> Muito se imagina que a ciência de dados se resume ao aprendizado de máquina e que os cientistas de dados constroem, treinam e ajustam modelos desta classe o dia todo. Na verdade, a ciência de dados está transformando problemas de negócios em problemas de dados. Coletando, entendendo, limpando e formatando dados. Por fim, vem o aprendizado de máquina. Ainda assim, é um passo interessante e essencial que você precisa saber para aplicar a ciência de dados.

Modelagem

Antes de falarmos sobre aprendizado de máquina, precisamos falar sobre *modelos*. O que é um modelo? É simplesmente uma especificação de uma relação matemática (ou probabilística) que existe entre diferentes variáveis.

Por exemplo, se você estiver tentando arrecadar dinheiro para sua empresa, poderá criar um modelo de negócios que receba entradas (provavelmente uma planilha) como 'número de usuários', 'receita de anúncios por usuário' e 'número de funcionários' e gerar seu lucro anual para os próximos anos. Você pode se basear em relações matemáticas simples: lucro é receita menos despesas, receita é unidades vendidas vezes preço médio e assim por diante.

Já a receita de uma sobremesa sofisticada envolve um modelo que relaciona insumos como 'número de pessoas que serão servidas' e 'intensidade de fome de cada pessoa' às quantidades de ingredientes necessários. Provavelmente, ela foi obtida de uma relação empírica: alguém entrou em uma cozinha e experimentou diferentes combinações de ingredientes até encontrar um que gostou.

Por fim, em um jogo de cartas (por exemplo, pôquer) sabe-se que a 'probabilidade de ganhar' de cada jogador é estimada em tempo real com base em um modelo que leva em consideração as cartas que foram reveladas até o momento e a distribuição das cartas no baralho baseando-se na teoria das probabilidades.

> No mundo real, podemos juntar todas essas abordagens.

O que é Aprendizado de Máquina?

Embora exista muitas definições, usaremos o termo 'aprendizado de máquina' para nos referir à criação e uso de modelos que são aprendidos a partir de dados. Em outros contextos, isso pode ser chamado de modelagem preditiva. Normalmente, nosso objetivo será usar dados existentes para desenvolver modelos que possamos usar para prever (ou predizer) resultados para novos dados, tais como:

- Se um e-mail é spam;
- Se uma transação com cartão de crédito é fraudulenta;
- Em qual anúncio um comprador provavelmente clicará;
- Qual time de futebol vai ganhar o campeonato.

Os modelos podem se dividir em classes. Os mais usuais são os supervisionados (nos quais há um conjunto de dados rotulados com as respostas corretas para aprender) e não supervisionados (nos quais não há esses rótulos). Existem vários outros tipos, como semi-supervisionado (em que apenas alguns dos dados são rotulados), online (em que o modelo precisa se ajustar continuamente aos dados recém-chegados) e de reforço (em que, após fazer uma série de previsões, o modelo recebe um sinal indicando quão bem ele foi) que não abordaremos neste curso.

Mesmo na situação mais simples, existe uma ampla gama de modelos que podem descrever o relacionamento em que estamos interessados. Na maioria dos casos, nós escolhemos uma família de modelos e, em seguida, usamos dados para descobrir o membro desta família que melhor se enquadra ao dados. Em outras palavras, estimamos os elementos desconhecidos do modelo através dos dados. Estes elementos serão chamados de *parâmetros*.

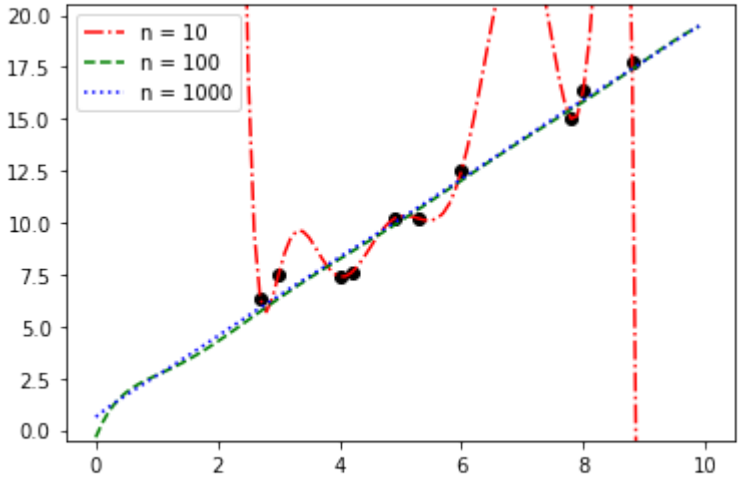
Por exemplo, podemos supor que a altura de uma pessoa é (aproximadamente) uma função linear de seu peso e, em seguida, usar dados para aprender qual função linear é essa (descobrir sua angulação e interceptação vertical, parâmetros de uma função linear).

