## # 3. Classificação e Métricas

Nesta trilha você vai aprender:

* O esquema geral de modelos de aprendizado com a biblioteca scikit-learn
* O que é o Dilema Viés-Variância e como empregar Conjuntos de Treinamento e Teste
* Outras Métricas de Classificação

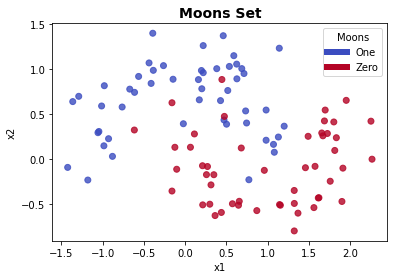
Vamos retomar os problemas de aprendizado supervisionado, de regressão e classificação, para entender de modo geral como podemos empregar esses modelos com a biblioteca scikit-learn. Você vai aprender também como podemos avaliar mais adequadamente o desempenho desses modelos empregando conjuntos separados de treinamento e teste, e verificando outras métricas importantes.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
from matplotlib.lines import Line2D  
import seaborn as sns

# Classificando *Moons* ou Empréstimos

Vamos começar classificando um conjunto aleatório de dados que denominaremos *Moons*. Cada um dos 100 pontos, *moons*, possui 2 atributos e recebe uma classe ou .

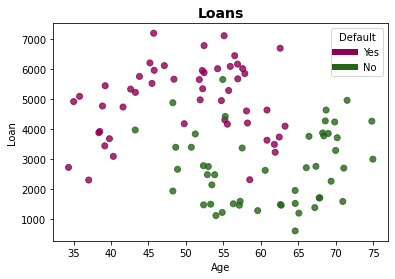
from sklearn.datasets import make\_moons  
cmap\_data = plt.cm.coolwarm   
  
X, y = make\_moons(n\_samples=100, noise=0.25, random\_state=1234)  
  
df = pd.DataFrame({'x1':X[:, 0], 'x2':X[:, 1], 'y':y})  
  
plt.scatter(df.x1, df.x2, c=df.y, cmap=cmap\_data, alpha=0.8)  
plt.title('Moons Set',weight='bold',fontsize=14)  
plt.xlabel("x1")  
plt.ylabel("x2")  
  
custom\_lines = [Line2D([0], [0], color=cmap\_data(0.), lw=6),  
 Line2D([0], [0], color=cmap\_data(1.), lw=6)]  
plt.legend(custom\_lines, ['One', 'Zero'], loc='upper right',title='Moons')  
  
plt.show()  
  
print(df.head())



x1 x2 y  
0 -0.763251 0.577354 0  
1 -0.391942 1.395641 0  
2 1.324561 -0.492319 1  
3 0.271295 -0.082532 1  
4 2.253887 0.420281 1

Talvez você considere esse problema um problema pouco interessante, afinal estamos apenas gerando dados aleatórios. Esse exemplo de *brinquedo*, entretanto, já traz os elementos essencias que precisamos saber para criar classificadores supervisionados e poderíamos igualmente ter uma base com dados de empréstimos, com idade e valor dos empréstimos tomados pelos clientes, classificados entre empréstimos pagos ou inadimplentes (*default*) para prever concessões de novos empréstimos.

from sklearn.datasets import make\_moons  
cmap\_data = plt.cm.PiYG  
  
X, y = make\_moons(n\_samples=100, noise=0.25, random\_state=1234)  
  
df\_loans = pd.DataFrame({'age':X[:, 0], 'loan':X[:, 1], 'default':y})  
  
df\_loans.age = df\_loans.age + 50 + df\_loans.age\*10  
df\_loans.loan = df\_loans.loan + 3000 + df\_loans.loan\*3000  
  
plt.scatter(df\_loans.age, df\_loans.loan, c=df\_loans.default, cmap=cmap\_data, alpha=0.8)  
plt.title('Loans',weight='bold',fontsize=14)  
plt.xlabel("Age")  
plt.ylabel("Loan")  
  
custom\_lines = [Line2D([0], [0], color=cmap\_data(0.), lw=6),  
 Line2D([0], [0], color=cmap\_data(1.), lw=6)]  
plt.legend(custom\_lines, ['Yes', 'No'], loc='upper right',title='Default')  
  
plt.show()  
  
print(df\_loans.head())



age loan default  
0 41.604235 4732.639984 0  
1 45.688635 7188.319998 0  
2 64.570169 1522.549822 1  
3 52.984240 2752.321406 1  
4 74.792752 4261.263330 1

Vamos então rever como podemos empregar o classificador logístico do scikit-learn entendo cada um dos passos que poderão, depois, ser empregados para outros algoritmos supervisionados.

# Aprendizado Supervisionado: Classificação com o SciKit-Learn

## Import dos Estimadores

Cada algoritmo de aprendizado na API do scikit-learn é exposto como objeto chamado de *estimador*. Por exemplo, você viu que para a regressão logística o estimador foi o from sklearn.linear\_model import LogisticRegression e há também um estimador regressão linear, sklearn.linear\_model.LinearRegression, embora tenhamos preferido empregar o modelo linear do pacote statsmodels na aula anterior.

from sklearn.linear\_model import LogisticRegression

## Instanciando e Definindo os Parâmetros

Os parâmetros do modelo podem ser definidos quando eles instanciados, por exemplo:

clf = LogisticRegression(max\_iter=1000)  
print(clf)

LogisticRegression(max\_iter=1000)

E empregamos o nome clf arbitrariamente apenas para designar nosso classificador.

## Ajustando os Parâmetros do Modelo

Os parâmetros do modelo são em seguida ajustados aos dados com a função fit() o que é feito em geral através de um otimizador (solver).

# clf.fit(df[['x1','x2']], df.y)  
# ou  
  
X = df[['x1','x2']]  
y = df.y  
  
clf.fit(X,y)  
  
print( clf.coef\_, clf.intercept\_)

[[ 1.21523438 -2.60803724]] [0.12484869]

Os valores coef\_ e clf.intercept\_ são os parâmetros estimados, o que poderá variar de acordo com o modelo aplicado.

## Predição

O método predict() fornece então a predição de dados com base nos parâmetros estimados do modelo.

# clf.predict(df[['x1','x2']])  
# ou   
clf.predict(X)

array([0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,  
 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,  
 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,  
 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0])

Ainda existem outros preditores úteis que retornam a probabilidade ou o log da probabilidade de cada classe, o que é mais útil em se tratando de modelos multiclasse.

clf.predict\_proba(X)[0:10]

array([[0.90957378, 0.09042622],  
 [0.98186015, 0.01813985],  
 [0.04659762, 0.95340238],  
 [0.33854573, 0.66145427],  
 [0.14582354, 0.85417646],  
 [0.14251362, 0.85748638],  
 [0.03145696, 0.96854304],  
 [0.91387591, 0.08612409],  
 [0.96036923, 0.03963077],  
 [0.87472292, 0.12527708]])

clf.predict\_log\_proba(X)[0:10]

array([[-0.09477916, -2.40322103],  
 [-0.01830639, -4.00964435],  
 [-3.06620578, -0.04771824],  
 [-1.08309611, -0.41331442],  
 [-1.925358 , -0.15761748],  
 [-1.94831768, -0.15374999],  
 [-3.45913493, -0.03196236],  
 [-0.09006048, -2.45196614],  
 [-0.04043746, -3.22814934],  
 [-0.13384811, -2.07722732]])

## Score do Modelo

Por último podemos medir a eficiência do nosso modelo, diretamente ou ainda a partir do score().

