Detecção de Fraudes em Cartões de Crédito com Machine Learning

Alunos(as): Laura Oliveira e Yasmin Faraj

Utilizando o arquivo “creditcard - menor balanceado.csv”, optamos por testar varias técnicas até chegar à melhor eficácia, partimos da seleção de atributos (Feature Selection), técnicas de balanceamento (Sampling) e tuning de hiperparâmetros.

Com a **feature selection** (Random Forest, Select Kbest e Recursive Feature Elimination (RFE)) esses foram os resultados:

```
Teste,k,n_features,Acur◆cia SelectKBest,Acur◆cia RFE,Acur◆cia Random Forest,Melhor Modelo,Melhor Acur◆cia
1,5,5,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
2,5,10,0.9522727272727273,0.9545454545454546,0.9568181818181818,Random Forest,0.9568181818181818
3,5,15,0.9522727272727273,0.9477272727272728,0.95,SelectKBest,0.9522727272727273
4,5,20,0.9522727272727273,0.9522727272727273,0.9522727272727273,SelectKBest,0.9522727272727273
5,5,25,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
6,10,5,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
7,10,10,0.9522727272727273,0.9545454545454546,0.9568181818181818,Random Forest,0.9568181818181818
8,10,15,0.9522727272727273,0.9477272727272728,0.95,SelectKBest,0.9522727272727273
9,10,20,0.9522727272727273,0.9522727272727273,0.9522727272727273,SelectKBest,0.9522727272727273
10,10,25,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
11,15,5,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
12,15,10,0.9522727272727273,0.9545454545454546,0.9568181818181818,Random Forest,0.9568181818181818
13,15,15,0.9522727272727273,0.9477272727272728,0.95,SelectKBest,0.9522727272727273
14,15,20,0.9522727272727273,0.9522727272727273,0.9522727272727273,SelectKBest,0.9522727272727273
15,15,25,0.9522727272727273,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
16,20,5,0.95,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
17,20,10,0.95,0.9545454545454546,0.9568181818181818,Random Forest,0.9568181818181818
18,20,15,0.95,0.9477272727272728,0.95,SelectKBest,0.95
19,20,20,0.95,0.9522727272727273,0.9522727272727273,RFE,0.9522727272727273
20,20,25,0.95,0.9522727272727273,0.9545454545454546,Random Forest,0.9545454545454546
21,25,5,0.9545454545454546,0.9522727272727273,0.9545454545454546,SelectKBest,0.9545454545454546
22,25,10,0.9545454545454546,0.9545454545454546,0.9568181818181818,Random Forest,0.9568181818181818
23,25,15,0.9545454545454546,0.9477272727272728,0.95,SelectKBest,0.9545454545454546
24,25,20,0.9545454545454546,0.9522727272727273,0.9522727272727273,SelectKBest,0.9545454545454546
25,25,25,0.9545454545454546,0.9522727272727273,0.9545454545454546,SelectKBest,0.9545454545454546
```

Utilizamos uma variação do k e do n_features de 5 em 5 para descobrir diferentes valores para diferentes tipos de teste. Será notável que está é um dos piores resultados.

```
Acurácia antes de usar técnicas:  0.9113636363636364
Matriz de confusão antes das técnicas:
 [[298  16]
 [ 23 103]]
```

Já esse é o resultado da **feature selection** sem nenhum tipo de técnica aplicada.

Com o **sampling** nosso resultado mais agradável foi no teste 13 que tinha o k = 15 e o n_features = 15

```
Teste 13:
Acurácia com SelectKBest (SMOTE): 0.9545454545454546
Acurácia com RFE (SMOTE): 0.95
Acurácia com Random Forest (SMOTE): 0.9454545454545454
Acurácia com SelectKBest (Undersampling): 0.9545454545454546
Acurácia com RFE (Undersampling): 0.95
Acurácia com Random Forest (Undersampling): 0.9454545454545454
```

A tendência foi do **selectKBest** se manter com o mesmo valor que tinha só com as features, o RFE cresceu um pouco os valores de acurácia e do **RandomForest** caiu.