Sobreajuste e Subajuste

Um perigo comum no aprendizado de máquina é o *sobreajuste* - produzir um modelo que funciona bem nos dados em que você o treina, mas generaliza mal para novos dados. Por outro lado, temos o *subajuste* - produzir um modelo que não funciona bem nem nos dados de treinamento.

A figura abaixo apresenta três polinômios ajustados em uma amostra de dados (não se preocupe em como foi feito... veremos em breve). A linha horizontal mostra o polinômio de grau 0 de melhor ajuste (isto é, constante - 1 parâmetro). O polinômio de grau 9 com melhor ajuste (10 parâmetros) passa exatamente por todos os dados de treinamento, mas sobreajusta muito severamente. O polinômio de grau 1 (função

linear - 2 parâmetros) atinge um bom equilíbrio; está bem perto de todos os pontos e, se esses dados forem representativos, a linha provavelmente também estará próxima de novos dados.



Claramente, modelos muito complexos levam ao sobreajuste e não generalizam muito além dos dados em que foram treinados. Então, como podemos garantir que nossos modelos não sejam muito complexos? A abordagem mais fundamental envolve o uso de dados diferentes para treinar o modelo e testá-lo. Vamos aprender a fazer isso utilizando a base de dados de uma fábrica de sorvetes que contém informações sobre a temperatura média e a receita obtida:

In [ ]:
```
import pandas as pd

# importando dados
vendas = pd.read_csv("/content/Dados_Sorvete.csv")

# matriz de correlacao
vendas.corr()

# matriz de dispersao
pd.plotting.scatter_matrix(vendas)
```

A maneira mais simples de evitar o sobreajuste é separar o conjunto de dados de forma aleatória, de modo que (por exemplo) 80% dele sejam usados para treinar o modelo e o restante sirva como um teste de desempenho. Chamaremos esses conjuntos, respectivamente, de **treino** e

teste:

In [ ]:
```
from sklearn.model_selection import train_test_split

# definindo conjunto de treino e teste simultaneamente
# aqui, o conjunto de treino sera 80% dos dados
treino, teste = train_test_split(vendas, test_size = 0.80)

print(treino.shape)
print(teste.shape)
print(vendas.shape)
```

Note que a cada vez que você realizar esse procedimento, uma amostra diferente será obtida. Essa é a melhor forma de evitar com que sua seleção induza determinados resultados.

Muitas vezes, teremos variáveis de entrada (que são chamadas de ***input*** ou simbolizadas pela letra `X`) e variáveis de saída (que são chamadas de ***output*** ou simbolizadas pela letra `y`) pareadas. Nesse caso, precisamos colocar os valores correspondentes juntos nos dados de treinamento e teste. No nosso exemplo, seria interessante predizer a receita da empresa (`y`) utilizando a temperatura média (`X`), então:

In [ ]:
```
# teste - input
X_teste = teste["Temperatura"]
# teste - output
y_teste = teste["Receita"]

# treino - input
X_treino = treino["Temperatura"]
# treino - output
y_treino = treino["Receita"]
```

Feito isso, veremos nos próximos capítulos que aplicaremos modelos da seguinte forma genérica:

In [ ]:
```
## NAO RODE!
## Isso e somente um esquema geral
# modelo = AlgumModelo()
# treinamento.modelo(X_treino, y_treino)
# performance = teste.modelo(X_teste, y_teste)
```

Se o modelo foi superajustado aos dados de treinamento, esperamos que ele tenha um desempenho muito ruim nos dados de teste. Em outras palavras, se ele tiver um bom desempenho nos dados de teste, você poderá ter mais confiança de que está se ajustando bem.

Performance

Como medir o quão bom é um modelo? Suponha um modelo de classificação binária (por exemplo, 'sim' ou 'não').

Dado um conjunto de dados rotulados e um modelo preditivo de classificação, cada dado está em uma das quatro categorias:

- Verdadeiro positivo: 'Esta mensagem é um \textit{spam} e previmos corretamente como \textit{spam}.'
- Falso positivo (erro tipo 1): 'Esta mensagem não é um \textit{spam}, mas previmos como \textit{spam}.'
- Falso negativo (erro tipo 2): 'Esta mensagem é um \textit{spam}, mas previmos que não era um \textit{spam}.'
- Verdadeiro negativo: 'Esta mensagem não é um \textit{spam} e previmos corretamente que não é um \textit{spam}.'

