



# Ciclo 02 - O sistema supervisionado de aprendizado I

## Fundamentos Machine Learning

[Como funciona o aprendizado supervisionado em classificação?](#)

[Tarefas de classificação](#)

[Tarefa de regressão](#)

[Algoritmos de ML com aprendizado supervisionado](#)

[K-Nearest Neighbors - Teoria](#)

[K-Nearest Neighbors - Prática](#)

[Treinamento KNN](#)

[K-Nearest Neighbors - Exemplo de uso](#)

[Métricas de Avaliação I: Matriz de Confusão e Acurácia](#)

[Métricas de Avaliação II: Recall e Precision](#)

[Exemplo 1](#)

[Exemplo 2](#)

## Como funciona o aprendizado supervisionado em classificação?

Os sistema de machine learning pode ser classificados de acordo com o tipo de aprendizado:

1. Aprendizado Supervisionado
2. Aprendizado Não-Supervisionado
3. Aprendizado Semi-Supervisionado
4. Aprendizado por Reforço

O tipo de aprendizado supervisionado requer que nós explicitamos para o algoritmo qual é o resultado (output) que esperamos para um dado input. Em outras palavras o conjunto de dados de treino possui características que descrevem o fenômeno observado e um nome ou um rótulo que identifica aquele observado e o classifica em classes.

Esse tipo de aprendizado é bastante parecido com a forma que uma criança aprende, nós mostramos uma imagem de um cachorro e passamos o rótulo “cachorro” (nome do animal) para ela, então na próxima vez que ela ver um cachorro ela conseguirá identificar que se trata de um “cachorro” (rótulo) por conta própria. No início a criança ao ver um gato, pode se confundir e também associar a classe (rótulo) “cachorro” ao animal, cabe a um adulto (supervisor) informar a ela que o rótulo certo é “gato”, assim a criança aprende um novo animal. Nesse processo, a criança criou novas conexões no cérebro dela e aprendeu nesse processo, o mesmo raciocínio é aplicado no caso de algoritmos supervisionados.



No aprendizado supervisionado só se classifica aquilo que foi ensinado a ele.

Conforme rodamos o algoritmo em um conjunto de treinamento, com o passar do tempo ele tende a decorar os dados no conjunto de dados se não pararmos no momento correto, esse processo é chamado de **overfitting**, nesse processo o algoritmo leva em consideração o “noise” e imprecisões nos dados e deixa o nosso modelo muito complexo, de modo que ele não será capaz generalizar.

Nós queremos parar o processo de aprendizagem antes que o algoritmo overfit, o que significa que nós precisamos saber o quão bem ele está generalizado a cada intervalo de tempo / etapa do aprendizado. Nós não podemos usar o conjunto de treinamento para essa tarefa, porque nós não conseguiríamos detectar o overfitting (pois ele estaria ocorrendo justamente em relação a esse conjunto), mas nós também não podemos usar o conjunto de testes, porque nós precisamos dele para os testes finais (avaliação de acurácia do modelo). Então para isso, nós necessitamos de um terceiro conjunto de dados o qual é chamado de **conjunto de validação**, esse é dado porque usamos ele para validar o aprendizado até o momento. Esse processo é conhecido como **cross-validation** em estatística, e é um do **model selection**: escolher os parâmetros certos para o modelo de modo que generalize o melhor possível.




O **conjunto de treinamento** é usado para treinar o algoritmo.

O **conjunto de validação** é usado para manter um registro do quão bem o algoritmo está aprendendo.

O **conjunto de testes** é usado para avaliarmos a performance do modelo treinado.

### Tarefas de classificação


Quando o conjunto de dados possui o rótulo do fenômeno observado como um variável discreta, dizemos que é um problema de classificação, ou seja, o algoritmo precisa aprender a classificar os exemplos do conjunto, através da interpretação de suas características, em dos seus rótulos ou classes.



**Variáveis discretas** são usadas para medir características que podem assumir apenas um número finito contável de valores e, assim, somente fazem sentido valores inteiro.