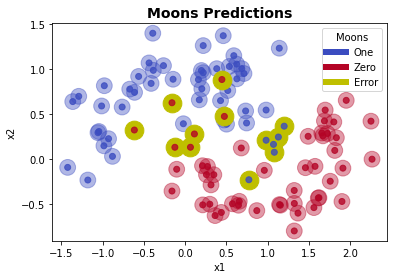
# clf.score(df[['x1','x2']], df.y)  
# ou  
  
clf.score(X,y)

0.87

Aqui o score , como você já viu, corresponde à acuracidade do modelo sobre o conjunto X. Você vai aprender mais sobre isso logo adiante.

Podemos ainda, no caso desse problema bastante simples, visualizar os erros e acertos do modelo o que, em geral, não será possível com dados com muitas dimensões e instâncias.

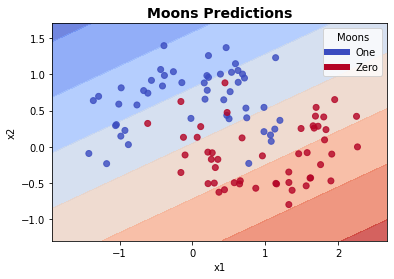
cmap\_data = plt.cm.coolwarm   
  
y\_pred = clf.predict(X)  
  
plt.scatter(X.x1, X.x2, c=y, cmap=cmap\_data, alpha=0.8)  
plt.scatter(X.x1, X.x2, c=y\_pred, s=250,cmap=cmap\_data, alpha=0.4)  
  
plt.scatter(X[y\_pred != y].x1, X[y\_pred != y].x2, color='y', s=350, alpha=1, label='wrong predictions')  
plt.scatter(X[y\_pred != y].x1, X[y\_pred != y].x2, c=y[y\_pred != y], cmap=cmap\_data, alpha=0.8)  
plt.title('Moons Predictions',weight='bold',fontsize=14)  
plt.xlabel("x1")  
plt.ylabel("x2")  
  
custom\_lines = [Line2D([0], [0], color=cmap\_data(0.), lw=6),  
 Line2D([0], [0], color=cmap\_data(1.), lw=6),  
 Line2D([0], [0], color='y', lw=6)]  
plt.legend(custom\_lines, ['One', 'Zero', 'Error'], loc='upper right',title='Moons')  
  
plt.show()



E podemos ainda verificar a *fronteira de decisão* do estimador.

x\_min, x\_max = X['x1'].min() - .5, X['x1'].max() + .5  
y\_min, y\_max = X['x2'].min() - .5, X['x2'].max() + .5  
xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.2),  
 np.arange(y\_min, y\_max, 0.2))  
  
if hasattr(clf, "decision\_function"):  
 Z = clf.decision\_function(np.c\_[xx.ravel(), yy.ravel()])  
else:  
 Z = clf.predict\_proba(np.c\_[xx.ravel(), yy.ravel()])[:, 1]  
  
Z = Z.reshape(xx.shape)  
plt.contourf(xx, yy, Z, cmap=cmap\_data, alpha=0.8)   
plt.scatter(X['x1'], X['x2'], c=y, cmap=cmap\_data, alpha=0.8)  
plt.title('Moons Predictions',weight='bold',fontsize=14)  
plt.xlabel("x1")  
plt.ylabel("x2")  
  
custom\_lines = [Line2D([0], [0], color=cmap\_data(0.), lw=6),  
 Line2D([0], [0], color=cmap\_data(1.), lw=6)]  
  
plt.legend(custom\_lines, ['One', 'Zero'], loc='upper right',title='Moons')  
  
plt.show()

/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names  
 "X does not have valid feature names, but"

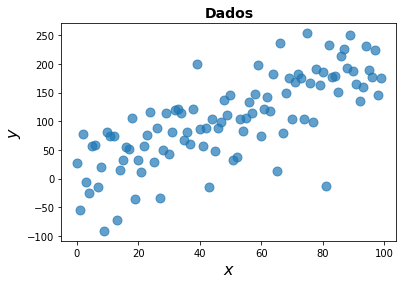


# Aprendizado Supervisionado: Regressão Linear com o SciKit-Learn

O exemplo abaixo mostra como empregar um estimador de regressão linear e, como você poderá notar, os passos são basicamente os mesmos:

1. Import do estimador
2. Preparação dos dados X e y do estimador (*variáveis preditoras e objetivo*)
3. Instanciação e configuração do estimador
4. Ajuste do modelo (*treinamento* ou estimativa dos parâmetros)
5. Predição
6. Obtenção de Métricas e Avaliação do Modelo

from scipy.stats import norm  
  
x = np.arange(0, 100)  
y = 2\*x + 3 + norm.rvs(loc=0, scale=50, size=100, random\_state=1234)  
  
plt.scatter(x=x,y=y,s=80,alpha=0.7)  
plt.title('Dados', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()  
  
print(df.head())



x1 x2 y  
0 -0.763251 0.577354 0  
1 -0.391942 1.395641 0  
2 1.324561 -0.492319 1  
3 0.271295 -0.082532 1  
4 2.253887 0.420281 1

## 

## 1. Import do estimador

from sklearn.linear\_model import LinearRegression

## 

## 2. Preparação dos dados X e y do estimador (*variáveis preditoras e objetivo*)

Essa etapa, dependendo dos dados, pode envolver diversas tarefas, como normalização, hot encode, separação de conjuntos de treinamento e teste que você verá mais adiante etc. e, normalmente antece a operação de instanciar o modelo pois os dados poderão ser aplicados a diferentes estimadores.