Comumente representamos isso como contagens em uma *matriz de confusão*:

	Spam	**Não Spam**
Predito Spam	Verdadeiro Positivo	Falso Positivo
Predito Não Spam	Falso Negativo	Verdadeiro Negativo

Um experimento realizado por uma grande empresa de servidores *online* avaliou 1 milhão de e-mails em busca de classificar quais deles eram *spams* através de modelos de classificação. Resultando assim na seguinte matriz de confusão:

	Spam	**Não Spam**	**Total**
Predito Spam	70	4.930	5.000
Predito Não Spam	13.930	981.070	995.000
Total	14.000	986.000	1.000.000

Podemos então usá-la para calcular várias estatísticas sobre o desempenho do modelo. Por exemplo, a **acurácia** é definida como a fração de previsões corretas:

In [ ]:
```
# Matriz de confusao
mat_confusao = pd.DataFrame(data =
                            {'vp': [70], # verdadeiro positivo
                             'fp': [4930], # falso positivo
                             'fn': [13930], # falso negativo
                             'vn': [986000]} # verdadeiro negativo
                            )

# Funcao que calcula a acuracia
def acuracia(matriz):
  correto = matriz["vp"] + matriz["vn"]
  total = matriz["vp"] + matriz["fp"] + matriz["fn"] + matriz["vn"]
  return (correto / total)[0]

# Acuracia
acuracia(mat_confusao)   # 0.981
```

Esse parece ser um número bastante impressionante. Mas com uma olhada mais atenta à tabela, não deveríamos dar muito crédito à precisão bruta (Por que?).

É comum observar a combinação de **precisão** e **sensibilidade**. A precisão mede quão acurada as previsões **positivas** foram:

In [ ]:
```
# Funcao que calcula a precisao
def precisao(matriz):
  return (matriz["vp"] / (matriz["vp"] + matriz["fp"]))[0]

# Precisao
precisao(mat_confusao)   # 0.014
```

E a sensibilidade mede qual fração dos **positivos** nosso modelo identificou:

In [ ]:
```
# Funcao que calcula a sensibilidade
def sensibilidade(matriz):
```

```
    return (matriz["vp"] / (matriz["vp"] + matriz["fn"]))[0]

# Sensibilidade
sensibilidade(mat_confusao)   # 0.005
```

Ambos são números terríveis, refletindo que este é um modelo ruim.

Às vezes, precisão e sensibilidade são combinados no **escore F1**, que é definido como:

In [ ]:
```
# Funcao que calcula o escore F1
def escore_f1(matriz):
    p = precisao(matriz)
    s = sensibilidade(matriz)
    return 2 * p * s / (p + s)

# Escore F1
escore_f1(mat_confusao)   # 0.007
```

Esta é a *média harmônica* de precisão e sensibilidade (e necessariamente está entre elas).

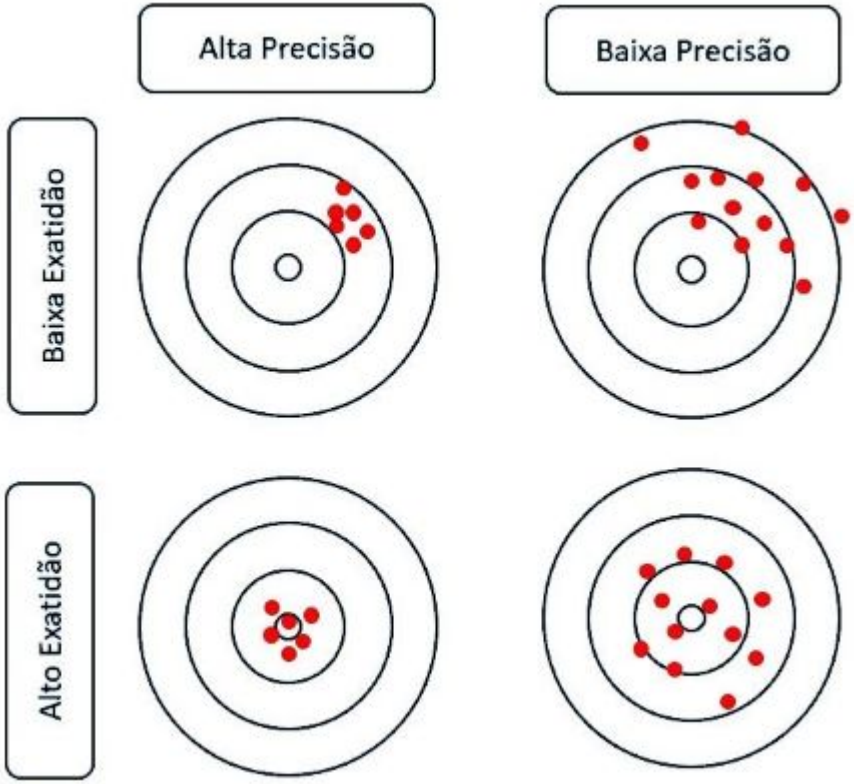
Normalmente, a escolha de um *bom* modelo envolve um equilíbrio entre precisão e sensibilidade. Predizer *sim* com muita frequência acarretará em muitos falsos positivos; predizer *não* com muita frequência acarretará em muitos falsos negativos.