Um exemplo de algoritmo de classificação seria uma máquina que identifica o valor da moeda baseado em suas medidas, diâmetro e espessura (features), e nós retornasse o valor que a moeda representa (classe). No início nós treinamos o algoritmo com as algumas moedas e passamos a classe que ela representa (aprendizado supervisionado), com isso o algoritmo muda o seus parâmetros internos (cria novas conexões) e generaliza a tarefa de classificação para moedas ainda não vistas.

| Diâmetro (mm) | Espessura (mm) | Classe |
|---------------|----------------|--------|
| 20            | 0.2            | 50c    |
| 12            | 0.15           | 10c    |
| ...           | ...            | ...    |
| 22            | 0.21           | 50c    |




É extremamente importante que o conjunto de treinamento possua datapoints com todas as classes presentes no dataset. Se isso não ocorrer, o algoritmo não vai ter oportunidade de aprender todas as classes e ele atribuirá a classe errada para datapoints cuja a classe verdadeira não foi vista no treinamento.

**No aprendizado supervisionado só se classifica aquilo que foi ensinado a ele.**

Em muitos casos o rótulo é atribuído por um especialista no assunto, pode ser um biólogo para nós dizer qual classe uma planta pertence, ou alguém que trabalhe com moda para nos dizer os diferentes tipos de sapato. Uma vantagem de implementar um algoritmo de machine learning é que o custo desse especialista pode ser bastante elevado em alguns casos, além do fato de que a atribuição do rótulo pode ser um processo bastante demorado dependendo do que estamos classificando.

### Tarefa de regressão

Quando o conjunto de treinamento possui o rótulo do fenômeno observado como uma variável real ou contínua, como salário ou peso, por exemplo. Nesse tipo de tarefa, algoritmo tenta ajustar um novo ponto, ao conjunto de pontos do conjunto de dados, de modo que apresente a menor distância possível dos pontos do conjunto de dados.



Uma regressão é uma versão da classificação onde os rótulos são números ao invés de classes. Porém o número de valores que o rótulo/variável consegue assumir não é limitado, podendo assumir valores reais.

Algoritmos de regressão conseguem extrapolar o valor real obtido para intervalos que não estavam inicialmente no seu intervalo, por exemplo, um algoritmo que foi treinado para prever o salário de novos funcionários em um startup pode prever salários maiores do que estavam inicialmente no teste. Digamos que o algoritmo foi treinado com funcionários com salários no intervalo [ \$4.500, \$18.000 ], nada impedirá que o algoritmo possa prever que um novo funcionário receba um salário de \$19.500, o algoritmo é capaz de extrapolar as suas previsões para valores fora do intervalo.

### Algoritmos de ML com aprendizado supervisionado

Toda a teoria do aprendizado de máquina surgiu para resolver problemas de classificação.

#### Classificação

1. K-Nearest Neighbors
2. Naive Bayes
3. Light Gradient Boost Machine ( LGBM )
4. Categorical Boost ( CatBoost )

- 5. Logistic Regression
- 6. Support Vector Machine ( SVM )
- 7. Decision Tree
- 8. Random Forest

Regressão

- 1. Linear Regression
- 2. Polinomial Regression

K-Nearest Neighbors - Teoria


O KNN é um algoritmo de machine learning da classe de aprendizado supervisionado. Ele pode ser utilizado tanto para resolver problemas de classificação quanto problemas de regressão.

