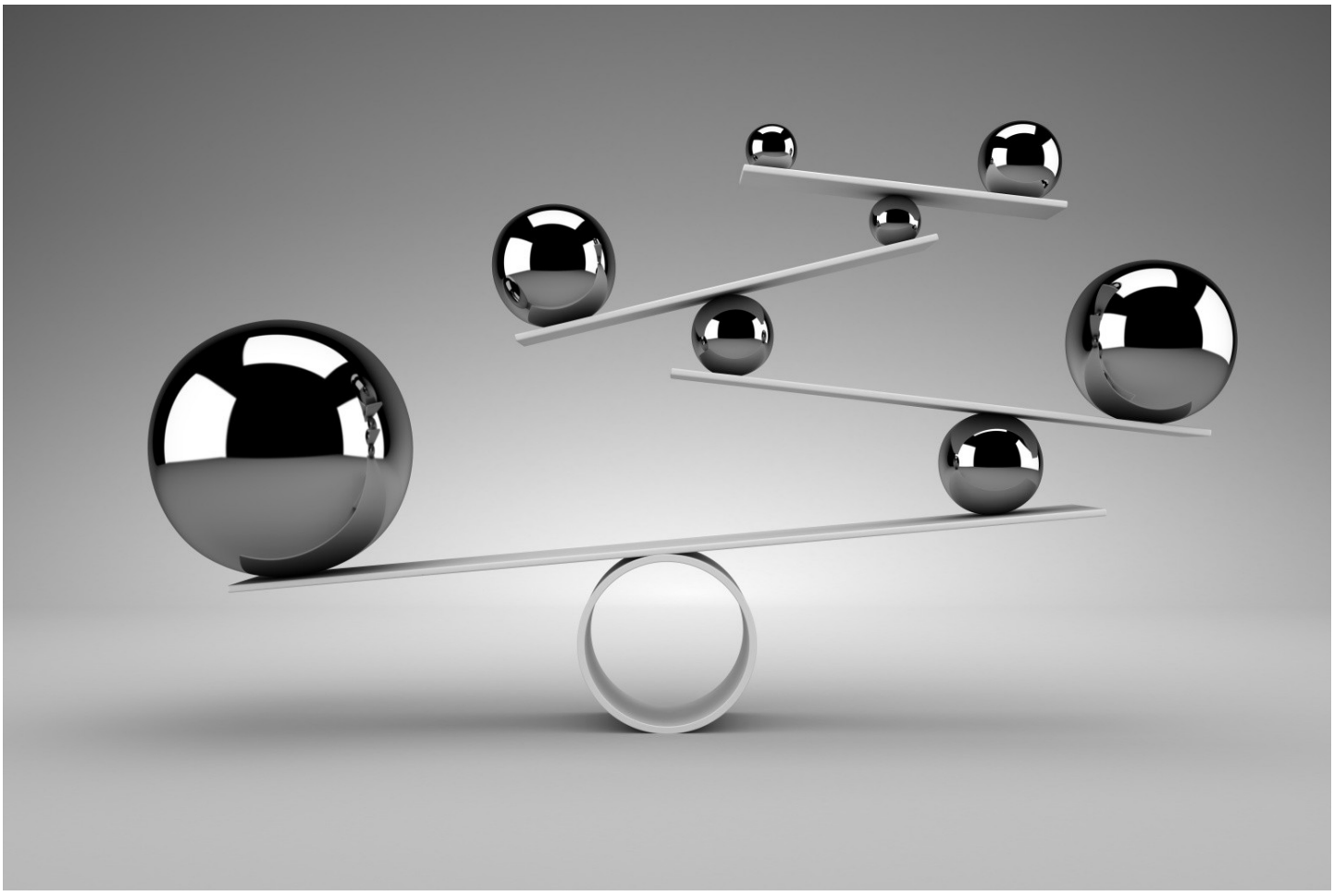
Manual básico para balanceamento de classes em datasets

Dados desbalanceados podem criar modelos altamente enviesados, muitas vezes inúteis, e, ainda assim, com uma acurácia extremamente alta



Conceito

Modelos de machine learning, como já dissemos anteriormente, aprendem com os dados, sem serem explicitamente programados. Dito isto, se os datasets estiverem desbalanceados e não aplicarmos nenhuma técnica para corrigir isto, podemos estar gerando modelos altamente enviesados ou até mesmo, modelos completamente inúteis. Ok, mas quais as possíveis consequências disso?

