Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

Previsão do Nível de Satisfação dos Clientes do Santander com Machine Learning

Definindo o Problema de Negócio

A satisfação do Cliente é uma medida fundamental de sucesso. Clientes insatisfeitos cancelam seus serviços e raramente expressam sua insatisfação antes de sair. Clientes satisfeitos, por outro lado, se tornam defensores da marca!

O Banco Santander está pedindo para ajudá-los a identificar clientes insatisfeitos no início do relacionamento. Isso permitiria que o Santander adotasse medidas proativas para melhorar a experiência destes clientes antes que seja tarde demais.

Neste projeto de aprendizado de máquina, vamos trabalhar com centenas de recursos anônimos para prever se um cliente está satisfeito ou insatisfeito com sua experiência bancária.

O desafio é exatamente desconhecer a informação de cada variável e ao mesmo tempo termos uma quantidade enorme delas. Nosso objetivo é entregar uma lista de clientes satisfeitos e insatisfeitos para o tomador de decisão, com uma acurácia de 70% em nosso modelo, buscando ter um modelo probabilístico mais simples e generalizável possível, facilitando assim para o tomador de decisão, utilizar seus resultados com boa clareza de entendimento.

Utilizaremos a linguagem Python e um dataset disponível no Kaggle, pelo endereço:

https://www.kaggle.com/c/santander-customer-satisfaction

Note que os dados estão em 2 arquivos separados, train.csv e test.csv. Apenas o arquivo train.csv possui variável resposta, então vamos trabalhar durante o processo de construção do modelo preditivo somente com ele.

Usaremos o arquivo test.csv para realizar as previsões do melhor modelo encontrado e entregar ao tomador de decisão.

Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

Pacotes e Versões

```
# Imports
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTE
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category = FutureWarning)

# Versões dos Pacotes
%reload_ext watermark
%watermark --iversions
```

```
pandas    : 1.5.2
numpy     : 1.23.5
matplotlib: 3.6.2
sklearn   : 1.0.2
seaborn   : 0.12.2
```

Carregando os Datasets

Como explicamos na definição do problema, vamos trabalhar inicialmente somente com o arquivo train.csv.

```
# Carregando os dados em um Dataframe do Pandas
df = pd.read_csv('Dados/train.csv')
```

Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

```
# Visualizando os dados
df
```

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_
0	1	2	23	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	
3	8	2	37	0.0	195.0	195.0	
4	10	2	39	0.0	0.0	0.0	
...	...	...	...	...	...	...	
76015	151829	2	48	0.0	0.0	0.0	
76016	151830	2	39	0.0	0.0	0.0	
76017	151835	2	23	0.0	0.0	0.0	
76018	151836	2	25	0.0	0.0	0.0	
76019	151838	2	46	0.0	0.0	0.0	

76020 rows × 371 columns

```
# Informações Gerais dos Dados
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76020 entries, 0 to 76019
Columns: 371 entries, ID to TARGET
dtypes: float64(111), int64(260)
memory usage: 215.2 MB
```

Limpeza e Análise Exploratória

Olhando inicialmente para nossos dados, verificamos que possuímos um número imenso de variáveis sem descrição (370), ou seja, não temos como entender cada variável e analisar seus comportamentos, da forma que estão.

Portanto inicialmente vamos verificar se temos dados faltantes (NaN), aplicando uma técnica para solucionar este problema. Logo depois vamos realizar 2 alternativas para uma redução de

dimensionalidade, Seleção de Variáveis mais Importantes e Vetorização de Variáveis com PCA, para então criarmos os modelos e compará-los.

```
# Identificando dados nulos ou NaN
pd.isnull(df).sum()
```

```
ID                              0
var3                            0
var15                           0
imp_ent_var16_ult1              0
imp_op_var39_comer_ult1         0
                               ..
saldo_medio_var44_hace3         0
saldo_medio_var44_ult1          0
saldo_medio_var44_ult3          0
var38                           0
TARGET                          0
Length: 371, dtype: int64
```

