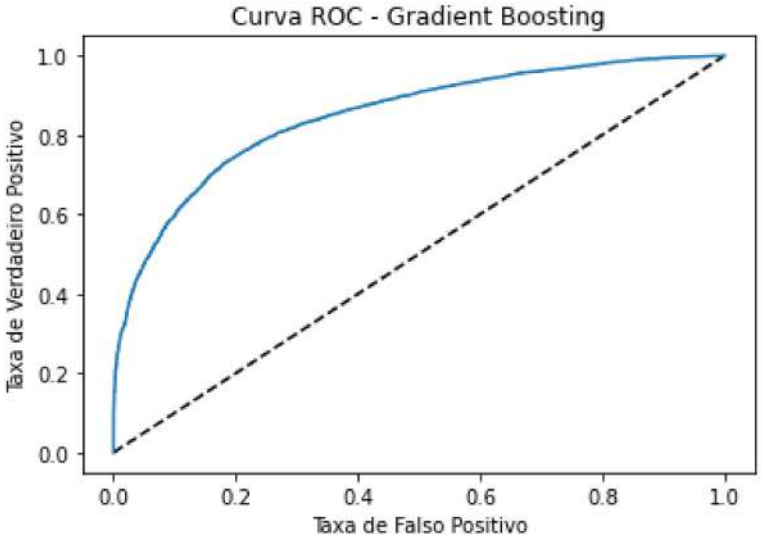
```
              precision    recall  f1-score   support

           0       0.75      0.81      0.78      6966
```

```
                1       0.80      0.73      0.76      7053

         accuracy                           0.77     14019
        macro avg       0.77      0.77      0.77     14019
     weighted avg       0.78      0.77      0.77     14019
```

Comentário: primeiro modelo usando Gradiente Boosting apresentou resultados próximos em treino=77,07% e teste=77,30%; um resultado bom, por ser superior a 75%, mas que ainda pode ser melhorado.

In [26]:

```
# Visualização dos % de Verdadeiros Positivos x Falsos Negativos através da Curva ROC
credito_pred_prob4 = gboost.predict_proba(X_test)[:,1]

fpr4, tpr4, thresholds = roc_curve(y_test, credito_pred_prob4)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr4, tpr4)
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC - Gradient Boosting')
plt.show()
```



Otimizando os hiperparâmetros com Randomized Search

In [27]:
```
# Parâmetros a serem testados
param_dist_gb = {"max_depth": [4, 6, 8, 10],
                 "max_features": [15, 16, 18, 20],
                 "min_samples_split": [2, 4, 6, 8, 10, 15],
                 "min_samples_leaf": [1, 2, 3, 4, 5, 6, 8, 10],
                 "learning_rate": [1.0, 0.1, 0.01]}

# Combinando parâmetros
rs_gb = RandomizedSearchCV(gboost, param_distributions = param_dist_gb, n_iter = 20, return_train_score = True)
rs_gb.fit(X_train, y_train)

# Melhor estimador
bestrsgb = rs_gb.best_estimator_
print (bestrsgb)
```

```
GradientBoostingClassifier(max_depth=10, max_features=18, min_samples_leaf=5,
                           min_samples_split=15, n_estimators=50)
```

In [28]:
```
# Nova predição
rs_gb_pred = bestrsgb.predict(X_test)
print("Acurácia em treino = ", rs_gb.score(X_train, y_train))
print("Acurácia em teste = ", accuracy_score(y_test, rs_gb_pred))
print(confusion_matrix(y_test, rs_gb_pred))
```

```
Acurácia em treino =  0.9244244703292672
Acurácia em teste =  0.8222412440259648
[[5921 1045]
 [1447 5606]]
```

Comentário: a otimização dos hiperparâmetros causou um aparente *overfitting* visto que a acurácia em treino (92,44%) está 10% maior do que a observada em teste (82,22%).

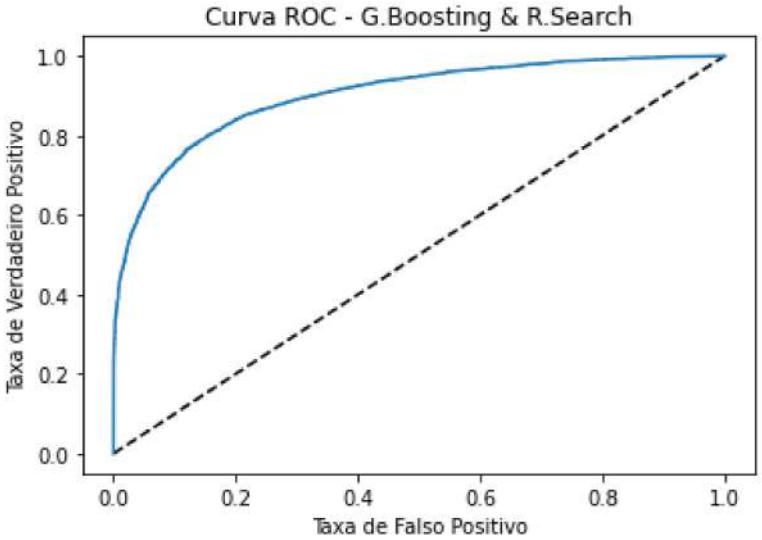
In [29]:
```
# Resultados da predição
print(classification_report(y_test, rs_gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.83      6966
           1       0.84      0.79      0.82      7053

    accuracy                           0.82     14019
   macro avg       0.82      0.82      0.82     14019
```

```
weighted avg       0.82      0.82      0.82     14019
```

In [30]:

```
# Visualização dos % de Verdadeiros Positivos x Falsos Negativos através da Curva ROC
credito_pred_prob5 = bestrsgb.predict_proba(X_test)[:,1]

fpr5, tpr5, thresholds = roc_curve(y_test, credito_pred_prob5)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr5, tpr5)
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC - G.Boosting & R.Search')
plt.show()
```



Conclusão: nesta segunda parte do experimento, conclui-se que o modelo otimizado do Random Forest foi o que apresentou o melhor resultado, com acurária em teste (79%) próxima da de treino (82%; diferença de 3%), enquanto que o resultado do modelo otimizado do Gradient Boosting, apesar de ser maior (82%), aparentou sofrer de *overfitting* (93% em treino; diferença de 11%).

Na **Parte 3** desse projeto, os hiperparâmentros dos 3 modelos de classificação criarão 100 árvores de decisão e serão ajustados usando o algortimo **Grid Search.**