X = x.reshape(-1, 1)  
y = y.reshape(-1, 1)

## 

## 3. Instanciação e configuração do estimador

lm = LinearRegression()  
print(lm)

LinearRegression()

## 

## 4. Ajuste do modelo (*treinamento* ou estimativa dos parâmetros)

lm.fit(X,y)  
  
print(lm.coef\_, lm.intercept\_)

[[1.96520459]] [6.4779872]

## 5. Predição

lm.predict(X)[0:10]

array([[ 6.4779872 ],  
 [ 8.44319178],  
 [10.40839637],  
 [12.37360095],  
 [14.33880554],  
 [16.30401012],  
 [18.26921471],  
 [20.23441929],  
 [22.19962388],  
 [24.16482846]])

## 

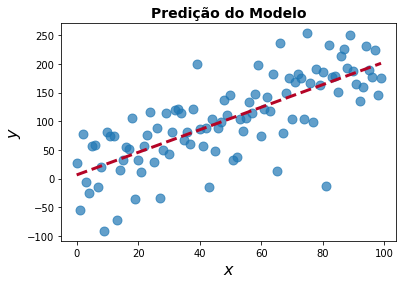
## 6. Obtenção de Métricas e Avaliação do Modelo

lm.score(x.reshape(-1, 1) ,y.reshape(-1, 1))

0.565013962458061

E, novamente, no caso desse problema simples, podemos visualizar a adequação dos dados ao modelo.

y\_pred = lm.predict(X)  
  
plt.scatter(x=x,y=y.reshape(1, -1)[0],s=80,alpha=0.7)  
plt.plot(x,y\_pred.reshape(1, -1)[0],color=cmap\_data(1.),lw=3,linestyle='dashed')  
plt.title('Predição do Modelo', fontsize=14, weight='bold')  
plt.xlabel('$x$', fontsize=16)  
plt.ylabel('$y$', fontsize=16)  
plt.show()



# Dilema Viés-Variância

Agora que sabemos como construir modelos Supervisionados para estimar valores e classes de dados, vamos nos deter um pouco sobre como avaliar melhor esses modelos. Afinal, parece bastante simples construir esses modelos, mas construir bons modelos pode ser de fato uma tarefa bastante difícil.

Vamos começar entendo o que é conhecido como **Dilema Viés-Variância** (ou *Bias–variance tradeoff*).

A capacidade de um modelo de capturar a verdadeira relação entre as variáveis preditoras e a variável objetivo é o que chamamos de viés (*bias*). Um alto erro de viés significa que o modelo não se ajusta aos dados e que, portanto, não consegue representar os dados. No limite ele não está aprendendo nada. É o que chamamos de **subajuste do modelo**, ou *underfitting*. Essa, é claro, é uma situação que não queremos.

Entretanto, ao tentarmos reduzir o erro de viés dos dados podemos nos deparar com um outro problema. Com um viés muito pequeno o modelo poder ficar tão ajustado aos dados de treinamento que falha ao tentar prever novos casos que não faziam parte do conjunto de treinamento. Neste caso, o modelo captura toda a variância dos dados. Você pode entender que o modelo passa a ser muito sensível a diferentes conjuntos de treinamento e erra ao *generalizar* novos casos. Isso é o que chamamos de **subreajuste do modelo**, ou *overfitting*.

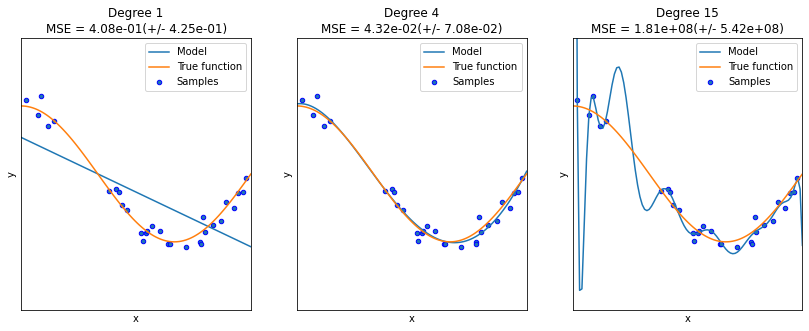
O dilema está, portanto, em obter o ponto de equilíbrio entre o subajuste e sobreajuste dos dados.

## Exemplo em Modelos de Regressão

O underfitting/overfitting pode ocorrer tanto em modelos de regressão como em modelos de classificação. O exemplo a seguir demonstra o problema de underfitting/overfitting para um caso de regressão.

# you can skip this code!  
  
# fonte: https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_underfitting\_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py  
print(\_\_doc\_\_)  
  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import cross\_val\_score  
  
  
def true\_fun(X):  
 return np.cos(1.5 \* np.pi \* X)  
  
np.random.seed(0)  
  
n\_samples = 30  
degrees = [1, 4, 15]  
  
X = np.sort(np.random.rand(n\_samples))  
y = true\_fun(X) + np.random.randn(n\_samples) \* 0.1  
  
plt.figure(figsize=(14, 5))  
for i in range(len(degrees)):  
 ax = plt.subplot(1, len(degrees), i + 1)  
 plt.setp(ax, xticks=(), yticks=())  
  
 polynomial\_features = PolynomialFeatures(degree=degrees[i],  
 include\_bias=False)  
 linear\_regression = LinearRegression()  
 pipeline = Pipeline([("polynomial\_features", polynomial\_features),  
 ("linear\_regression", linear\_regression)])  
 pipeline.fit(X[:, np.newaxis], y)  
  
 # Evaluate the models using crossvalidation  
 scores = cross\_val\_score(pipeline, X[:, np.newaxis], y,  
 scoring="neg\_mean\_squared\_error", cv=10)  
  
 X\_test = np.linspace(0, 1, 100)  
 plt.plot(X\_test, pipeline.predict(X\_test[:, np.newaxis]), label="Model")  
 plt.plot(X\_test, true\_fun(X\_test), label="True function")  
 plt.scatter(X, y, edgecolor='b', s=20, label="Samples")  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.xlim((0, 1))  
 plt.ylim((-2, 2))  
 plt.legend(loc="best")  
 plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(  
 degrees[i], -scores.mean(), scores.std()))  
plt.show()

Automatically created module for IPython interactive environment



A função que queremos aproxima é uma parte da função cosseno (linha laranja) e foram empregados modelos polinomiais de diferentes graus para aproximar a função (regressão). A função linear (polinômio com grau 1, linha azul) claramente é insuficiente para ajustar os dados de treinamento (underfitting). Um polinômio de grau 15 entretanto ajusta a maior parte dos pontos de dados do conjunto de treinamento, mas aprende desse modo todo o *ruído* ou variância dos dados, o que leva a um modelo (linha azul) que claramente falhará na predição de valores de novos casos (overfitting). O polinômio de grau 4, por outro lado, parece aproximar a função verdadeira quase perfeitamente.

## Exemplo em Modelos de Classificação

Isso pode igualmente ocorrer em modelos de Classificação.

