desafio

April 25, 2023

1 Desafio - Bázico

1.0.1 Por Vitor Anfrizio

Para este desafio, usei as bibliotecas *pandas*, *numpy*, *matplotlib* e *pickle*.

Como usei o *Google Colab* como ambiente, fiz o upload do arquivo "vendas_de_produtos.csv" no *Google Drive*.

De início, modifiquei a coluna "Data" para o formato aceitável de data e criei uma coluna que indica se a compra foi feita nos últimos 14 dias. Após isso, transformei as descrições em variáveis categóricas, removi dados nulos e algumas colunas que não seriam necessárias à análise.

[1]:
```
import pandas as pd
import numpy as np
import pickle

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

[58]:
```
from google.colab import drive
drive.mount('/content/drive')

dados = pd.read_excel("/content/drive/MyDrive/vendas_de_produtos.xlsx")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[59]:
```
dados['Data'] = pd.to_datetime(dados['Data'])
dados['Compra_14_Dias'] = (dados['Data'].max() - dados['Data']).dt.days <= 14
dados['Categoria_Produto'] = pd.Categorical(dados['Descrição_Produto']).codes
```

[60]:
```
dados = dados.drop(['ID_Produto', 'Descrição_Produto', 'ID_Pedido',␣
 ↪'Preço_Unitário', 'Desconto'], axis=1)
dados = dados[dados['ID_Cliente'].notna()]
dados = dados[dados['Data'].notna()]
dados = dados[dados['Compra_14_Dias'].notna()]
```

```
print(dados)
```

```
        ID_Cliente       Data  Quantidade  Frete  Total_do_Pedido  \
0                1 2021-05-27           1   4.90           113.90
1                2 2021-05-26           2   4.90           301.90
2                2 2021-05-26           1   4.90           301.90
3                3 2021-05-17           1  24.97           133.97
4                4 2021-05-17           1   4.90           192.90
…              …        …          …     …                …
17021  15990645865 2022-12-31           1   0.00           214.20
17022  15990645865 2022-12-31           1   0.00           214.20
17023  15990665892 2022-12-31           1  21.19           140.19
17024  15990672946 2022-12-31           1   0.00           214.20
17025  15990672946 2022-12-31           1   0.00           214.20

       Compra_14_Dias  Categoria_Produto
0               False                300
1               False                300
2               False                285
3               False                300
4               False                288
…               …                  …
17021            True                431
17022            True                454
17023            True                312
17024            True                559
17025            True                526

[17026 rows x 7 columns]
```

A partir deste ponto, separei os dados em conjuntos de treino e teste, usando a função train_test_split. Os dados de entrada, que são as informações dos clientes, foram selecionados usando a função drop. A partir daqui começa a parte preditiva.

70% dos dados foram usados para treino (X_train e y_train) e 30% para teste (X_test e y_test). Essa divisão permitiu avaliar a capacidade de generalização do modelo.

Daqui, criei uma instância da classe LogisticRegression e usei a função fit para treinar o modelo nos dados de treino. Fiz isso para encontrar os coeficientes da equação de regressão logística que se ajustam melhor aos dados.

Como último ponto até aqui, usei o modelo treinado para fazer previsões nos dados de teste (y_pred) e comparei com as respostas reais (y_test) pra avaliar a qualidade do modelo.

[61]:
```
X_train, X_test, y_train, y_test = train_test_split(
    dados.drop(['ID_Cliente', 'Data', 'Compra_14_Dias'], axis=1),
    dados['Compra_14_Dias'],
    test_size=0.3,
    random_state=42
```

```
)
```

[62]:
```
modelo = LogisticRegression()
modelo.fit(X_train, y_train)

y_pred = modelo.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       False       0.88      1.00      0.93      4471
        True       1.00      0.00      0.01       637

    accuracy                           0.88      5108
   macro avg       0.94      0.50      0.47      5108
weighted avg       0.89      0.88      0.82      5108
```

Como visto acima, o modelo gerado teve uma acurácia de 88% (previu corretamenta, em 88% das vezes, se um cliente faria uma nova compra nos próximos 14 dias ou não).

Em contrapartida, o recall da classe "True" (propensos a fazer recompra) ficou zerado (o modelo não conseguiu identificar corretamente nenhum cliente que realmente fez uma nova compra nesse período), indicando que o modelo precisa de alguns ajustes.

Uma das soluções que me veio à mente foi o uso da técnica SMOTE (Synthetic Minority Oversampling Technique) pra gerar, com base nas observações que já existem, novas observações sintéticas para a classe dos clientes que fizeram recompra.