Paradoxo da acurácia

Imagine que você está criando um modelo de prevenção de fraudes. Você faz todas as transformações necessárias, treina o modelo e quando vai para as métricas, 98% de acurácia!!! Parece um excelente modelo, certo? Calma, não comemore ainda. Quando o modelo entra em contato com novos dados ele é incapaz de detectar fraudes, mas acerta sempre que há transações verdadeiras. O que está acontecendo?

Olhando para o dataset você percebe que 98% das transações são verdadeiras e apenas 2% são fraudes. Então o modelo decidiu que esta classe minoritária não tem importância e decidiu tratá-la como ruído. Durante o treino, visando um acerto total maior, o modelo decidiu ignorar a classe minoritária.

Dessa forma, o modelo acerta 98% das vezes, mas é incapaz de detectar qualquer fraude.

Dados no mundo real que possuem comportamento desbalanceado

Quando nos deparamos com problemas de classificação que possuem classes minoritárias e majoritárias, costumamos chamar as classes com menos observações de raras. **Embora raras para o dataset, problemas com classes desbalanceadas são muito frequentes. Mais do que isso, geralmente as observações raras são justamente o que desejamos prever ou ressaltar.** Vejamos mais alguns exemplos de dados desbalanceados:

- Emails verdadeiros x Spams
- Pessoas saudáveis x Pessoas doentes (câncer, diabetes, cardíacas)
- Produtos com qualidade x Produtos defeituosos
- Funcionamento normal x Falhas de funcionamento
- Usuários de uma plataforma x Usuários que abandonam (churn)
- Clientes sem sinistro x Clientes com sinistro

Diferentes graus de desbalanceamento

Mesmo quando falamos de dados que precisam de balanceamento, eles podem variar de desbalanceamentos extremos ao moderado. Vale lembrar que não existe uma regra específica de que para estar balanceado o dataset precisa estar com as classes a serem previstas balanceadas em 50%, como cientista de dados você já deve saber a resposta: DEPENDE! E cabe a você responder a esta pergunta através de experimentos com

diferentes proporções, por exemplo. Mas para fins de referência, podemos adotar as seguintes medidas para avaliar o quão desbalanceados estão seus dados.

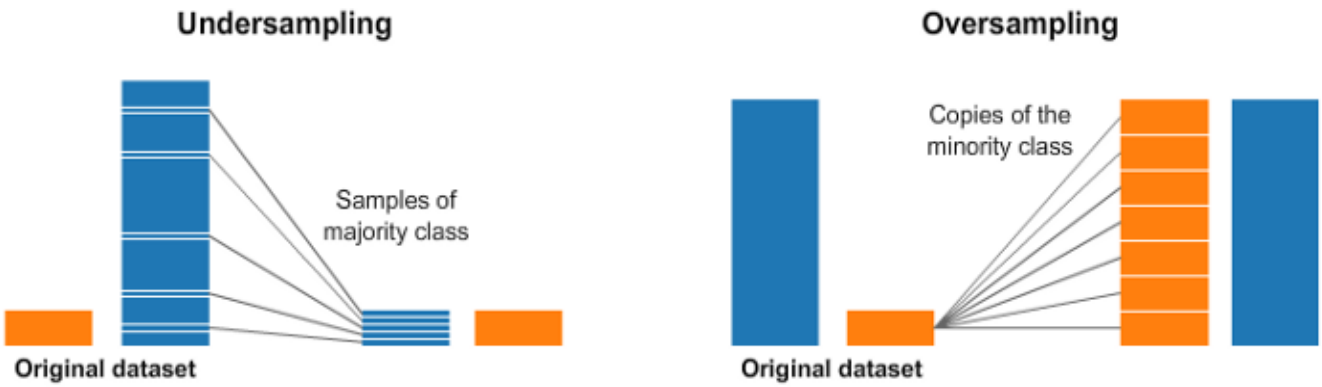
Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