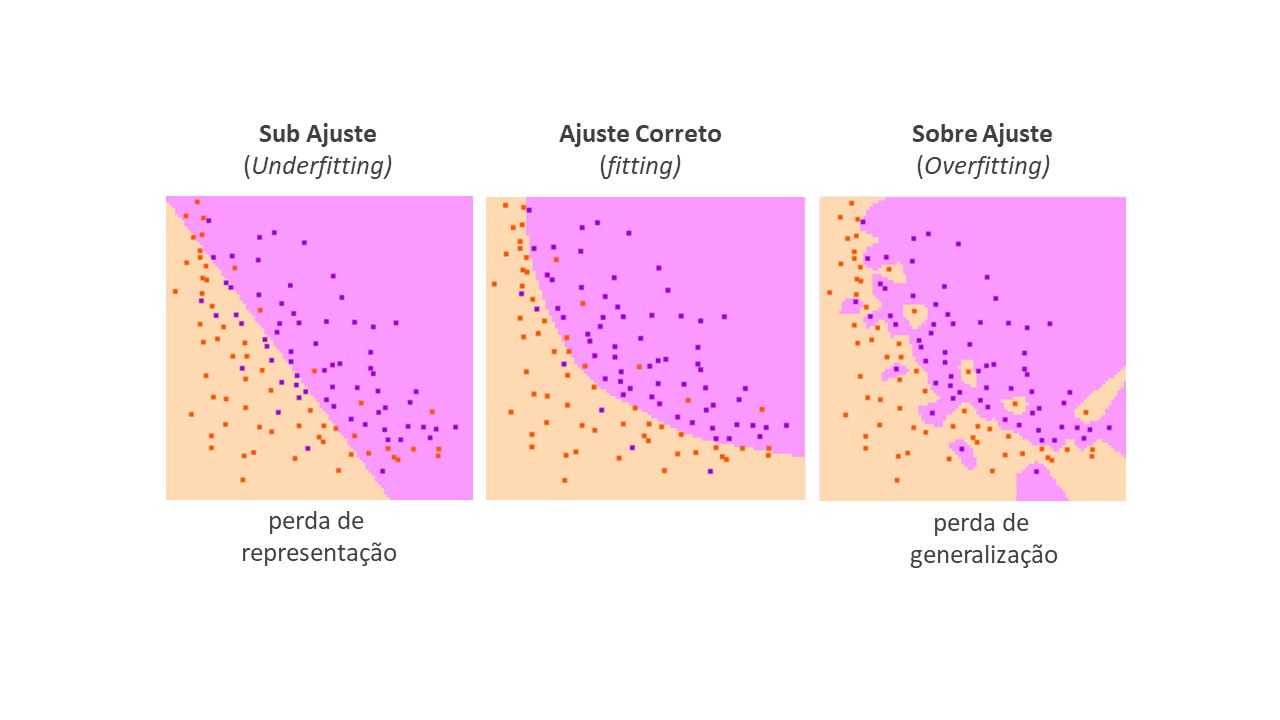


Figura 1. Exemplos de underfitting e overfitting. (Produzido com [https://ml-playground.com/#](https://ml-playground.com/))

Na figura acima o primeiro classificador é simplesmente um separador linear e erra ao classificar vários pontos de dados do conjunto de treinamento - é um modelo muito simples que não representa um modelo da classificação dos dados. Por outro lado, o terceiro modelo, captura toda a variância do conjunto de treinamento e classifica todas as instâncias corretamente, o que certamente, levará a falha de classificação de novos casos.

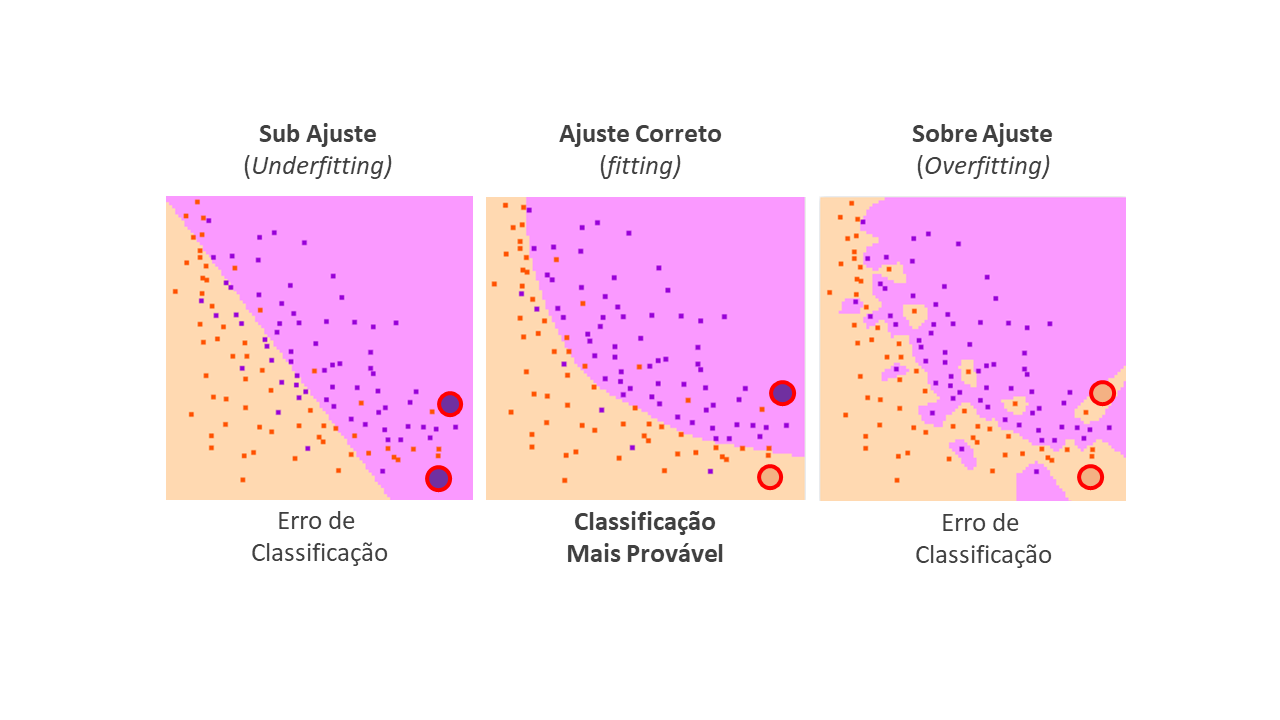


Figura 2. Exemplo de underfitting e overfitting, e de valores mais prováveis de predição. (Produzido com [https://ml-playground.com/#](https://ml-playground.com/))

*Isso pode ficar mais claro se você pensar do seguinte modo. Olhe os pontos grandes em vermelho da figura acima. Considere que eles seriam novos casos que você deve classificar. São novos casos, e você pode pensar no nosso exemplo de empréstimos, você não sabe antecipadamente se o cliente irá ou não pagar o empréstimo, mas você deseja definir a classe mais provável com base nos casos anteriores. Em quais classificações você apostaria? Certamente a classificação do meio. O primeiro é um modelo insuficiente para capturar todo o viés dos dados, e o terceiro, erra ao capturar todo o viés dos dados não generalizando suficientemente novos casos.*

# Conjuntos de Treinamento e Teste

Aprender ou treinar parâmetros de um modelo de predição e testá-lo com os mesmos dados é um erro de método. É como dar uma prova para avaliar o conhecimento de um aluno somente com exercícios que já foram dados previamente na sala da aula. Seria um modelo que apenas repete os rótulos das amostras podendo ser pefeito nesses dados, mas sem grande utilidade para predição de novos casos (sobreajuste).