Aqui vemos também a tendência da acurácia crescer principalmente com a junção de **xgboost** e **gridsearch**:

```
XGBoost com RFE (Grid),0.9991876523151909
XGBoost com RFE (Random),0.9967506092607636
SVM com RFE (Grid),0.9675060926076361
SVM com RFE (Random),0.9675060926076361
XGBoost com Random Forest (Grid),0.9991876523151909
XGBoost com Random Forest (Random),0.9967506092607636
SVM com Random Forest (Grid),0.9658813972380179
SVM com Random Forest (Random),0.9658813972380179
XGBoost com SelectKBest (Grid),0.9991876523151909
XGBoost com SelectKBest (Random),0.9959382615759546
SVM com SelectKBest (Grid),0.9675060926076361
SVM com SelectKBest (Random),0.9675060926076361
```

Independente do **feature selection** o resultado foi igual para cada tipo de ajuste e otimização específico. Nosso melhor resultado é usando **xgboost** e **gridsearch**, com a variação de features sem impactar em nada.

```
Descrição,Resultados
XGBoost com RFE (Grid),"(0.9991876523151909, np.float64(0.9989868788048626), array([[889,   1],
       [  0, 341]]))"
XGBoost com RFE (Random),"(0.9967506092607636, np.float64(0.995943853174734), array([[888,   2],
       [  2, 339]]))"
SVM com RFE (Grid),"(0.9675060926076361, np.float64(0.9585183887207758), array([[882,   8],
      [ 32, 309]]))"
SVM com RFE (Random),"(0.9675060926076361, np.float64(0.9585183887207758), array([[882,   8],
       [ 32, 309]]))"
XGBoost com Random Forest (Grid),"(0.9991876523151909, np.float64(0.9989868788048626), array([[889,   1],
       [  0, 341]]))"
XGBoost com Random Forest (Random),"(0.9967506092607636, np.float64(0.995936489073744), array([[889,   1],
       [  3, 338]]))"
SVM com Random Forest (Grid),"(0.9658813972380179, np.float64(0.9563598979013046), array([[882,   8],
       [ 34, 307]]))"
SVM com Random Forest (Random),"(0.9658813972380179, np.float64(0.9563598979013046), array([[882,   8],
       [ 34, 307]]))"
XGBoost com SelectKBest (Grid),"(0.9991876523151909, np.float64(0.9989868788048626), array([[889,   1],
       [  0, 341]]))"
XGBoost com SelectKBest (Random),"(0.9959382615759546, np.float64(0.9949159836353318), array([[889,   1],
       [  4, 337]]))"
SVM com SelectKBest (Grid),"(0.9675060926076361, np.float64(0.9585183887207758), array([[882,   8],
       [ 32, 309]]))"
SVM com SelectKBest (Random),"(0.9675060926076361, np.float64(0.9585183887207758), array([[882,   8],
       [ 32, 309]]))"
```

Nossa análise envolveu uma abordagem com várias técnicas de seleção de atributos, balanceamento de dados e otimização de hiperparâmetros. No primeiro teste, foram usados os features selection Random Forest, SelectKBest e Recursive Feature Elimination (RFE), com isso, foi possível observar que o RFE se encaixou como pior dos métodos, pois durante os teste foi o que apareceu menos, enquanto o Random Forest e o SelectKBest alternavam entre os melhores.

Observamos que o processo de sampling, usando métodos como Random Forest, SelectKBest e Recursive Feature Elimination (RFE), trouxe resultados variados: enquanto o SelectKBest se manteve relativamente consistente, o RFE apresentou um pequeno aumento na acurácia, enquanto o RandomForest teve uma ligeira queda.

Além disso, as técnicas de balanceamento, especialmente no teste 13 (com k = 15 e n_features = 15), trouxeram um aumento significativo na performance do modelo. No entanto, o uso de XGBoost aliado ao GridSearch foi o fator decisivo para alcançar a melhor performance. Observamos que a precisão do modelo mostrou-se mais robusta com essa combinação, independentemente do tipo de seleção de atributos utilizado.

Assim, concluímos que, para o nosso cenário de detecção de fraudes, a combinação de XGBoost com ajuste de hiperparâmetros por GridSearch foi a estratégia mais eficaz, consolidando-se como a escolha ideal para alcançar uma alta precisão no modelo, enquanto os métodos de feature selection apresentaram uma contribuição secundária no resultado final.