Processo de treinamento de KNN

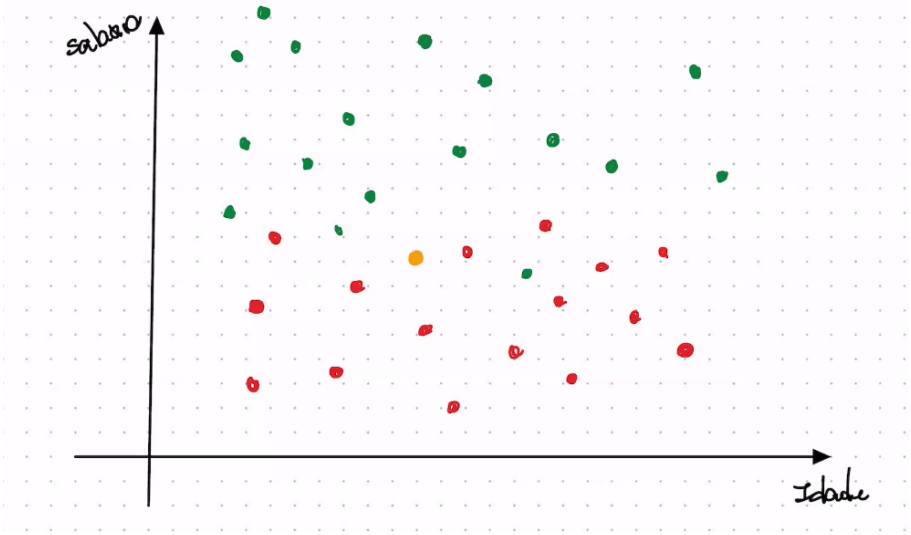
Imagine um conjunto de dados com a idade e o salário (features), e suponha que também temos uma classe que queremos classificar (pobre ou rico). Com os dados que temos podemos plotar um gráfico com cada feature representando um dos eixos: x → idade e y → salário.

Suponha que nos foi passado um novo datapoint (registro) sem a classe especificada (ponto em laranja). Como podemos descobrir qual classe o ponto laranja pertence?

Uma saída é analisar qual classe os vizinhos mais próximos pertencem e verificar se há mais vizinhos pobres (vermelho) ou vizinhos ricos (verde), esse basicamente é o processo que algoritmos de KNN utilizam: para um dado “k” nós verificamos qual classe os “k” vizinhos mais próximos pertencem e atribuímos a classe da maioria para esse novo ponto.

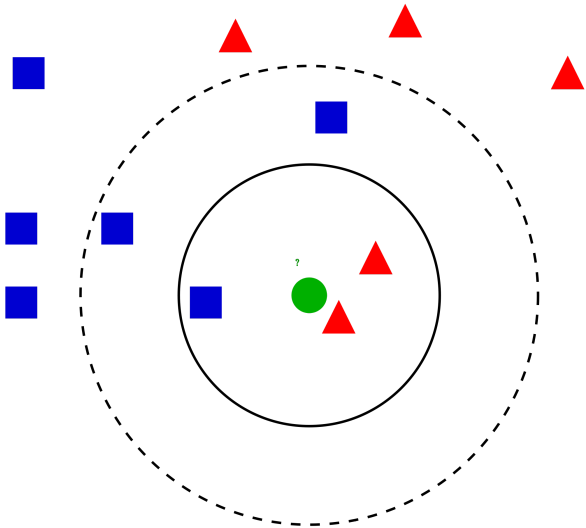


É preferível escolher um número “k” ímpar de modo para que não haja empate entre o número de vizinhos em cada classe



Os 8 passos para treinar o KNN

- 1. Carregue os dados.
- 2. Defina o número de K vizinhos mais próximos.
- 3. Para cada exemplo sem rótulo:
  - a. Calcule a distância entre o ponto sem o rótulo e todos os outros pontos rotulados do conjunto de dados.
  - b. Encontre o K pontos mais próximos.
  - c. Identifique os rótulos de cada um dos K vizinhos mais próximos
  - d. Calcule a “moda” dos K vizinhos mais próximos.
  - e. Classifique o ponto sem rótulo com a moda de seus K vizinhos mais próximos.



No algoritmo precisamos calcular a distância entre 2 pontos e isso pode ser feito usando diferentes métodos:

Euclidean distance

$$d = \sqrt{(P_x - Q_x)^2 + (P_y - Q_y)^2}$$

Manhattan distance

$$d = |P_x - Q_x| + |(P_y - Q_y)|$$

Mahallanobis distance

$$d = \sqrt{\frac{(P_x - Q_x)^2 + (P_y - Q_y)^2}{\sigma_{xy}^2}}$$



Devido a diferença entre o valor numérico entre as features é necessário fazer uma normalização de modo a fazer que as grandezas variem em um mesmo intervalo.

idade - [18, 80] → [0, 1]  
salário - [1500, 30000] → [0, 1]

Caso não fizermos a normalização, a distância seria dominada pelos valores numéricos de um dos eixo. (Tópico será discutido posteriormente)

### Premissas assumidas do KNN

A premissa assumida pelo KNN é a seguinte:

A proximidade entre duas evidências de um fenômeno observado, pode ser representada por uma medida de distância.