Não possuímos dados nulos, nem mesmo NaN em nosso dataset de trabalho. Vamos renomear as colunas para facilitar nossas visualizações.

```
# Função para Renomear as Variáveis
def ajusta_dados(df):

    # Reserva o Nome das Novas Colunas e Velhas Colunas
    NewCols = []
    for i in range(0, len(df.columns)):
        col = "Var" + str(i)
        NewCols.append(col)
    NewCols[0] = 'ID'
    if NewCols[-1] == 'TARGET':
        NewCols[-1] = 'RESP'
    else:
        pass
    OldCols = list(df.columns)

    # Aplica o De-Para
    DePara = {}
    for i,j in zip(OldCols, NewCols):
        DePara[i] = j
    df.rename(columns = DePara, inplace = True)

    return df
```

Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

```
# Aplica a Função ajusta_dados()
df = ajusta_dados(df=df)
df
```

	ID	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	...	Var361	Var362	Var363	Var364	Va
0	1	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
3	8	2	37	0.0	195.0	195.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
4	10	2	39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
76015	151829	2	48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
76016	151830	2	39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
76017	151835	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
76018	151836	2	25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
76019	151838	2	46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

76020 rows × 371 columns

Para continuar com nossos trabalhos, vamos eliminar a varável 'ID' em uma nova versão do nosso Dataset, pois a mesma não possui nenhum ganho de informação estatística.

```
# Vamos eliminar as variáveis RESP e ID em um novo Dataset
df1 = df.drop(['ID'], axis =1)
df1
```

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	...	Var361	Var362	Var363	Var364
0	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
1	2	34	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
3	2	37	0.0	195.0	195.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
4	2	39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
76015	2	48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
76016	2	39	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
76017	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
76018	2	25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
76019	2	46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

76020 rows × 370 columns

Nosso Dataset está organizado e limpo, como temos um número muito grande de variáveis e não sabemos a informação que cada uma representa ao problema de negócio, fica inviável estudar a correlação entre as variáveis ou mesmo analisar as estatísticas centrais e de dispersão. Nosso trabalho agora é iniciar uma redução de dimensionalidade antes de aplicarmos modelagem preditiva.

Para finalizarmos, vamos contar a variável target e identificar como ela está distribuída.

```
df1['RESP'].value_counts()
```

```
0    73012
1     3008
Name: RESP, dtype: int64
```

Percebemos que temos muitas amostras do tipo 0 (clientes contentes) e pouquíssimas amostras do tipo 1 (Clientes descontentes). É muito comum acontecer esse tipo de desbalanceamento em problemas de negócio como fraudes ou análise de sentimentos.

Vamos trabalhar esse tema posteriormente.

Estratégia de Trabalho

Como vimos temos alguns problemas aqui:

- Variáveis com Escalas Diferentes
- Variável TARGET desbalanceada
- Muitas variáveis, 369 para ser específico.

Vamos criar 2 Cenários para este projeto:

- Cenário 1 -> Inicialmente vamos trabalhar com a variável RESP, nossa variável resposta, desbalanceada. Vamos utilizar o Métodos de redução de dimensionalidade por Importância de Variáveis e Vetorização por PCA. Aplicaremos em algoritmos diferentes e por fim, avaliaremos as métricas e definiremos o melhor modelo.
- Cenário 2 -> Aplicamos uma técnica de Oversampling a Variável RESP com SMOTE, balanceando o Dataset. Usamos então as mesmas técnicas de redução de dimensionalidade do Cenário 1 e aplicamos aos mesmos algoritmos.

No final, comparamos qual o melhor modelo entre os cenários desenvolvidos.

Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

Cenário 1 - Dataset Desbalanceado + Técnicas de Redução de Dimensionalidade

Inicialmente vamos dividir os dados em Treino e Teste. Usaremos o pacote Sklearn.

```
# Separando os dados em Treino e Teste
Xtreino, Xteste, Ytreino, Yteste = train_test_split(df1.iloc[:, 0:369],
                                                    df1.iloc[:, -1],
                                                    test_size = 0.3,
                                                    random_state = 0)
```