Graus de desbalanceamento: https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data

Estratégias e implementações de balanceamento

1. **Enriquecimento de dados**
   Colher mais amostras ou enriquecer a base de dados com novos dados para a classe minoritária. Essa estratégia, embora nem sempre disponível, é a que oferece uma melhor resultados pois adicionaria informações novas e reais aos dados.

2. **Resampling ou reamostragem**
   Frequentemente, enriquecer a base ou colher novas amostras não é uma opção viável. Nesses casos podemos criar uma nova versão balanceada do dataset. Em linhas gerais, existem duas estratégias de resampling:



fonte: https://raw.githubusercontent.com/rafjaa/machine_learning_fecib/master/src/static/img/resampling.png

- **Undersampling:** consiste em remover observações da classe majoritária, preservando a amostra da classe minoritária do dataset original. Dessa forma, obtemos um dataset com menos observações, porém equilibrado.
Cuidado: Ao adotar essa estratégia você poderá estar abrindo mão de informações

importantes à respeito da classe majoritária e eventualmente não terá informações suficientes para a classe minoritária também. Ou seja, a menos que seja um dataset muito grande, raramente se usa o undersampling porque ele pode levar o modelo a um underfitting.

- Oversampling: nessa estratégia, vamos adicionar mais exemplos da classe minoritária para equipará-la à classe majoritária.
Cuidado: Ao fazer um oversample, podemos estar praticamente dobrando o tamanho do dataset e como vamos adicionar novos exemplos da classe minoritária, podemos acabar overfitando esta classe no modelo.

Implementações

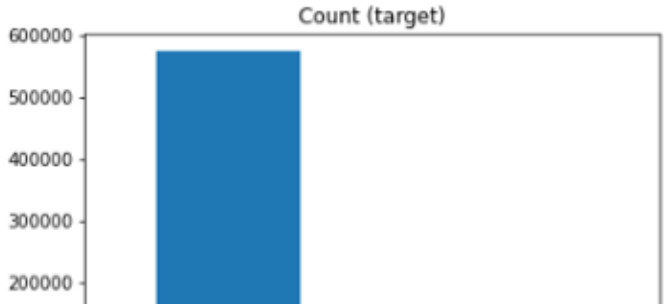
Vamos utilizar o dataset da competição da Porto Seguro no Kaggle e trazer algumas técnicas comparando os resultados a partir de um modelo base para facilitar um entendimento e suas comparações.

O desbalanceamento entre as classes

No dataset as classes 1 e 0 estão na coluna target e são referentes a probabilidade de um segurado ter ou não um sinistro no próximo ano.
Importando os dados percebemos que a classe 1 corresponde à aproximadamente 4% do dataset.

```
import numpy as np
import pandas as pd

df_train = pd.read_csv('../input/train.csv')

target_count = df_train.target.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (target)');
```

```
Class 0: 573518
Class 1: 21694
Proportion: 26.44 : 1
```



Temos mais de 570k linhas contra um pouco mais de 21k. Estamos falando de um desbalanceamento moderado com 3.78% das observações na classe minoritária.

Paradoxo da acurácia neste dataset

Lembra que falamos sobre isso no exemplo de detecção de fraudes? Agora vamos mostrar como ele acontece na prática. Para isso, vamos usar um classificador e não vamos realizar nenhuma transformação ou análise no dataset. Primeiro vamos avaliar apenas a métrica `accuracy_score` .

Antes de prosseguir, precisamos falar sobre o XGBoost

Para fins didáticos vamos usar o XGBoost. Esse algorítimo é extremamente complexo e robusto. Em seu default ele vai realizar operações como regularização, estratégia para lidar com dados faltantes já vem preparado para computação paralela e cross-validation. Tudo isso para dizer que as métricas alcançadas por esse modelo são extremamente performáticas mas que falaremos mais sobre ele em materiais e aulas mais adiante.