Coisas similares estão mais próximas uma das outras, portanto, todas as suas características precisam ser numéricas.

## K-Nearest Neighbors - Prática

Para vermos um uso do KNN na prática vamos utilizar o dataset do último hackday. Nossa tarefa era treinar algoritmos para prever se o banco deveria liberar um empréstimo para os seus clientes, ao final do treinamento nós comparamos os outputs com targets da coluna `limite_adicional`.

### ▼ Treinamento KNN

```
# Importando as bibliotecas necessárias
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics as mt

# Seleção de Features
features = ['idade', 'saldo_atual', 'divida_atual', 'renda_anual',
            'valor_em_investimentos', 'taxa_utilizacao_credito', 'num_emprestimos',
            'num_contas_bancarias', 'num_cartoes_credito', 'dias_atraso_dt_venc',
            'num_pgts_atrasados', 'num_consultas_credito', 'taxa_juros']

label = 'limite_adicional'

x_train = df.loc[:, features] # Dados que serão usado para treinar o algoritmo
y_train = df.loc[:, label]    # Series que contém as labels de cada classe

#
# ===== Treinamento =====

# definição parâmetro de treinamento, esse parâmetro vai dizer ao algoritmo quantos vizinhos devemos levar em consideração
k = 7

# criação do objeto que perfomará o algoritmo de classificação. Nesse caso
# o objeto 'knn_classifier' é uma instância da classe 'KNeighborsClassifier'
# que avaliará e treinará o algoritmo utilizando os 7 vizinhos mais próximos.
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Vamos agora treinar o algoritmo no conjunto de treino `x_train`. Para isso nós chamamos o método `.fit()` do objeto `KNeighborsClassifier`
# e passamos como parâmetros o conjunto de treino e os rótulos das classes.

knn_classifier.fit(x_train, y_train) # treinamento do algoritmo no conjunto de dados.

#
# ===== Previsão =====

# Vamos agora utilizar o algoritmo treinado para tentar prever qual é classe (`Liberar` ou `Negar`) de cada registro (cliente) do dataset de treino.
```

```
# Para isso nós chamamos o método `.predict()`.

y_pred = knn_classifier.predict(x_train) # y_pred é uma series que contém os rótulos atribuídos pelo algoritmo
```

Colocando lado a lado os rótulos reais com os rótulos atribuídos podemos montar um tabela.

```
df_result = df.copy()
df_result['classificacao'] = y_pred

df_result.loc[:, ['id_cliente', 'idade', 'limite_adicional', 'classificacao']].sample(10)
```

|      | id_cliente | idade | limite_adicional | classificacao |
|------|------------|-------|------------------|---------------|
| 2155 | 10385      | 39    | Negar            | Negar         |
| 4956 | 8261       | 35    | Negar            | Negar         |
| 3753 | 1544       | 34    | Negar            | Negar         |
| 6795 | 1472       | 25    | Negar            | Negar         |
| 8013 | 10602      | 24    | Negar            | Negar         |
| 7192 | 4349       | 25    | Negar            | Negar         |
| 7861 | 4862       | 35    | Negar            | Negar         |
| 8138 | 1858       | 22    | Negar            | Conceder      |
| 469  | 10985      | 43    | Negar            | Negar         |
| 4494 | 7531       | 47    | Negar            | Negar         |

## K-Nearest Neighbors - Exemplo de uso

O KNN consegue encontrar um rótulo a partir de seus K vizinhos mais próximos.