Para evitar o sobreajuste, temos que definir dois conjuntos diferentes de dados a partir dos dados originais, um conjunto de Treinamento e um conjunto de Teste:

* O *conjunto de treinamento* X\_train, y\_train será a parte dos dados que empregada para o treinamento dos parâmetros do modelo
* O *conjunto de teste* X\_test, y\_test será a parte dos dados que empregada para avaliar o modelo preditivo ajustado

Essa divisão do conjunto original de dados deve ser aleatória (*por que?*) e pode ser obtida com o scikit-learn empregando-se a função train\_test\_split(). Seguiremos daqui em diante somente analisando modelos de classificação.

Vamos então aplicar essa abordagem mais correta ao nosso conjunto de dados *Moons*.

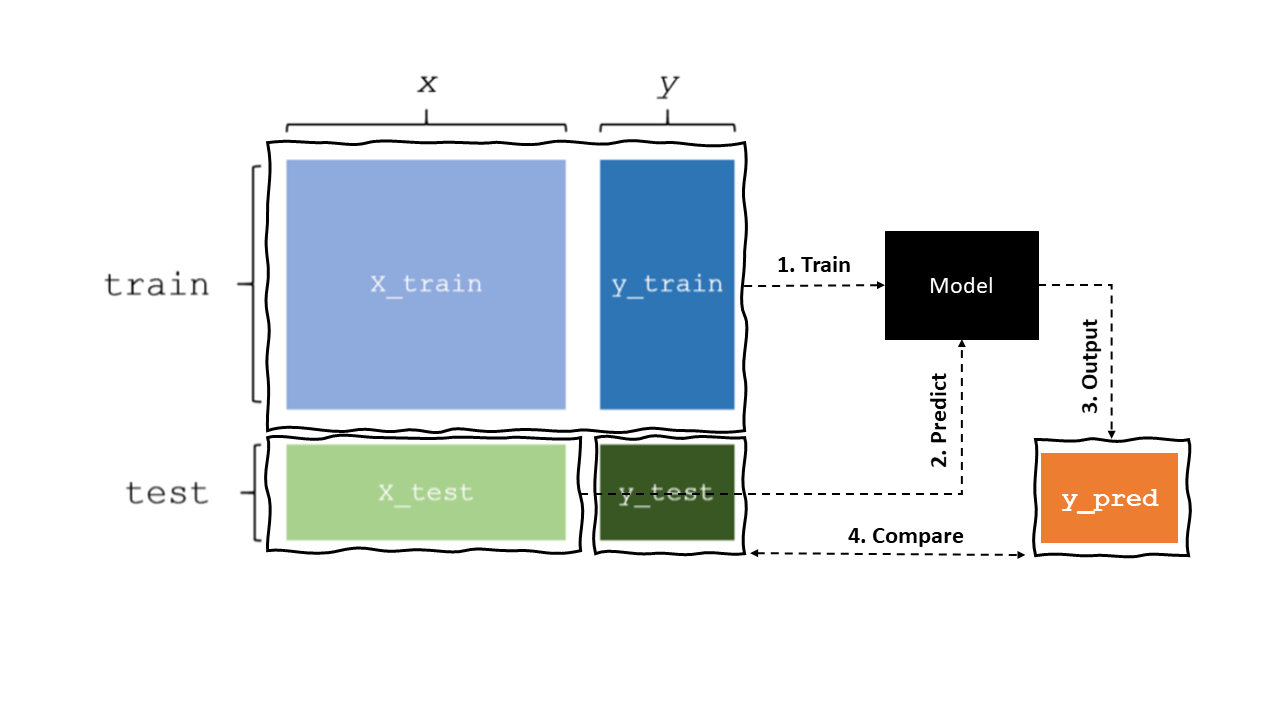


Figura 3. Conjuntos de Treinamento e Teste.

### Recriando os dados de *Moons*

X, y = make\_moons(n\_samples=100, noise=0.25, random\_state=1234)  
df = pd.DataFrame({'x1':X[:, 0], 'x2':X[:, 1], 'y':y})

### Modelo *sem* a Separação dos Conjuntos de Treinamento e Teste

Este é um modelo básico, mas com uma abordagem metodologicamente pois, ao empregar o mesmo conjunto para treinamento e a medida de desempenho do modelo pode levar ao sobre ajuste do modelo, e consequente erro de generalização.

from sklearn.linear\_model import LogisticRegression   
  
X = df[['x1','x2']]  
y = df.y  
  
clf = LogisticRegression(max\_iter=1000)  
  
clf.fit(X,y)  
  
y\_pred = clf.predict(X)  
  
print( y\_pred[0:10], '...' )  
print( clf.score(X,y) )

[0 0 1 1 1 1 1 0 0 0] ...  
0.87

### Modelo *com* a Separação dos Conjuntos de Treinamento e Teste

Aqui a abordagem correta, emprega conjunto diferentes treinamento e teste escolhidos aleatoriamente sobre os dados originais.

from sklearn.linear\_model import LogisticRegression   
from sklearn.model\_selection import train\_test\_split  
  
X = df[['x1','x2']]  
y = df.y  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.3, random\_state=123)  
  
clf = LogisticRegression(max\_iter=1000)  
  
clf.fit(X\_train,y\_train)  
  
y\_pred = clf.predict(X\_test)  
  
print( y\_pred[0:10], '...' )  
print( clf.score(X,y) )

[0 1 0 0 0 0 1 1 0 1] ...  
0.86

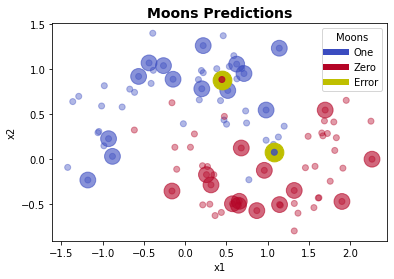
O parâmetro stratify=y indica que os conjuntos de treinamento e teste manterão a mesma proporção dos valores do atributo objetivo y. O parâmetros test\_size=0.3 indica o percentual (30%) dos dados que será empregado para teste. Esse valor varia, em geral, de 20-30% dos dados. Por fim, o pârametro random\_state=123 é para a reprodutibilidade dos resultados.

Abaixo, a inspeção visual, mostrando apenas 2 erros de classificação no conjunto de teste.

sum(y\_test != y\_pred)

2

# you can skip this code!  
  
cmap\_data = plt.cm.coolwarm   
  
plt.scatter(X.x1, X.x2, c=y, cmap=cmap\_data, alpha=0.4) # todos dados  
plt.scatter(X\_test.x1, X\_test.x2, c=y\_pred, s=250,cmap=cmap\_data, alpha=0.6) # somente dados de teste  
  
plt.scatter(X\_test[y\_pred != y\_test].x1, X\_test[y\_pred != y\_test].x2, color='y', s=350, alpha=1, label='wrong predictions')  
plt.scatter(X\_test[y\_pred != y\_test].x1, X\_test[y\_pred != y\_test].x2, c=y\_test[y\_pred != y\_test], cmap=cmap\_data, alpha=0.8)  
plt.title('Moons Predictions',weight='bold',fontsize=14)  
plt.xlabel("x1")  
plt.ylabel("x2")  
  
custom\_lines = [Line2D([0], [0], color=cmap\_data(0.), lw=6),  
 Line2D([0], [0], color=cmap\_data(1.), lw=6),  
 Line2D([0], [0], color='y', lw=6)]  
plt.legend(custom\_lines, ['One', 'Zero', 'Error'], loc='upper right',title='Moons')  
  
plt.show()



# CASO: 10 year risk of coronary heart disease CHD

Aqui o objetivo da classificação é prever se o paciente tem 10 anos de risco de futura doença cardíaca coronariana (ACS). O conjunto de dados fornece informações de mais de 4.000 pacientes residentes da cidade de Framingham, Massachusettse com 15 atributos.