Agora podemos prosseguir...

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Remove 'id' and 'target' columns
labels = df_train.columns[2:]

X = df_train[labels]
y = df_train['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42, stratify=y)

model = XGBClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 96.35%
```

accuracy_score usando todas as features do dataset

Se essa métrica é real, vamos conseguir piorá-la se ao invés do dataset inteiro, usarmos apenas uma feature.

```
model = XGBClassifier()
model.fit(X_train[['ps_calc_01']], y_train)
y_pred = model.predict(X_test[['ps_calc_01']])

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 96.36%
```

accuracy_score usando apenas uma feature

Ops! Ao invés de piorar, a performance melhorou? Precisamos investigar o que está acontecendo. Uma boa forma de fazer isso é através da Matrix de Confusão.

Investigando as predições do modelo com uma Matriz de Confusão

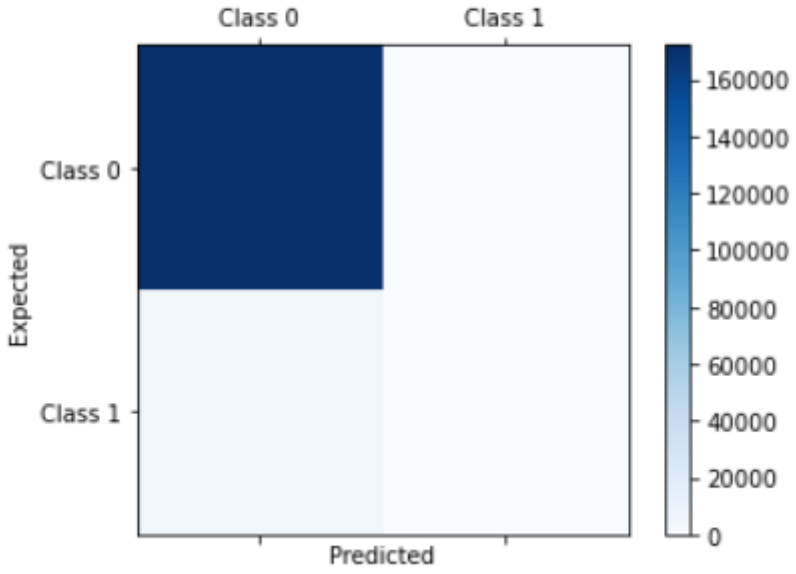
Uma forma interessante de avaliar os resultados é através da matriz de confusão que mostra os valores preditos e esperados ou reais.
Na primeira linha, a primeira coluna indica quantos “Classe 0” foram preditos corretamente (como “Classe 0”) Na primeira linha, segunda coluna os erroneamente classificados como “Classe 1”. Na segunda linha, a primeira coluna nos mostra quantos “Classe 1” foram preditos erroneamente (como “Classe 0”) Na segunda linha, e segunda coluna os corretamente classificados como “Classe1”

A diagonal descendente nos mostra as predições corretas. Modelos que acertam mais possuem concentração de valores maiores na primeira linha, primeira coluna e na segunda linha, segunda coluna.

```
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print('Confusion matrix:\n', conf_mat)

labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
```

```
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Confusion matrix:
 [[172056      0]
 [  6508      0]]
```



Conforme havíamos previsto, o modelo decidiu ignorar a classe minoritária, em nome de uma maior acurácia nas predições de uma maneira geral.

Sklearn Imblearn

Essa biblioteca reúne diversas técnicas das mais simples como Random Oversampler e Random Undersampler a integrações com Keras e Tensorflow para redes neurais. Vale a pena dar uma olhada na documentação.