- **Sistemas de recomendação.**
  - Filmes que você pode gostar.
    - Dado um filme encontre os k filmes mais similares.
    - Recomendamos para o filme para um cliente baseado nos vizinhos (clientes) mais próximos (com interesses parecidos) dentro do nosso espaço de dados.
- **Classificação de Notícias.**
  - Dado uma nova notícia, em qual categoria ela mais se encaixa?
- **Agrupamento de Clientes.**
  - Dado um novo cliente, qual o tipo de outros clientes parecido com ele?
- **Classificação de Imagem.**
  - Dado uma nova imagem sem rótulo e grupos de imagens com fotos de pandas, ursos, cachorros, cavalos e etc, em qual grupo a nova imagem pode ser classificada?
  - O KNN é muito usado para classificações em imagens, pois uma imagem nada mais é que uma matriz de números em que cada número da matriz representa um pixel e o seu valor nos diz a quantidade de vermelho, azul e verde.
- **Sistema de Busca.**
  - Dada uma nova consulta, quais as notícias que podem ser retornadas como resultado?

### Vantagens e Desvantagens

#### Vantagens

1. O KNN é fácil de entender e simples de explicar o seu funcionamento.

#### Desvantagens

1. O KNN se torna significativamente devagar quanto mais o número de exemplos e/ou preditores aumenta.

2. Não há necessidade de construir um modelo, ajustar vários parâmetros ou assumir premissas adicionais.

3. O KNN é versátil. Ele pode ser usado para resolver problemas de classificação, regressão e busca.
2. Em alta dimensionalidade, a distância entre os pontos podem ser distorcidas.

Métricas de Avaliação I: Matriz de Confusão e Acurácia

A matriz de confusão é uma ferramenta para medir a performance de um algoritmo de machine learning. A ideia geral é contar o número de vezes em que os exemplos da classe A são classificados erroneamente como classe B.

Para nós auxiliar a entender a performance do nosso algoritmo fazemos uso da matriz de confusão. Para construir a matriz de confusão construímos uma tabela com as classes, sendo que no topo da tabela estará listadas as classes atribuídas pelo algoritmo (predicts) e na primeira coluna estão os targets (valores que deveriam ocorrer).

No exemplo ao lado, tínhamos 6 registros com target C1 (soma dos valores da primeira linha) e o algoritmo acertou 5 deles e errou 1 (onde foi atribuído C2). Ao todo o algoritmo previu que 8 dos data points seriam da classe C1 (soma da primeira coluna), portanto ele acertou esse classe 62.5%(5/8).

No conjunto de teste havia 6 registros com target C2, do qual foi acertado 4. No entanto o algoritmo previu que haveria 5 registros da classe C2, o que significa que ele acertou 80%(4/5).

Por fim, tínhamos também 6 registros com target C3, do quai foi acertado 4. Novamente o algoritmo previu que haveria 5 registros na classe C3, significando que ele acertou 80%(4/5).

1. Classe C1 (6 registros)

a. 5 acertos

b. 1 erro
2. Classe C2 (6 registros)


a. 4 acertos

b. 2 erros
3. Classe C3 (6 registros)

a. 4 acertos

b. 2 erros

|             | C1 (predict) | C2 (predict) | C3 (predict) |
|-------------|--------------|--------------|--------------|
| C1 (target) | 5            | 1            | 0            |
| C2 (target) | 1            | 4            | 1            |
| C3 (target) | 2            | 0            | 4            |



Se quisermos saber o número de registros que o algoritmo classificou corretamente basta somarmos o termo da diagonal principal. Nesse caso foram classificados corretamente 13 registros dos 18 disponíveis no conjunto de testes, o que resulta em uma acurácia de 72.2%.