Na vida real, cada caso é um caso. E se faz necessário identificar qual a melhor medida de performance para o modelo aplicado.

> Existem muitas nomenclaturas distintas para os elementos de uma matriz de confusão. Um breve resumo pode ser encontrado aqui.

Dilema Viés-variância

Outra maneira de pensar sobre o problema de sobreajuste é como um equilíbrio entre viés e variância.

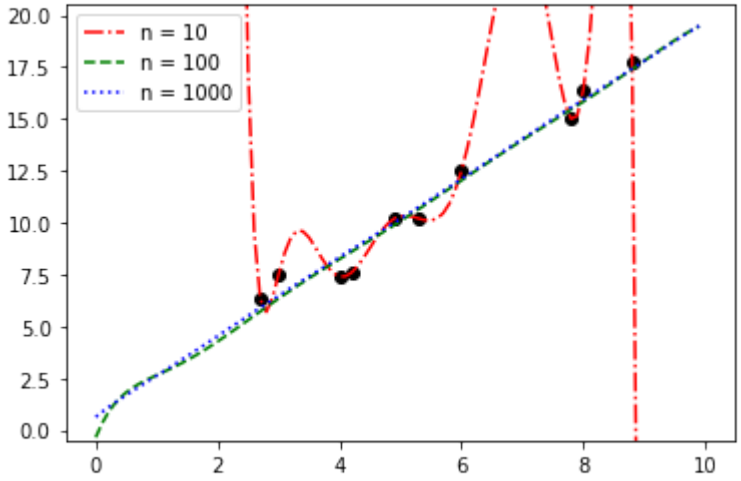


Por exemplo, o modelo linear de grau 0 que vimos na figura inicial cometerá muitos erros para praticamente qualquer conjunto de treinamento (ou seja, ele tem um viés alto). No entanto, qualquer subconjunto de treinamento escolhido aleatoriamente deve fornecer um modelo similar. Então dizemos que tem uma variância baixa. Alto viés e baixa variância geralmente correspondem a subajuste. Por outro lado, o modelo de grau 9 que se encaixou perfeitamente no conjunto de treinamento tem um viés muito baixo, mas uma variância muito alta (já que subconjuntos de treinamento provavelmente daria origem a um modelo bem diferente). Isso corresponde ao sobreajuste.

Pensar nos problemas de um modelo dessa maneira pode ajudá-lo a descobrir o que fazer quando seu modelo não funcionar tão bem. Se o seu modelo tiver um viés alto (o que significa que ele tem um desempenho ruim mesmo em seus dados de treinamento), uma coisa a tentar é adicionar mais atributos. Passar do modelo de grau 0 para o modelo de grau 1 foi uma grande melhoria.

Se o seu modelo tiver alta variação, você também poderá remover atributos. Mas outra solução é obter mais dados (se possível). Na figura abaixo, ajustamos um polinômio de grau 9 para amostras de tamanhos diferentes. O ajuste do modelo baseado em 10 dados passa

exatamente em cima de todos os pontos, como vimos antes. Se, ao invés disso, treinarmos em 100 dados, haverá muito menos sobreajuste. E repare que se o modelo for treinado a partir de 1000 dados, ele é muito semelhante ao modelo de grau 1 (uma reta).



Fixada a complexidade do modelo, quanto mais dados você tiver, mais difícil será o sobreajuste. Por outro lado, mais dados não o ajudarão quanto ao viés. Se o seu modelo não têm *atributos* suficientes para capturar regularidades nos dados, inserir mais dados não o melhorará.

Extração e Seleção de Atributos

Como mencionado, se o seu modelo não tem atributos suficientes, provavelmente ele será inadequado para a modelagem desse problema específico. Por outro lado, se seu modelo tem muitos atributos, provavelmente ele irá sobreajustar nos dados. Mas o que são atributos e de onde eles vêm?

A grosso modo, os atributos são entradas (*inputs*) que fornecemos ao nosso modelo.