A variável objetivo é TenYearCHD, *10 year risk of coronary heart disease CHD* (onde “1”, significa “Yes”, “0” significa “No”)

df = pd.read\_csv('https://raw.githubusercontent.com/TarekDib03/Analytics/master/Week3%20-%20Logistic%20Regression/Data/framingham.csv')  
df.head()

male age education currentSmoker ... BMI heartRate glucose TenYearCHD  
0 1 39 4.0 0 ... 26.97 80.0 77.0 0  
1 0 46 2.0 0 ... 28.73 95.0 76.0 0  
2 1 48 1.0 1 ... 25.34 75.0 70.0 0  
3 0 61 3.0 1 ... 28.58 65.0 103.0 1  
4 0 46 3.0 1 ... 23.10 85.0 85.0 0  
  
[5 rows x 16 columns]

## 

## Preparação dos Dados

As variáveis preditoras são todas numéricas e não há, portanto, necessidade do hot encode dos atributos. Há entretanto valores ausentes.

df.isnull().sum() / len(df)

male 0.000000  
age 0.000000  
education 0.024764  
currentSmoker 0.000000  
cigsPerDay 0.006840  
BPMeds 0.012500  
prevalentStroke 0.000000  
prevalentHyp 0.000000  
diabetes 0.000000  
totChol 0.011792  
sysBP 0.000000  
diaBP 0.000000  
BMI 0.004481  
heartRate 0.000236  
glucose 0.091509  
TenYearCHD 0.000000  
dtype: float64

O percentual é pequeno e assim vamos simplesmente excluir os dados ausentes.

print('Before',len(df))  
df = df.dropna()  
print('After',len(df))

Before 4240  
After 3658

## Aplicando o Modelo Logístico

Aplicamos o modelo ajustando agora as entradas X e y para o modelo. O max\_iter do regressor também foi ajustado para uma vez que o conjunto de dados agora é maior e um maior número de iterações será necessário para convergência do parâmetros.

from sklearn.linear\_model import LogisticRegression   
from sklearn.model\_selection import train\_test\_split  
  
X = df.drop(columns=['TenYearCHD'])  
y = df.TenYearCHD  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.3, random\_state=123)  
  
clf = LogisticRegression(max\_iter=10000)  
  
clf.fit(X\_train,y\_train)  
  
y\_pred = clf.predict(X\_test)  
  
print( y\_pred[0:10], '...' )  
print( clf.score(X,y) )

[0 0 1 0 0 0 0 0 0 0] ...  
0.8523783488244943

## Predição de Novos Casos

Se considerarmos 0.85 um bom resultado podemos então aplicar o modelo para novos casos. Por exemplo, podemos fazer predição para pacientes homens e mulheres hipotéticos em que todos os indicadores estejam no percentil 0.75.

X\_new = pd.DataFrame( df.drop(columns=['TenYearCHD']).groupby('male').quantile(0.75) ).reset\_index()  
X\_new

male age education currentSmoker ... diaBP BMI heartRate glucose  
0 0 56.0 3.0 1.0 ... 89.0 27.71 85.0 86.0  
1 1 56.0 3.0 1.0 ... 90.0 28.30 80.0 87.0  
  
[2 rows x 15 columns]

clf.predict(X\_new)

array([0, 0])

Surpreendentemente esses pacientes não apresentam risco segundo nosso modelo e você pode verificar outros percentis, como o percentil 0.9 em que já estimamos o risco para pacientes do sexo masculino.

X\_new = pd.DataFrame( df.drop(columns=['TenYearCHD']).groupby('male').quantile(0.90) ).reset\_index()  
clf.predict(X\_new)

array([0, 1])

# Outras Métricas: Matriz de Confusão e Classification Report

O resultado de 0.85 parece à primeira vista um resultado bastante satisfatório. Mas uma análise mais detalhada de outras métricas irá mostrar que esse resultado não é tão bom quanto parece. Algumas dessas métricas encontram-se abaixo e você vai ver em detalhe cada uma delas.

from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import classification\_report  
  
cm = confusion\_matrix(y\_test, y\_pred, labels=[1,0])  
print('\nMatriz de Confusão:\n')  
print(cm)  
   
accuracy = accuracy\_score(y\_test, y\_pred)  
print('\nScore de Acuracidade:\n')  
print(accuracy)  
  
print('\nClassification Report:\n')  
print(classification\_report(y\_test, y\_pred, labels=[1,0]))

Matriz de Confusão:  
  
[[ 14 153]  
 [ 11 920]]  
  
Score de Acuracidade:  
  
0.8506375227686703  
  
Classification Report:  
  
 precision recall f1-score support  
  
 1 0.56 0.08 0.15 167  
 0 0.86 0.99 0.92 931  
  
 accuracy 0.85 1098  
 macro avg 0.71 0.54 0.53 1098  
weighted avg 0.81 0.85 0.80 1098

## Matriz de Confusão

Uma matriz de confusão é uma matriz quadrada, para avaliar o desempenho de modelos de classificação e onde é o número de classes objetivo. Lembrando que avaliamos o modelo sobre os resultados no conjunto de teste, a matriz compara os valores reais (conjunto de teste) com aqueles estimados pelo modelo. Ela permite uma visão mais ampla do desempenho do modelo, que a simples medida de acuracidade, pois permite identificar os tipos de erros que o modelo está cometendo.

Em problema de classificação binária, como no nosso exemplo anterior, a matriz de confusão é uma matriz com o seguinte formato:

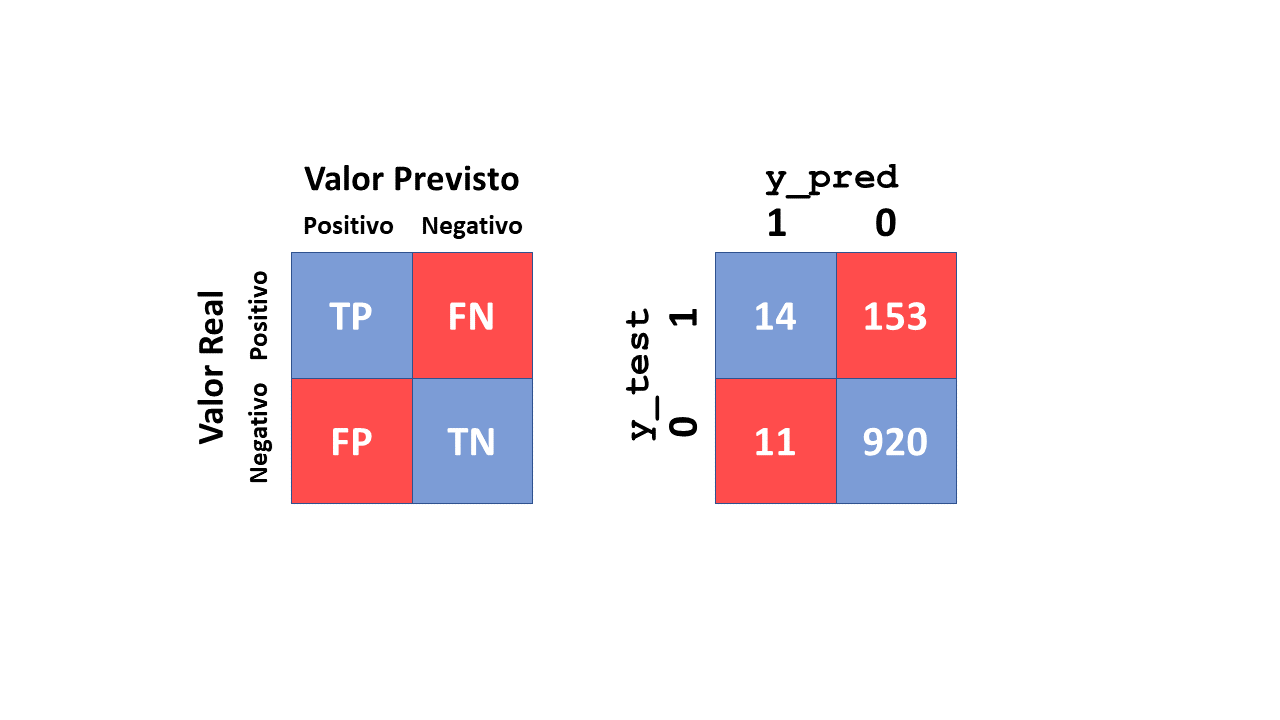


Figura 4. Matriz de Confusão: TP Verdadeiro Positivos (True Positives), TN, FP e FN.

A ordem dos labels, se não especificada será a ordem alfabética. Como no nosso caso '1' significa o caso positivo, isto é, a presença da doença cardíaca coronariana, colocamos a ordem [1,0] que será mais adequada para nossos propósitos.

# you can skip this code!  
  
cm = confusion\_matrix(y\_test, y\_pred, labels=[1,0])  
print('\nMatriz de Confusão:\n')  
  
print('\t\t\t y\_pred \n')  
print('\t\t\t 1 \t 0')  
print('\t\t' + 29\*'-')  
print(' y\_test \t 1 | \t ' + str(cm[0,0]) + '\t' + str(cm[0,1]) + ' | ' + str(cm[0,0] + cm[0,1]))  
print('\t\t 0 | \t ' + str(cm[1,0]) + '\t' + str(cm[1,1]) + ' | ' + str(cm[1,0] + cm[1,1]))  
print('\t\t' + 29\*'-')  
print('\t\t | \t ' + str(cm[0,0] + cm[1,0]) + '\t' + str(cm[0,1] + cm[1,1]) + ' | ')

Matriz de Confusão:  
  
 y\_pred   
  
 1 0  
 -----------------------------  
 y\_test 1 | 14 153 | 167  
 0 | 11 920 | 931  
 -----------------------------  
 | 25 1073 |

É um bom exercício para entender essa matriz verificar alguns de seus valores, como por exemplo a soma das linhas e colunas:

print( (y\_test == 1).sum(), (y\_test == 0).sum() )  
print( (y\_pred == 1).sum(), (y\_pred == 0).sum() )

167 931  
25 1073

De fato 167, 931 (soma das linhas) correspondem às quantidades de elementos reais classes 1 e 0 em y\_test. Já os valores 25 e 1073 (soma das colunas) correspondem às quantidades de elementos previstos nas classes 1 e 0 em y\_pred.

E você ainda pode verificar as instâncias classificadas corretamente para cada classe, que são os elementos da diagonal em que coincidem das classes reais e previstas pelo modelo.

print( ( y\_test + y\_pred == 2 ).sum() )  
print( ( y\_test + y\_pred == 0 ).sum() )

14  
920

Dessa matriz ainda saem os valores,

* Verdadeiro positivo (TP), em que valor previsto corresponde ao valor real, isto é o valor real era positivo e o modelo previu um valor positivo
* Verdadeiro negativo (TN), em que o valor real era negativo e o modelo previu um valor negativo também corretamente
* Falso Positivo (FP), Ou **Erro Tipo 1**, em que o valor real era negativo, mas o modelo previu um valor positivo
* Falso negativo (FN), Ou **Erro tipo 2**, em que o valor real era positivo, mas o modelo previu um valor negativo

Esses valores já permitem você identificar que nosso modelo com 0.85 de acurácia já não parece tão bom e para isso basta você notar o número de falsos negativos, *erro tipo 2*. E no caso esse é o erro mais grave, pois dizemos a uma pessoa que ela não está doente quando ela realmente está!

Que dão origem a dezenas de métricas e nos deteremos aqui apenas nas mais importantes.

## Acuracidade

Você já conhece essa métrica e apenas vamos verificar que, como outras métricas, ela pode ser obtida diretamente da matriz de confusão:

Note que é a soma dos valores diagonais, ou o total de acertos, e é simplesmente a soma de todos valores, ou o total de casos.

cm = confusion\_matrix(y\_test, y\_pred, labels=[1,0])  
print('\nMatriz de Confusão:\n')  
print(cm)  
  
TP, FP, FN, TN = cm.ravel()  
print('\nTP = ', TP, '\nFP = ', FP, '\nFN = ', FN, '\nTN = ', TN)

Matriz de Confusão:  
  
[[ 14 153]  
 [ 11 920]]  
  
TP = 14   
FP = 153   
FN = 11   
TN = 920

Accuracy = (TP+TN)/(TP+FP+TN+FN)  
Accuracy

0.8506375227686703

## Precisão e *Recall*

Outras duas métricas importante são a Precisão e o *Recall*,

e

Mas o que de fato essas métricas nos dizem? Você dificilmente irá guardar essas fórmulas. Mas é mais importante, e talvez até mais fácil, entender o conceito por trás dessas métricas o que já nos dá um modo direto de calcular os seus valores. A Precisão é um valor que, dados todos elementos previstos uma classe, quantos foram previstos corretamente. Note que é a exata noção de precisão que temos. O *Recall* (Revocação, ou Sensibilidade) por outro lado nos diz quantos casos de uma determinada classe foram corretamente previstos. E você pode pensar que precisamos fazer uma nova busca (*recall*) para os elementos da classe que ainda não foram identificados!

Precision = TP/(TP + FP)  
Precision

0.08383233532934131

Recall = TP/(TP + FN)  
Recall

0.56

Os valores acima são os valores para a classe 1, e esses mesmos valores você pode observar no classification\_report do scikit-learn.