```
#vamos importar os algoritmos a serem explorados

from sklearn.metrics import f1_score,accuracy_score
from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
# counter takes values returns value_counts dictionary
```

Random Undersampling e Oversampling

Observe que a implementação dessas duas técnicas por meio do imblearn do scikit learn é virtualmente a mesma, mudando apenas o nome do algoritmo a ser instanciado. Para fins didáticos, alterei o nome dos resamples dos datasets de acordo com o nome de cada algoritmo.

Random Undersampling

```
print('Original dataset shape %s' % Counter(y))

rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X, y)

print('Resampled dataset shape %s' % Counter(y_rus))
```

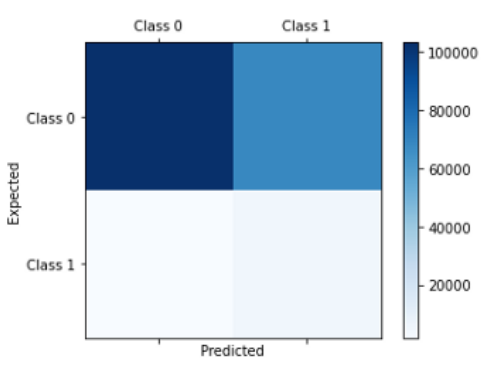
```
Original dataset shape Counter({0: 573518, 1: 21694})
Resampled dataset shape Counter({0: 21694, 1: 21694})
```

```
model = XGBClassifier()
model.fit(X_rus, y_rus)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 60.57%
```

```
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print('Confusion matrix:\n', conf_mat)

labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Confusion matrix:
 [[103291  68765]
 [  1643   4865]]
```



Random Oversampling

```
print('Original dataset shape %s' % Counter(y))
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_sample(X, y)

print(X_ros.shape[0] - X.shape[0], 'new random picked points')

print('Resampled dataset shape %s' % Counter(y_ros))
```

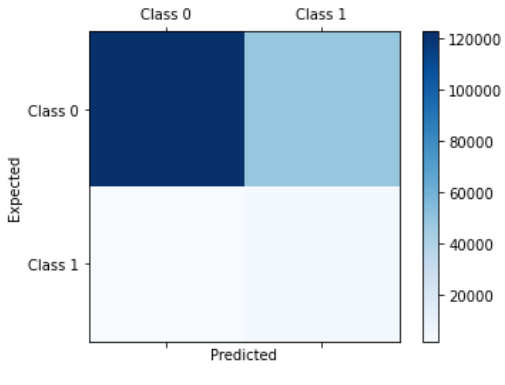
```
Original dataset shape Counter({0: 573518, 1: 21694})
551824 new random picked points
Resampled dataset shape Counter({0: 573518, 1: 573518})
```

```
model = XGBClassifier()
model.fit(X_ros, y_ros)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 71.35%
```

```
conf_mat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print('Confusion matrix:\n', conf_mat)

labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Confusion matrix:
 [[122787  49269]
 [  1884   4624]]
```

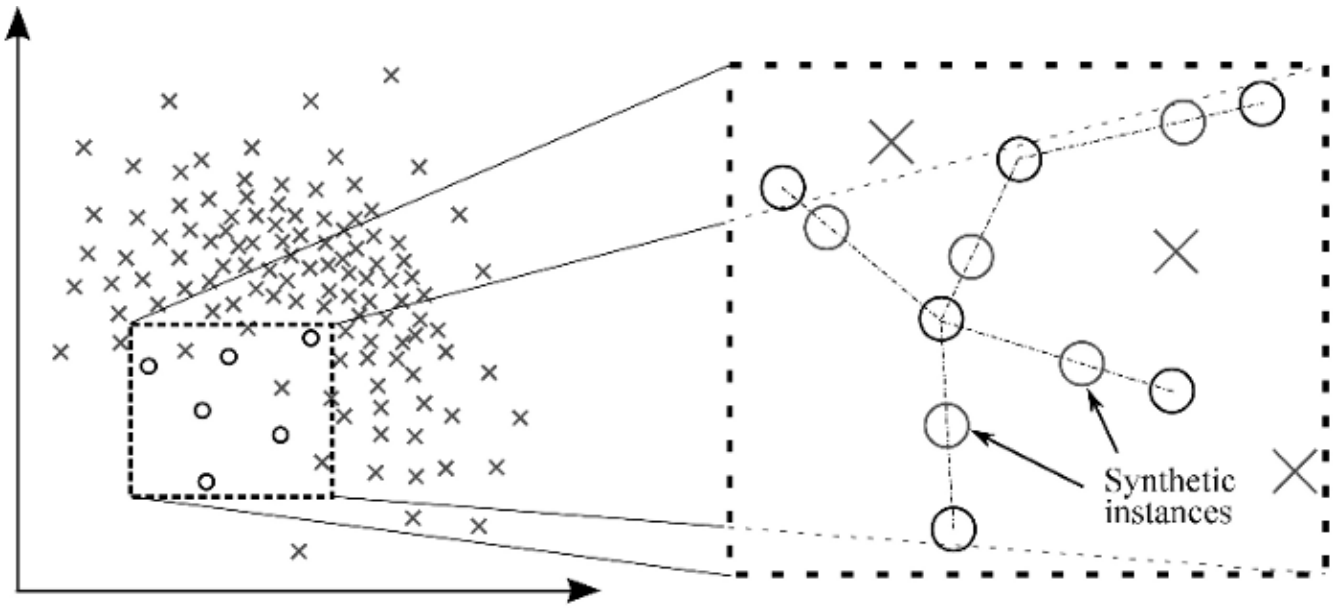


Comparação entre Random Undersampling (esquerda) e Random Oversampling (Direita).

Podemos notar que os dados que descartamos no Random Undersampling nos fez errar mais classes 0, mas as classes 1 corretamente preditas são bem próximas nos dois modelos. Devemos lembar que não realizamos se quer uma EDA nesses datasets prováveis datacleaning e transformações tornariam qualquer modelo muito mais funcional. Mas vamos em frente e testar técnicas de oversampling mais avançadas.

(SMOTE)

Synthetic Minority OverSampling Technique ou Técnica de Oversampling Sintética da Minoria (em uma tradução livre), consiste em sintetizar elementos da classe minoritária baseado nos elementos que já existem. Funciona de forma randômica selecionando aleatoriamente observações da classe minoritária e computando pontos através de um KNN. Os pontos sintéticos são adicionados entre os pontos escolhidos e seus vizinhos.



Dados sintéticos sendo gerados a partir dos dados da classe minoritária do dataset.
Fonte:https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/

Vamos à implementação:

```
print('Original dataset shape %s' % Counter(y))
smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_sample(X, y)
print('Resampled dataset shape %s' % Counter(y_sm))
```

```
Original dataset shape Counter({0: 573518, 1: 21694})
Resampled dataset shape Counter({0: 573518, 1: 573518})
```

```
model = XGBClassifier()
model.fit(X_sm, y_sm)
```

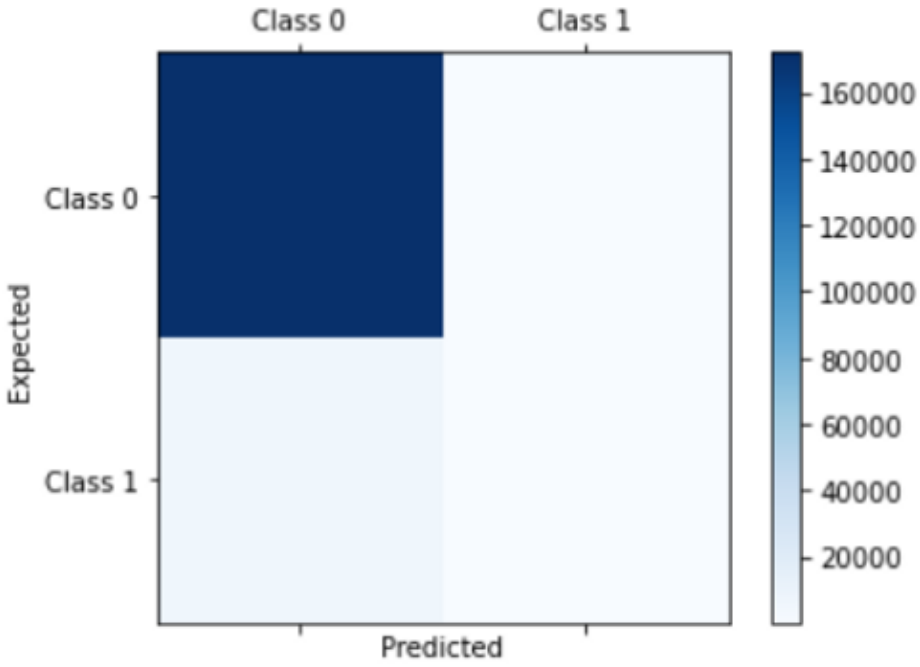
```
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 96.36%
```

Perceba o aumento do número de dados e o aumento da acurácia do modelo.

Temos uma melhora significativa no aumento da acurácia do modelo. Mas como será que estão sendo as predições quanto às classes? Vamos investigar com a matriz de confusão.