Por exemplo, se você deseja prever o salário de alguém com base em seus anos de experiência, 'anos de experiência' é seu único atributo (embora vimos que você também pode considerar adicionar anos de experiência ao quadrado, ao cubo e assim por diante se isso ajuda a construir um modelo melhor.)

> Se dispomos de n pontos de dados. Um polinômio de grau $n-1$ irá passar exatamente sobre os n pontos. Como vimos na figura anterior, se temos $n = 10$ pontos, um polinômio de grau $n - 1 = 10 - 1 = 9$ passa exatamente estes pontos.

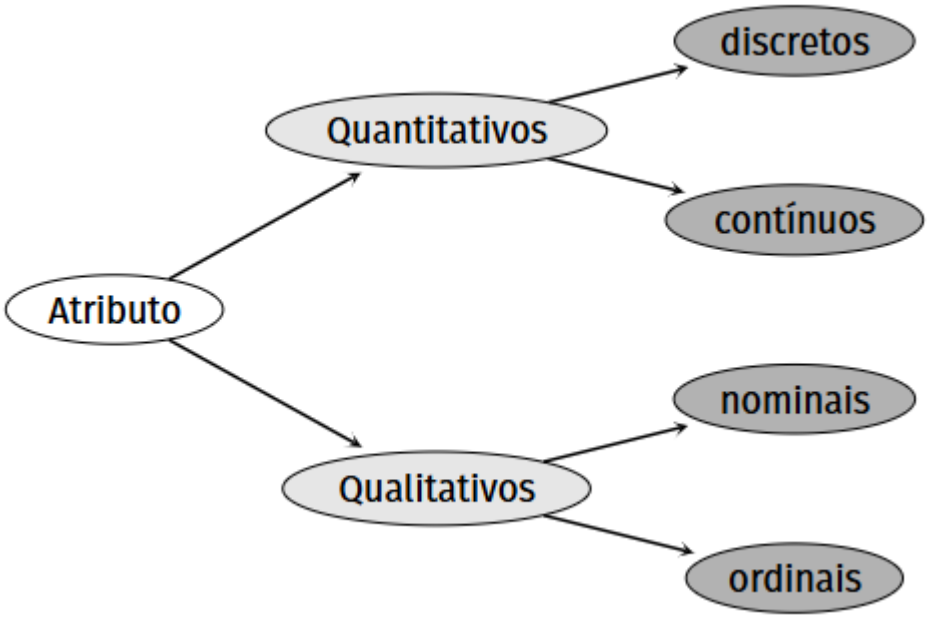
As coisas se tornam mais interessantes à medida que seus dados se tornam mais complicados. Imagine tentar construir um filtro de *spam* para prever se um e-mail é lixo ou não. A maioria dos modelos não sabe o que fazer com um e-mail bruto, que é apenas uma coleção de texto. Você terá que extrair atributos. Por exemplo:

- O e-mail contém a palavra 'promoção'?
- Quantas vezes a letra 'd' aparece?
- Quantos destinatários tem o e-mail?
- Qual era o domínio do remetente?

A resposta para a primeira pergunta é simplesmente `sim` ou `não`, que normalmente codificamos como `1` ou `0`. Para a segunda e terceira, é um número inteiro. E para a terceira, é um item de um conjunto discreto de opções.

Em geral, extraímos atributos de todo o banco de dados que dispomos. São os tipos de atributos que temos que guiarão qual tipo de modelo que podemos usar na solução do problema. Um atributo pode ser classificado como:

- **Quantitativos:** São características mensuráveis numericamente.
  - **Discretos:** Valores inteiros, resultado de contagens (e.g., número de filhos, bactérias).
  - **Contínuos:** Valores em uma escala contínua (e.g., peso, altura, tempo).
- **Qualitativos (ou categóricos):** Características definidas por categorias.
  - **Nominais:** Sem ordem entre as categorias (e.g., sexo, cor dos olhos).
  - **Ordinais:** Com ordem entre as categorias (e.g., escolaridade, estágio da doença).



> Dependendo do contexto, um *atributo* (*feature*) é também chamado de *variável de entrada* (*input*), *variável preditora*, *regressora*, *covariável* ou *variável explicativa*.

Como escolher atributos? É aí que entra uma combinação de experiência e conhecimento do domínio do problema. Se você recebeu muitos e-mails, provavelmente tem a sensação de que a presença de certas palavras pode ser um bom indicador de *spam*. E você também pode ter a sensação de que o número de 'd's provavelmente não é um bom indicador de spam. Mas, em geral, você terá que experimentar coisas diferentes, o que faz parte de seu aprendizado.