Separado os Datasets, vamos aplicar uma normalização nos dados de treino e teste para igualar as escalas das variáveis.

```
# Modelo para Normalização dos Dados
scaler = preprocessing.StandardScaler().fit(Xtreino)
```

```
# Normalizando os dados de Treino
XtreinoS = scaler.transform(Xtreino)
XtreinoS
```

```
array([[ 3.99807940e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -4.84081375e-04],
       [ 3.99807940e-02,  1.12995969e+00, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02,  1.84784599e+00],
       [ 3.99807940e-02,  5.15816600e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -4.20753830e-01],
       ...,
       [ 4.01560803e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -2.66835804e-01],
       [ 3.99807940e-02, -2.51862256e-01, -1.50190114e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -2.81950539e-01],
       [ 3.99807940e-02,  1.66733488e+00, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -2.28898997e-01]])
```

```
# Normalizando os dados de Teste
XtesteS = scaler.transform(Xteste)
XtesteS
```

```
array([[ 3.99807940e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -1.24801003e-01],
       [ 3.99807940e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02,  1.36030923e-02],
       [ 3.99807940e-02,  8.22888143e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -4.84081375e-04],
       ...,
       [ 3.99807940e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02,  2.12808818e-01],
       [ 3.99807940e-02, -1.75094370e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02,  1.76104562e-01],
       [ 3.99807940e-02, -7.89237455e-01, -5.24513963e-02, ...,
        -1.91798836e-02, -2.00718862e-02, -4.15936973e-01]])
```

Modelo 00 - Regressão Logística + Variáveis Mais Importantes (SelectFromModel - Sklearn)

Datasets com escalas equivalentes, podemos aplicar nosso primeiro método de redução de dimensionalidade. Vamos aplicar um Feature Selection, baseado em um modelo de Regressão Logística, utilizando SelectFromModel do pacote sklearn.

```
# Modelo para Seleção de Variáveis utilizando Regressão Logística e SelectFromModel do pacote Sklearn
LR = LogisticRegression('l2', random_state = 0, max_iter = 10000).fit(XtreinoS, Ytreino)
model = SelectFromModel(LR, prefit=True)
```

```
# Aplicando Redução de Dimensionalidade aos dados de Treino
Xtreino00 = model.transform(XtreinoS)
Xtreino00.shape
```

```
(53214, 105)
```

```
# Aplicando Redução de Dimensionalidade aos dados de Teste
Xteste00 = model.transform(XtesteS)
Xteste00.shape
```

```
(22806, 105)
```

Chegamos a 105 variáveis importantes para a classificação de nossos Clientes. Apesar de termos ainda uma quantidade muito grande de variáveis, vamos aplicar a um modelo preditivo e calcular as métricas.

```
# Modelo Preditivo 00 - Regressão Logística
modelo00 = LogisticRegression(random_state=0, max_iter=10000).fit(Xtreino00, Ytreino)
modelo00
```

Realizando as Previsões com o Modelo Treinado e Avaliando a performance deste modelo.

```
# Aplicando Modelo aos Dados de Teste e Calculando as Métricas
prev00 = modelo00.predict(Xteste00)
Acc00 = accuracy_score(Yteste, prev00)
Prec00 = precision_score(Yteste, prev00, average='weighted')
AUC00 = roc_auc_score(Yteste, prev00)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc00)
print('A Precisão do Modelo é ', Prec00)
print('A área sob a Curva ROC do Modelo é ', AUC00)
```

```
A Acurácia do Modelo é  0.9593966500043848
A Precisão do Modelo é  0.9214315812010174
A área sob a Curva ROC do Modelo é  0.4997259272793715
```

Vamos alocar os resultados em uma tabela para compararmos depois.

```
# Colocando as Métricas em um Dataframe de Resultado
GLM00 = pd.Series(data=[Acc00, Prec00, AUC00], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado = pd.DataFrame(data={'GLM00': GLM00})
Resultado
```

	GLM00
Acurácia	0.959397
Precisão	0.921432
ROC AUC	0.499726

Modelo 01 - Regressão Logística + PCA

```
# Aplicando Redução de Dimensionalidade com Vetorização para os dados de Treino
Xtreino01 = PCA(n_components = 100).fit_transform(XtreinoS)
Xtreino01.shape
```

```
(53214, 100)
```