```
Confusion matrix:
 [[172048      8]
 [  6487     21]]
```



O modelo é realmente muito preciso em acertar a classe zero. Note que, ao contrário dos modelos Randomicos, praticamente não temos erros do tipo II. Mas aumentamos e muito os nossos erros do tipo I.

(ADASYN)

ADASYN (Adaptive Synthetic) também é um algoritmo que gera dados sintéticos. Sua maior vantagem é que ele tenta aprender prioritariamente com os dados mais “difíceis de aprender” da classe minoritária. Sua principal vantagem pode virar uma fraqueza caso os dados da classe minoritária sejam muito esparsos.

Vamos à sua implementação:

```
print('Original dataset shape %s' % Counter(y))

adasyn = ADASYN(random_state=42)
X_ada, y_ada = adasyn.fit_resample(X, y)

print('Resampled dataset shape %s' % Counter(y_ada))
```

```
Original dataset shape Counter({0: 573518, 1: 21694})
Resampled dataset shape Counter({1: 581126, 0: 573518})
```

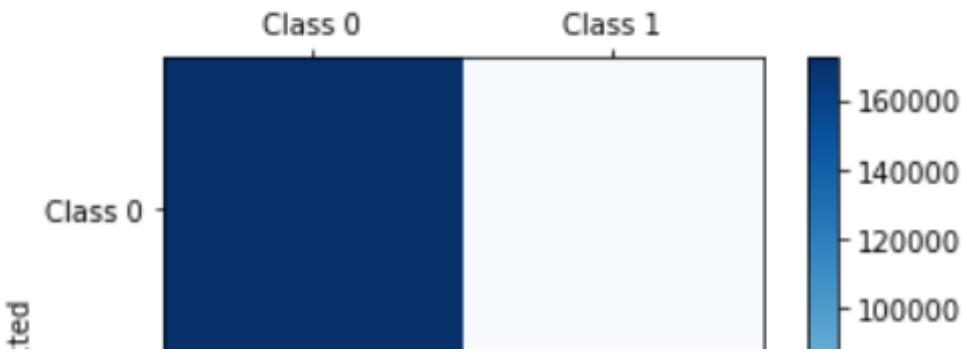
```
model = XGBClassifier()
model.fit(X_ada, y_ada)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