Nós podemos analisar o resultado medindo mais do que apenas a acurácia. Se nós considerarmos os possíveis outputs de um treinamento com duas classes e estudarmos os resultados, podemos separar a avaliação em 4 diferentes categorias:

Resultado quando estamos analisando o comportamento da classe 1:

1. True positive: Uma observação (predict) é corretamente colocada na classe 1.

2. False positive: Uma observação (predict) é incorretamente colocada na classe 1.

3. False negative: Uma observação (predict) é incorretamente colocada na classe 2.

4. True negative: Uma observação (predict) é corretamente colocada na classe 2.

|                    | 1 (predict)      | Not 1 (2) (predict) |
|--------------------|------------------|---------------------|
| 1 (target)         | 5 True positive  | 1 False negative    |
| Not 1 (2) (target) | 3 False positive | 9 True negative     |

Criando a tabela da matriz de confusão para cada uma das classes com o exemplo das classes C1, C2 e C3 obtemos o seguinte resultado:

Montando as tabelas:

1. A classe prevista é  $\hat{C}_k$ ? Se sim atribuímos positivo, caso contrário negativo.

|                 | C1 (predict)     |                 | C2 (predict)     |                 | C3 (predict)     |       |
|-----------------|------------------|-----------------|------------------|-----------------|------------------|-------|
| C1 (target)     | 5 True positive  | C2 (target)     | 4 True positive  | C3 (target)     | 4 True positive  | True  |
| Not C1 (target) | 3 False positive | Not C2 (target) | 1 False positive | Not C3 (target) | 1 False positive | False |

2. A classe target é igual a prevista? Se  
sim atribuímos True caso contrário False

Na hora de mensurarmos a performance de um modelo podemos identificar uma das suas classes como sendo a classe positiva e outra como sendo negativa, e a partir disso verificarmos quando o modelo acertou ou errou as classes positivas e negativas.

|             |                 | Previsão do Rótulo  |                     |
|-------------|-----------------|---------------------|---------------------|
|             |                 | Classe Positiva (P) | Classe Negativa (N) |
| Rotúlo Real | Classe Positiva | True Positive (TP)  | False Negative (FN) |
|             | Classe Negativa | False Positive (FP) | True Negative (TN)  |

- **Total de previsões** = P + N
- **True Positive** = Classificação correta da classe positiva.
- **True Negative** = Classificação correta da classe negativa.
- **False Positive** = Classificação errada da classe positiva.
- **False Negative** = Classificação errada da classe negativa.

A acurácia é a quantidade de previsões acertadas da classe A e da classe B. Em outras palavras, quantas vezes o algoritmo classificou uma observação como classe A, sendo que ela era realmente da classe A (True Positive), mais a quantidade de vezes que o classificador classificou como classe B (True Negative).

A acurácia pode então ser definida da seguinte maneira:

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#FP + \#TN + \#FN}$$


Um problema da acurácia é tendência para classes com um número maior de observações. Em datasets desbalanceados o valor da acurácia é dominado pela classe majoritária.

|             |               | Previsão do Rótulo |          |
|-------------|---------------|--------------------|----------|
|             |               | Classe A           | Classe B |
| Rotúlo Real | Classe A (98) | 95                 | 3        |
|             | Classe B (2)  | 2                  | 0        |

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#FP + \#TN + \#FN} = \frac{95 + 0}{95 + 2 + 0 + 3} =$$

A acurácia de 95% nos leva a crer que nosso modelo é bastante preciso na hora de identificar as classes, porém se prestarmos atenção na tabela verificamos que o modelo não acertou nenhuma observação da classe B

Esse downside se torna ainda maior quando é mais importante identificar a classe minoritária do que a majoritária. Um exemplo disso seria um algoritmo para identificar transações fraudulentas, nesse exemplo a grande maioria das movimentações financeiras são honestas e se usarmos somente a acurácia poderíamos concluir que nosso modelo é bom mesmo que ele errasse mais da metade das classificações fraudulentas, sendo que elas representam o maior problema dentro da nossa base.