Vamos trabalhar com 100 variáveis para este Modelo e avaliar como performa.

```
# Modelo Preditivo 01 - Regressão Logística
modelo01 = LogisticRegression(random_state=0, max_iter=10000).fit(Xtreino01, Ytreino)
modelo01
```

```
LogisticRegression(max_iter=10000, random_state=0)
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev01 = modelo01.predict(Xteste01)
Acc01 = accuracy_score(Yteste, prev01)
Prec01 = precision_score(Yteste, prev01, average='weighted')
AUC01 = roc_auc_score(Yteste, prev01)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc01)
print('A Precisão do Modelo é ', Prec01)
print('A área sob a Curva ROC do Modelo é ', AUC01)
```

```
A Acurácia do Modelo é  0.040164868894150 66
A Precisão do Modelo é  0.961529147963921
A área sob a Curva ROC do Modelo é  0.5000456787867714
```

```
# Salvando os resultados na Tabela Resultado
GLM01 = pd.Series(data=[Acc01, Prec01, AUC01], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['GLM01'] = pd.DataFrame(GLM01)
Resultado
```

	GLM00	GLM01
Acurácia	0.959397	0.040165
Precisão	0.921432	0.961529
ROC AUC	0.499726	0.500046

Modelo 02 - Stochastic Gradient Descent (SGD) + SelectFromModel

Para este modelo vamos utilizar o Stochastic Gradient Descent e a Seleção de Variáveis que fizemos com SelectFromModel do Sklearn.

```
# Modelo 02 - Stochastic Gradient Descent
modelo02 = SGDClassifier().fit(Xtreino00, Ytreino)
modelo02
```

```
SGDClassifier()
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev02 = modelo02.predict(Xteste00)
Acc02 = accuracy_score(Yteste, prev02)
Prec02 = precision_score(Yteste, prev02, average='weighted')
AUC02 = roc_auc_score(Yteste, prev02)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc02)
print('A Precisão do Modelo é ', Prec02)
print('A área sob a Curva ROC do Modelo é ', AUC02)
```

```
A Acurácia do Modelo é  0.95970358677541
A Precisão do Modelo é  0.9214433981705747
A área sob a Curva ROC do Modelo é  0.49988580303307145
```

```
# Salvando os resultados na Tabela Resultado
SGD02 = pd.Series(data=[Acc02, Prec02, AUC02], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['SGD02'] = SGD02
Resultado
```

	GLM00	GLM01	SGD02
Acurácia	0.959397	0.040165	0.959704
Precisão	0.921432	0.961529	0.921443
ROC AUC	0.499726	0.500046	0.499886

Modelo 03 - Stochastic Gradient Descent (SGD) + PCA

Neste modelo utilizaremos o algoritmo Stochastic Gradient Descent e a redução de dimensionalidade com PCA.

```
# Modelo 03 - Stochastic Gradient Descent
modelo03 = SGDClassifier().fit(Xtreino01, Ytreino)
modelo03
```

```
SGDClassifier()
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev03 = modelo03.predict(Xteste01)
Acc03 = accuracy_score(Yteste, prev03)
Prec03 = precision_score(Yteste, prev03, average='weighted')
AUC03 = roc_auc_score(Yteste, prev03)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc03)
print('A Precisão do Modelo é ', Prec03)
print('A área sob a Curva ROC do Modelo é ', AUC03)
```

```
A Acurácia do Modelo é  0.04016486889415066
A Precisão do Modelo é  0.961529147963921
A área sob a Curva ROC do Modelo é  0.5000456787867714
```

```
# Salvando os resultados na Tabela Resultado
SGD03 = pd.Series(data=[Acc03, Prec03, AUC03], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['SGD03'] = SGD03
Resultado
```

Cenário 2 - Balanceamento do Dataset + Técnicas de Redução de Dimensionalidade

Para este cenário, vamos repetir os algoritmos utilizados, porém vamos aplicar uma técnica para balancear a variável alvo utilizando dados sintéticos, denominada Oversampling.

```
# Criando um modelo de Oversampling
sm = SMOTE(random_state=0)
```

```
# Balanceando os dados de Treino
XtreinoB, YtreinoB = sm.fit_resample(XtreinoS, Ytreino)
YtreinoB.value_counts()
```

```
0    51120
1    51120
Name: RESP, dtype: int64
```