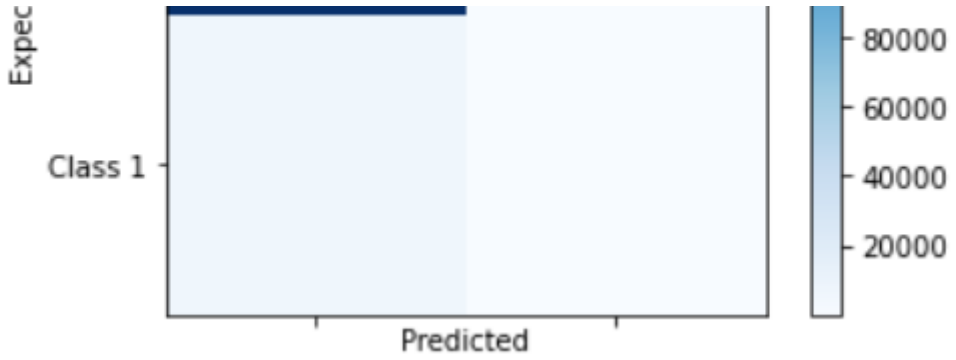
```
Accuracy: 96.37%
```

Essa técnica é muito similar ao SMOTE, mas adiciona erros aleatórios aos dados sintéticos. A literatura diz que o ADASYN performa melhor que o SMOTE principalmente em datasets com desbalanceamentos extremos.

Vamos à matriz de confusão para o ADASYN:

```
Confusion matrix:
 [[172050      6]
 [  6481     27]]
```





Realmente a performance dos dois algoritmos é muito similar para este dataset.

Conclusão:

Existem muitas outras técnicas e abordagens de balanceamento de datasets. Mais importante que conhecer todas elas em profundidade é saber identificar quando é necessário tomar alguma medida e como provar que a sua tese de que o dataset está desbalanceado é verdadeira. Fica ainda o lembrete de que em usos gerais as técnicas de oversampling vão performar melhor, mas ao duplicar os dados ou criar novos data points de forma sintética, alguns modelos em determinados datasets podem interpretar os novos dados como ruídos. Por isso a importância de testar as duas técnicas. Vale lembrar que em datasets muito grandes técnicas de undersampling serão melhores tanto computacionalmente quanto mais precisas, já que há dados ou informações suficientes para que o modelo possa aprender com as duas classes.

Te convido a explorar um notebook que criei no Kaggle, criar a sua própria versão e tentar aplicar outros conhecimentos que já tivemos até aqui como EDA e testes de hipótese por exemplo.

Embora este assunto não seja tão sexy e motivador, cientistas de dados não podem ignorar os conceitos e os perigos de um dataset desbalanceado.

Que bom que você chegou até aqui! Espero que tenhamos sido claro em todo o texto e nos exemplo. Estude um pouco mais sobre este assunto nos links e videos abaixo e crie seu próprio notebook no kaggle.
Se em algum momento tiver dúvidas, será um prazer para o time Tera te ajudar nessa jornada.

Nos vemos no próximo texto.

Para estudar mais:

Estudo de caso de Câncer de Mama

comparison

medium.com

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset - Machine Learning Mastery

Last Updated on Has this happened to you? You are working on your dataset. You create a classification model and get...

machinelearningmastery.com

SMOTE for Imbalanced Classification with Python - Machine Learning Mastery

Imbalanced classification involves developing predictive models on classification datasets that have a severe class...

machinelearningmastery.com

7 Techniques to Handle Imbalanced Data - KDnuggets

What have datasets in domains like, fraud detection in banking, real-time bidding in marketing or intrusion detection...

www.kdnuggets.com

How to Handle Imbalanced Classes in Machine Learning

Imbalanced classes put "accuracy" out of business. This is a surprisingly common problem in machine learning...

elitedatascience.com

Methods for Dealing with Imbalanced Data

Imbalanced classes are a common problem in machine learning classification where there are a disproportionate ratio of...

towardsdatascience.com

Handling Imbalanced Data in Classification Problems

Introduction

medium.com

Imbalanced Data

A classification data set with skewed class proportions is called imbalanced . Classes that make up a large proportion…

developers.google.com

Credit Fraud || Dealing with Imbalanced Datasets

Explore and run machine learning code with Kaggle Notebooks | Using data from Credit Card Fraud Detection

www.kaggle.com

The 5 Sampling Algorithms every Data Scientist need to know

Or at least should have heard about

towardsdatascience.com

An Introduction to ADASYN (with code!)

Introduction

to ADASYN (with code!) Introductionmedium.com

Imbalanced Data

Edit description

amueller.github.io

What Is Balance And Imbalance Dataset?

Techniques To Convert Imbalanced Dataset Into Balanced Dataset and their comparison