 A acurácia não é a melhor métrica de avaliação para conjuntos de dados desbalanceados.

```
from sklearn import metrics as mt

# para calcularmos a matriz de confusão, precisamos passar como parâmetros os rótulos reais e os previstos
mt.confusion_matrix(y_train, y_pred)

# calcula a acurácia do modelo
mt.accuracy_score(y_train, y_pred)
```

Métricas de Avaliação II: Recall e Precision



Pensando em um exemplo de detecção de fraude de cartão, o que é melhor nos avaliarmos, a precisão ou o recall?. A precisão vai nos indicar quais observações eram realmente fraude dentro das quais foram classificadas como fraude, enquanto que o recall vai nos dizer qual é a porcentagem fraudes dentro das observações que realmente eram fraudes.

$$\text{Precision} = \frac{\#TP}{\#TP + \#FP}$$

$$\text{Recall} = \frac{\#TP}{\#TP + \#FN}$$

Antes de escolher qual métrica queremos otimizar a performance, devemos decidir qual critério é mais importante. No exemplo da detecção de fraude devemos escolher entre:

- Precisão:** Toda vez que disser que é fraude, realmente será uma fraude (sem se interessar como o algoritmo classifica transações não-fraudulentas) .
- Recall:** É mais importante recuperar a maioria dos clientes fraudulentos dentro base.


▼ Exemplo 1

Você está interessado em prever o Churn do cliente, ou seja, o momento em que ele vai parar de usar o seu produto. Isso pode acontecer quando ele não renova o contrato por mais um período de tempo, quando ele para de acessar o site e etc. Os resultados do treinamento do algoritmo são mostrado abaixo:

|         |           | Previsão |           |   |
|---------|-----------|----------|-----------|---|
| n = 400 |           | Churn    | Not Churn |   |
| Real    | Churn     | 300      | 35        | <b>Acurácia:</b> ( 300 + 50 ) / ( 300 + 35 + 15 + 50 ) = <b>87.5%</b> |
|         | Not Churn | 15       | 50        |   |
|         |           |          |           | <b>Precisão:</b> ( 300 ) / ( 300 + 15 ) = <b>95%</b>                  |
|         |           |          |           | <b>Recall:</b> ( 300 ) / ( 300 + 35 ) = <b>89%</b>                    |

Qual seria métrica mais adequada para medir a performance desse classificador?

Nesse exemplo a métrica mais adequada é o **recall**, pois é mais importante atestar a performance do modelo nos datapoints (clientes) classificados como Churn dentro dos que eram realmente Churn, desse modo podemos direcionar propagandas para esse tipo de cliente. Observamos que a métrica a ser escolhida nem sempre é aquela que possui a maior porcentagem de acerto, o conhecimento de negócio é crucial antes de tomarmos a decisão de qual métrica queremos otimizar.



É crucial definirmos a métrica a ser utilizada antes de iniciarmos o treinamento modelo.

▼ Exemplo 2

Você foi contrato como cientista de dados para classificar se um produto será devolvido ou não. Os resultados do treinamento do algoritmo são mostrados abaixo:

|         |               | Previsão  |               |   |
|---------|---------------|-----------|---------------|---|
| n = 210 |               | Devolução | Não Devolução |   |
| Real    | Devolução     | 20        | 80            | <b>Acurácia:</b> ( 20 + 50 ) / ( 20 + 60 + 80 + 50 ) = <b>33.3%</b> |
|         | Não Devolução | 60        | 50            |   |
|         |               |           |               | <b>Precisão:</b> ( 20 ) / ( 20 + 60 ) = <b>25%</b>                  |
|         |               |           |               | <b>Recall:</b> ( 20 ) / ( 20 + 80 ) = <b>20%</b>                    |

Qual seria métrica mais adequada para medir a performance desse classificador?

Quando um cliente devolve um produto estamos dobrando o custo com logística (entrega + retirada). Como o custo será maior no caso da devolução, devemos tentar otimizar o algoritmo de modo que ele seja capaz de classificar como *Devolução* o maior número de clientes que realmente irão devolver, ou seja, devemos maximizar o **Recall**.





No geral, o recall é uma métrica melhor se a classificação errada da classe analisada implica em perdas financeiras a empresa.

Essas duas métricas podem ser combinadas em uma única chamada **F1-score**.

Essa métrica é geralmente usada quando estamos interessado em obter bons valores tanto para a precisão quanto para o recall.

$$F_1 = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$