Modelo 04 - Regressão Logística + SelectFromModel

Agora vamos repetir os algoritmos e técnicas utilizadas anteriormente, porém com nosso dataset balanceado.

```
# Modelo de Redução de Dimensionalidade com SelectFromModel()
LROS = LogisticRegression('l2', random_state = 0, max_iter = 10000).fit(XtreinoB, YtreinoB)
modelOS= SelectFromModel(LR, prefit=True)
```

```
# Aplicando Redução de Dimensionalidade aos Datasets de Treino e Teste
XtreinoB00 = modelOS.transform(XtreinoB)
XtesteB00 = modelOS.transform(XtesteS)
```

```
# Modelo 04 - Regressão Logística
modelo04 = LogisticRegression(random_state=0, max_iter=10000).fit(XtreinoB00, YtreinoB)
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev04 = modelo04.predict(XtesteB00)
Acc04 = accuracy_score(Yteste, prev04)
Prec04 = precision_score(Yteste, prev04, average='weighted')
AUC04 = roc_auc_score(Yteste, prev04)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc04)
print('A Precisão do Modelo é ', Prec04)
print('A área sob a Curva ROC do Modelo é ', AUC04)
```

```
A Acurácia do Modelo é  0.6973164956590371
A Precisão do Modelo é  0.9494225971764026
A área sob a Curva ROC do Modelo é  0.7222963655678303
```

```
# Salvando os resultados na Tabela Resultado
GLMB04 = pd.Series(data=[Acc04, Prec04, AUC04], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['GLMB04'] = GLMB04
Resultado
```

	GLM00	GLM01	SGD02	SGD03	GLMB04
Acurácia	0.959397	0.040165	0.959704	0.040165	0.697316
Precisão	0.921432	0.961529	0.921443	0.961529	0.949423
ROC AUC	0.499726	0.500046	0.499886	0.500046	0.722296

Modelo 05 - Regressão Logística + PCA

```
# Modelo de Redução de Dimensionalidade com PCA
PCA01 = PCA(n_components = 100)
```

```
# Aplicando Redução de Dimensionalidade aos dados de Treino e Teste
XtreinoB01 = PCA01.fit_transform(XtreinoB)
XtesteB01 = PCA01.fit_transform(XtesteS)
```

```
# Modelo 05 -Regressão Logística
modelo05 = LogisticRegression(random_state=0, max_iter=10000).fit(XtreinoB01, YtreinoB)
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev05 = modelo05.predict(XtesteB01)
Acc05 = accuracy_score(Yteste, prev05)
Prec05 = precision_score(Yteste, prev05, average='weighted')
AUC05 = roc_auc_score(Yteste, prev05)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc05)
print('A Precisão do Modelo é ', Prec05)
print('A área sob a Curva ROC do Modelo é ', AUC05)
```

```
A Acurácia do Modelo é  0.04012102078400421
A Precisão do Modelo é  0.9615290775267082
A área sob a Curva ROC do Modelo é  0.5000228393933857
```

```
# Salvando os resultados na Tabela Resultado
GLMB05 = pd.Series(data=[Acc05, Prec05, AUC05], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['GLMB05'] = GLMB05
Resultado
```

	GLM00	GLM01	SGD02	SGD03	GLMB04	GLMB05
Acurácia	0.959397	0.040165	0.959704	0.040165	0.697316	0.040121
Precisão	0.921432	0.961529	0.921443	0.961529	0.949423	0.961529
ROC AUC	0.499726	0.500046	0.499886	0.500046	0.722296	0.500023

Modelo 06 - Stochastic Gradient Descent (SGD) + SelectFromModel

```
# Modelo 06 - Stochastic Gradient Descent
modelo06 = SGDClassifier().fit(XtreinoB00, YtreinoB)
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev06 = modelo06.predict(XtesteB00)
Acc06 = accuracy_score(Yteste, prev06)
Prec06 = precision_score(Yteste, prev06, average='weighted')
AUC06 = roc_auc_score(Yteste, prev06)
```

```
# Visualiza as Metricas do Modelo
print('A Acurácia do Modelo é ', Acc06)
print('A Precisão do Modelo é ', Prec06)
print('A área sob a Curva ROC do Modelo é ', AUC06)
```

```
A Acurácia do Modelo é  0.7280540208717005
A Precisão do Modelo é  0.9489354547371738
A área sob a Curva ROC do Modelo é  0.7257258229278323
```

```
# Salvando os resultados na Tabela Resultado
SGDB06 = pd.Series(data=[Acc06, Prec06, AUC06], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['SGDB06'] = SGDB06
Resultado
```