Para isso, usei a biblioteca *imbalanced-learn*. Apliquei a SMOTE no conjunto de treino, retreinei o modelo e fiz as previsões com o conjunto reamostrado

[63]:
```
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
```

[64]:
```
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

modelo = RandomForestClassifier(random_state=42)
modelo.fit(X_train_res, y_train_res)

y_pred = modelo.predict(X_test)

print(classification_report(y_test, y_pred))

# Salvando o modelo
with open('modelo_classificacao.pkl', 'wb') as arquivo_modelo:
    pickle.dump(modelo, arquivo_modelo)
```

```
              precision    recall  f1-score   support

       False       0.94      0.85      0.90      4471
        True       0.38      0.62      0.47       637

    accuracy                           0.83      5108
   macro avg       0.66      0.74      0.68      5108
weighted avg       0.87      0.83      0.84      5108
```

Com a aplicação da SMOTE, houve uma melhora no recall da classe de clientes que farão recompra (minoritária), passando de 0.0 para 0.62. No entanto, a precisão da classe minoritária caiu, passando de 1.0 para 0.38, indicando que o modelo está prevendo muitos falsos positivos. Em termos mais claros: o modelo está "acertando" 62% das recompras, mas indica que 38% dos clientes que não farão recompra, na verdade, farão.

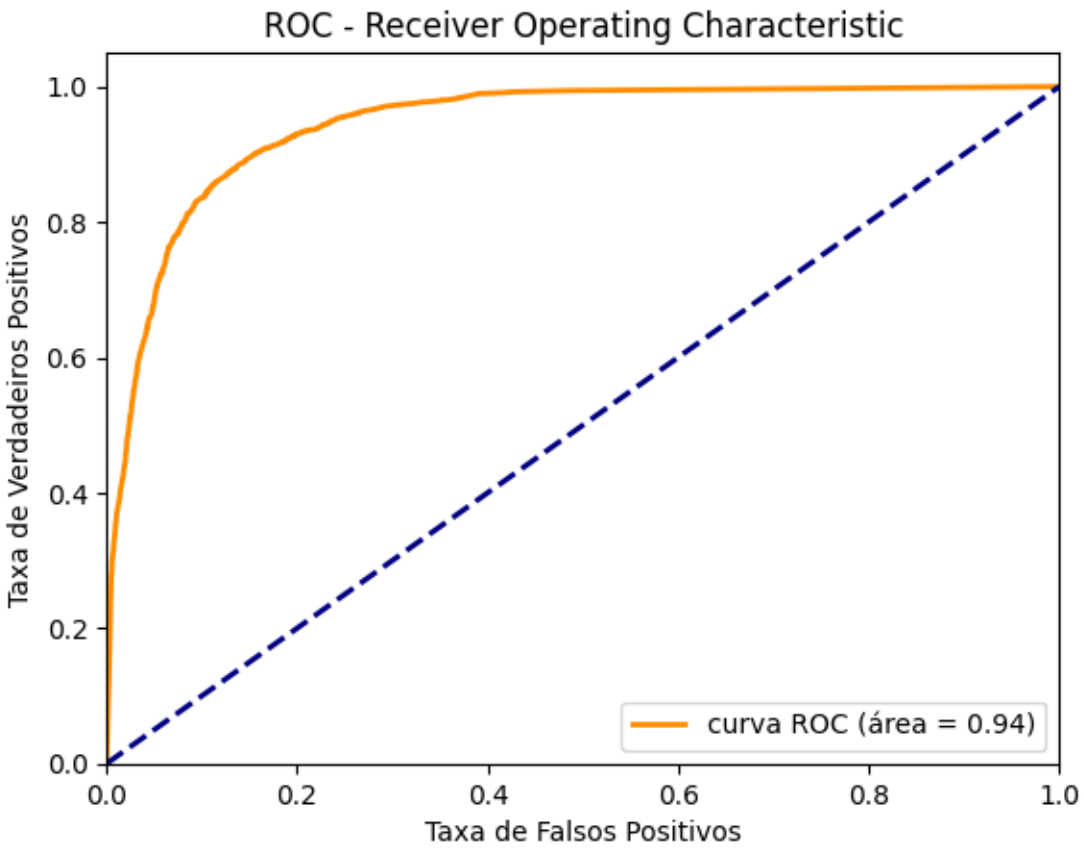
Apesar disso, apliquei a curva ROC para avaliar melhor o desempenho do modelo. Separei o conjunto de teste, calculei as probabilidades de classe, AUC e plotei o gráfico da curva:

[65]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

[66]:
```
X_test = dados.drop(['ID_Cliente', 'Data', 'Compra_14_Dias'], axis=1)
y_test = dados['Compra_14_Dias']
```

[67]:
```
y_pred = modelo.predict(X_test)
y_prob = modelo.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
```

[68]:
```
plt.plot(fpr, tpr, color='darkorange', lw=2, label='curva ROC (área = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taxa de Falsos Positivos')
plt.ylabel('Taxa de Verdadeiros Positivos')
plt.title('ROC - Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Após a aplicação do SMOTE, a área da curva ROC, indicada no gráfico acima, foi de 0.94, indicando que o modelo teve um desempenho bastante satisfatório, sendo capaz distinguir muito bem entre as classes positivas e negativas, tendo uma alta taxa de assertividade. Abaixo, seguem as tabelas pré e pós-aplicação da SMOTE:

1.0.2 Pre-SMOTE

	Precision	Recall	F1-Score	Support
False	0.88	1.00	0.93	4471
True	1.00	0.00	0.01	637
Accuracy			0.88	5108
Macro Average	0.94	0.5	0.47	5108
Weighted Average	0.98	0.88	0.82	5108

1.0.3 Post-SMOTE

	Precision	Recall	F1-Score	Support
False	0.94	0.85	0.90	4471
True	0.36	0.65	0.47	637

	Precision	Recall	F1-Score	Support
Accuracy			0.83	5108
Macro Average	0.66	0.74	0.68	5108
Weighted Average	0.87	0.83	0.84	5108

Para finalizar, carreguei o modelo salvo e apliquei ao conjunto de clientes que ainda não fizeram uma recompra. Daí, classifiquei-os de acordo com as probabilidades geradas pelo modelo. No final, "imprimi" a lista de clientes ordenada por probabilidade de recompra e, logo após, imprimi a lista dos 100 clientes com a maior probabilidade de recompra, segundo o modelo.

[69]:
```
with open('modelo_classificacao.pkl', 'rb') as arquivo_modelo:
    modelo_carregado = pickle.load(arquivo_modelo)
```

[70]:
```
probabilidades = modelo_carregado.predict_proba(X_test)[:, 1]

dados_probabilidades = pd.DataFrame({'ID_Cliente': dados.ID_Cliente,␣
 ↪'Probabilidade_Recompra': probabilidades})
dados_probabilidades = dados_probabilidades.
 ↪sort_values(by=['Probabilidade_Recompra'], ascending=False)

print(dados_probabilidades)
```

```
        ID_Cliente  Probabilidade_Recompra
15894   15976215729                     1.0
16195   15981930701                     1.0
12851   15857019832                     1.0
12866   15857201125                     1.0
16301   15982976984                     1.0
…             …                       …
8666    15722120544                     0.0
8667    15722120544                     0.0
8668    15722120544                     0.0
2993    14470402150                     0.0
0                 1                     0.0

[17026 rows x 2 columns]
```

[71]:
```
top_100_clientes = dados_probabilidades.head(100)['ID_Cliente'].tolist()
print(top_100_clientes)
```

```
[15976215729, 15981930701, 15857019832, 15857201125, 15982976984, 15858834205,
15858834205, 15985852626, 14473973818, 15705442956, 15861341966, 15982612981,
15982530555, 15982530555, 15694565514, 15694565514, 15694565514, 15694032200,
15982500938, 15982500938, 14498338163, 13726700091, 15982484472, 15982472862,
15687518208, 15985828514, 15713412521, 15983076880, 15983507525, 15983546127,
15835478387, 15742691980, 15983544017, 15983540734, 15983540734, 15836998075,
15984944794, 15983507525, 15984944794, 15984960606, 15983134450, 15723766475,
```

15722120544, 14470402150, 15983439028, 15983439028, 15983360457, 15844147401,
15983304514, 15985716853, 15848760965, 15687154615, 15687154615, 15986141393,
15889085338, 12268398905, 15984412428, 15982021748, 14606303926, 15986168814,
15986168814, 15986168814, 15982014637, 15986206508, 15986225887, 15889169723,
15878567533, 15661313510, 15982009906, 15986279572, 15893201633, 15661313510,
12268398905, 15981930701, 15893716214, 15895149927, 15981930701, 15982032558,
15982264225, 15686827448, 15877369743, 15982472862, 15682359584, 15982457524,
15874483694, 15982417350, 15982417350, 15986163384, 15982383117, 12268398905,
12268398905, 12268398905, 12268398905, 15878541029, 15982372389, 15982317346,
15986168814, 14574478912, 15982317346, 15982267458]