A última métrica é o F1-score que pode ser entendido como uma média harmônica dos valores de precisão e recall:

Na prática, quando tentamos aumentar a precisão do nosso modelo, o recall diminui e vice-versa. A pontuação F1 permite capturar ambas as tendências em um único valor e, por isso é bastante empregada sendo seu valor máximo quanto a precisão e o recall são iguais.

F1\_score = 2 / ( (1/Recall) + (1/Precision) )  
F1\_score

0.14583333333333331

## 

## Classification Report

Todos conceitos acima são importantes para entendermos as métricas, mas todas essas métricas são mais facilmente obtidas no classification\_report.

print('\nClassification Report:\n')  
print(classification\_report(y\_test, y\_pred, labels=[1,0]))

Classification Report:  
  
 precision recall f1-score support  
  
 1 0.56 0.08 0.15 167  
 0 0.86 0.99 0.92 931  
  
 accuracy 0.85 1098  
 macro avg 0.71 0.54 0.53 1098  
weighted avg 0.81 0.85 0.80 1098

Além das métricas que já discutimos você encontra os valores de suporte, que nada mais são que as quantidades de casos de cada classe. Esse valor é importante para identificarmos classes desbalanceadas e são empregados para o cálculo das médias ponderadas das métricas weighted avg. O macro avg, por outro lado, é apenas a média das métricas de todas as classes de dados.

## Análise Final

Como você pode ver, apesar da acuracidade de do nosso modelo, ele acerta apenas pouco mais que uma moeda ( é são as chances de cara ou coroa), , dos casos positivos de doença. Além disso apenas 0.08 dos casos de doença foram identificados pelo modelo. Esses dois valores se refletem também baixo F1-score médio desse modelo.

Dados com classes desbalanceadas como esse (são somente 167 elementos da classe 1 para 931 casos da classe 0) são bastante comuns e casos até bastante mais desbalanceados, com 1/100 ou 2/100, nos casos de *churn de clientes*, *fraude de transações* ou mesmo diagnósticos de doenças graves, e em todos esses casos é uma felicidade que a classe de maior risco seja realmente bastante menor. Mas para o aprendizado de máquina isso é um grande problema pois permite atingirmos facilmente uma acuracidade geral alta apesar do mal desempenho do modelo.

## Um Caso Trivial

O exemplo abaixo ilustra bem o problema de classes desbalanceadas,

|  |  |  |
| --- | --- | --- |
| Real/Pred | Maligno | Benigno |
| Maligno | 2 | 3 |
| Benigno | 0 | 95 |

O modelo tem 0.97 de acuracidade, mas falha na maior parte dos casos na detecção de tumores malignos. Felizmente as amostras de tumores malignos é bastante menor que os casos benignos. Mas desse modo, mesmo um modelo trivial que classifica-se *qualquer* caso como benigno teria 0.95 de acerto!

# Síntese

Nesta trilha aprendeu alguns retomamos os problemas de aprendizado supervisionado, de regressão e classificação, para entender de modo geral como podemos empregar esses modelos com a biblioteca scikit-learn e você pode entender como instanciar um estimador, aplicar o treinamento, e fazer predições e medir o desempenho dos modelos obtidos.

Você aprendeu também sobre o Dilema Viés-Variância, e uma regra básica para buscarmos evitar o sobreajuste dos modelos dividindo o conjunto inicial de dados em dados para treinamento e teste. A partir daí verificamos algumas das métricas básicas para medir o desempenho dos modelos como Matriz de Confusão, Precisão, *Recall* e *F1-score*.

É importante que você saiba que ainda existem muitas outras técnicas e métricas envolvendo o desempenho dos modelos que não tratamos aqui, como a curva ROC (Receiver Operating Characteristic Curve), uso da validação cruzada (Cross Validation), Curva de Aprendizado ou Regularização. Isso permite você entender que a avaliação e obtenção de desempenho dos modelos não é uma tarefa fácil e voltaremos em alguns desses pontos mais adiante.

# Para Saber Mais

* Quer conhecer outras métricas obtidas a partir da Matriz de Confusão? São dezenas... rs. Acesse então **Precision and Recall** <https://en.wikipedia.org/wiki/Precision_and_recall> e **Confusion Matrix** <https://en.wikipedia.org/wiki/Confusion_matrix> e aproveite para ver outras referências nessas páginas.
* Você saberia aplicar os conceitos de Precisão e Recall quando temos mais de duas classes para a variável objetivo? Pense um pouco e depois acesse **Computing Precision and Recall for Multi-Class Classification Problems** <http://text-analytics101.rxnlp.com/2014/10/computing-precision-and-recall-for.html>.
* Não estudamos aqui a curva ROC (Receiver Operating Characteristic Curve), mas esse é um recurso que muitas vezes aparece na avaliação de modelos de classificação. Embora bastante sintético este artigo **Classification: ROC Curve and AUC**, disponível em <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc> é uma boa introdução a Curva ROC e é uma oportinidade também de você acessar o **Machine Learning Crash Course** da Google que traz vários temas com uma abordagem bastante interessante.
* Acesse o Livro Digital Larose, et. al. (2019) **Data Science Using Python and R**, **E-book disponível na Biblioteca do Mackenzie**. O Capítulo 7 é todo dedicado á avaliação de modelos.
* Esse é um tema bastante avançado, mas se você tiver interesse, pode entender o que é a Regularização de Modelos, uma técnica para se evitar o sobreajuste dos modelos durante o aprendizado, acessando o **Capítulo 5. Regressão e Regularização de Modelos** em Deep Learning I dispoível em: <https://github.com/Rogerio-mack/Deep-Learning-I/blob/main/T5.ipynb>

# Referências

Larose, Chantal D.; Larose, Daniel T. **Data Science Using Python and R** Hoboken: Wiley, c2019. E-book (259 p.) (Wiley Series on Methods and Applications in Data Mining Ser.). ISBN 9781119526834 (electronic bk.). Disponível em: <https://www3.mackenzie.br/biblioteca_virtual/index.php?tipoBiblio=ebookcentral&flashObg=n>

Kotu, Vijay; Deshpande, Balachandre **Data Science: concepts and practice**. 2nd ed. Cambridge, [England]: Morgan Kaufmann, c2019. E-book (570 p.) ISBN 9780128147627 (electronic bk.). Disponível em: <http://pergamum.mackenzie.br:8080/pergamumweb/vinculos/00003c/00003cef.jpg>.

Jake VanderPlas. **Python Data Science Handbook** O'Reilly Media, Inc. (2016). ISBN: 9781491912058. Disponível em: <https://jakevdp.github.io/PythonDataScienceHandbook/>. Acesso: 06 de Novembro de 2021.

\_\_\_. **An introduction to machine learning with scikit-learn** Disponível em: <https://scikit-learn.org/stable/tutorial/basic/tutorial.html> Acesso em: 06 de Novembro de 2021.

\_\_\_. **scikit-learn: machine learning in Python** Disponível em: <http://scipy-lectures.org/packages/scikit-learn/index.html> Acesso em: 06 de Novembro de 2021.