	GLM00	GLM01	SGD02	SGD03	GLMB04	GLMB05	SGDB06
Acurácia	0.959397	0.040165	0.959704	0.040165	0.697316	0.040121	0.728054
Precisão	0.921432	0.961529	0.921443	0.961529	0.949423	0.961529	0.948935
ROC AUC	0.499726	0.500046	0.499886	0.500046	0.722296	0.500023	0.725726

Modelo 07 - Stochastic Gradient Descent (SGD) + PCA

```
# Modelo 08 - Stochastic Gradient Descent
modelo07 = SGDClassifier().fit(XtreinoB01, YtreinoB)
```

```
# Aplicando Modelo Preditivo e Calculando as Métricas
prev07 = modelo07.predict(XtesteB01)
Acc07 = accuracy_score(Yteste, prev07)
Prec07 = precision_score(Yteste, prev07, average='weighted')
AUC07 = roc_auc_score(Yteste, prev07)
```

```
# Visualizando as Métricas do Modelo
print('A Acurácia do Modelo é ', Acc07)
print('A Precisão do Modelo é ', Prec07)
print('A área sob a Curva ROC do Modelo é ', AUC07)
```

```
A Acurácia do Modelo é  0.9598351311058494
A Precisão do Modelo é  0.9214484603652541
A área sob a Curva ROC do Modelo é  0.4999543212132286
```

```
# Salvando os resultados na Tabela Resultados
SGDB07 = pd.Series(data=[Acc07, Prec07, AUC07], index=['Acurácia', 'Precisão', 'ROC AUC'])
Resultado['SGDB07'] = SGDB07
Resultado
```

	GLM00	GLM01	SGD02	SGD03	GLMB04	GLMB05	SGDB06	SGDB07
Acurácia	0.959397	0.040165	0.959704	0.040165	0.697316	0.040121	0.728054	0.959835
Precisão	0.921432	0.961529	0.921443	0.961529	0.949423	0.961529	0.948935	0.921448
ROC AUC	0.499726	0.500046	0.499886	0.500046	0.722296	0.500023	0.725726	0.499954

Conclusão e Entrega das Previsões

Analisando a tabela de resultado dos modelos criados, percebemos que o Modelo de SGD, utilizando o dataset balanceado com a técnica de oversampling e aplicando a técnica de redução de dimensionalidade do Sklearn (SelectFromModel), obteve-se a performance mais equilibrada entre as métricas. Sendo Acurácia de 72,8%, Precisão de 94,89% e ROC AUC de 72,57%.

```
# Modelo de Performance mais Equilibrada
Resultado['SGDB06']
```

```
Acurácia    0.728054
Precisão    0.948935
ROC AUC     0.725726
Name: SGDB06, dtype: float64
```

```
# Carregando Novos Dados
ND = pd.read_csv('Dados/test.csv')
ND.shape
```

```
(75818, 370)
```

Elaborado por Thiago Bulgarelli

Contato: bugath36@gmail.com

```
# Ajustanto os Novos Dados
ND1 = ajusta_dados(ND)
ND1 = ND1.drop('ID', axis=1)
ND1
```

	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	...	Var360	Var361	Var362	Var363	Var36
0	2	32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
1	2	35	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
2	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
3	2	24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
4	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
75813	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
75814	2	26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
75815	2	24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
75816	2	40	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.
75817	2	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.

75818 rows × 369 columns

```
# Normalizando os Novos Dados com Scaler()
NDNorm = scaler.transform(ND1)

# Aplicando Redução de Dimensionalidade com SelectFromModel()
NDNormR = modelOS.transform(NDNorm)
```

```
# Previsão de Classificação com o Modelo de Melhor Performance SGD06
PrevisaoFinal = modelo06.predict(NDNormR)
```

```
# Preparando Tabela Final
data = {'ID Cliente':ND['ID'], 'Status Felicidade': PrevisaoFinal}
TabelaFinal = pd.DataFrame(data=data)
TabelaFinal.head()
```

	ID Cliente	Status Felicidade
0	2	0
1	5	0
2	6	0
3	7	0
4	9	0

```
# Salando em Disco em extensão .CSV
TabelaFinal.to_csv('dados/TabelaFinal.csv', sep=',')
```

Finalizamos nosso trabalho, entregando a tabela PrevisaoFinal para o Cliente, com as previsões dos Exemplos Solicitados.