Machine Learning

O que é Machine Learning?

O aprendizado de máquina (ML, na sigla em inglês) capacita algumas das tecnologias mais importantes que usamos, de apps de tradução a veículos autônomos. Este curso explica os principais conceitos por trás do ML.

O ML oferece uma nova maneira de resolver problemas, responder a perguntas complexas e criar novo conteúdo. O ML pode prever o clima, estimar o tempo de viagem, recomendar músicas, autocompletar frases, resumir artigos e gerar imagens nunca vistas antes.

Em termos básicos, o ML é o processo de [treinamento](https://developers.google.com/machine-learning/glossary?hl=pt-br#training) de um software, chamado [modelo](https://developers.google.com/machine-learning/glossary?hl=pt-br#model), para fazer [previsões](https://developers.google.com/machine-learning/glossary?hl=pt-br#prediction) úteis ou gerar conteúdo de dados.

Por exemplo, vamos criar um aplicativo para prever precipitação. Poderíamos usar uma abordagem tradicional ou uma abordagem de ML. Usando uma abordagem tradicional, criaríamos uma representação física da atmosfera e da superfície da Terra, calculando grandes quantidades de equações de dinâmica fluida. Isso é incrivelmente difícil.

Usando uma abordagem de ML, daríamos a um modelo de ML enormes quantidades de dados meteorológicos até o modelo de ML aprender a relação matemática entre padrões climáticos que produzem quantidades diferentes de chuva. Nesse caso, forneceríamos ao modelo os dados meteorológicos atuais e a previsão da quantidade de chuva.

Um modelo é uma relação matemática derivada de dados que um sistema de ML usa para fazer predições.

**Tipos de sistemas de ML**

Os sistemas de ML se enquadram em uma ou mais das seguintes categorias com base em como aprendem a fazer previsões ou gerar conteúdo:

* Aprendizado supervisionado
* Aprendizado não supervisionado
* Aprendizado por reforço
* IA generativa

**Aprendizado supervisionado**

Os modelos de [aprendizado supervisionado](https://developers.google.com/machine-learning/glossary?hl=pt-br#supervised-machine-learning) fazem previsões depois de ver muitos dados com as respostas corretas e, em seguida, descobrir as conexões entre os elementos nos dados que produzem as respostas corretas. É como um aluno que aprende novos materiais estudando exames antigos que contêm perguntas e respostas. Depois que é treinado em exames antigos, ele já está preparado para fazer um novo exame. Esses sistemas de ML são "supervisionados", no sentido de que uma pessoa fornece dados do sistema de ML com os resultados corretos conhecidos.

Dois dos casos de uso mais comuns para aprendizado supervisionado são regressão e classificação.

**Regressão**

Um [modelo de regressão](https://developers.google.com/machine-learning/glossary?hl=pt-br#regression-model) prevê um valor numérico. Por exemplo, um modelo meteorológico que prevê a quantidade de chuva, em polegadas ou milímetros, é um modelo de regressão.

Consulte a tabela abaixo para ver mais exemplos de modelos de regressão:

| **Cenário** | **Possíveis dados de entrada** | **Previsão numérica** |
| --- | --- | --- |
| Preço da casa futura | Áreas quadradas, CEP, número de quartos e banheiros, tamanho do terreno, taxa de juros hipotecários, taxa de impostos da propriedade, custos de construção e número de casas à venda na área. | O preço da casa. |
| Tempo de viagem futura | Condições históricas do trânsito (coletadas por smartphones, sensores de tráfego, transporte por aplicativo e outros aplicativos de navegação), distância do destino e condições climáticas. | O tempo em minutos e segundos para chegar a um destino. |

**Classificação**

Os [modelos de classificação](https://developers.google.com/machine-learning/glossary?hl=pt-br#classification-model) preveem a probabilidade de algo pertencer a uma categoria. Ao contrário dos modelos de regressão, em que a saída é um número, os modelos de classificação geram um valor que indica se algo pertence ou não a uma determinada categoria. Por exemplo, os modelos de classificação são usados para prever se um e-mail é spam ou se uma foto contém um gato.

Os modelos de classificação são divididos em dois grupos: classificação binária e classificação multiclasse. Os modelos de classificação binária geram um valor de uma classe que contém apenas dois valores. Por exemplo, um modelo que gera rain ou no rain. Os modelos de classificação multiclasse produzem um valor de uma classe que contém mais de dois valores. Por exemplo, um modelo que pode gerar rain, hail, snow ou sleet.

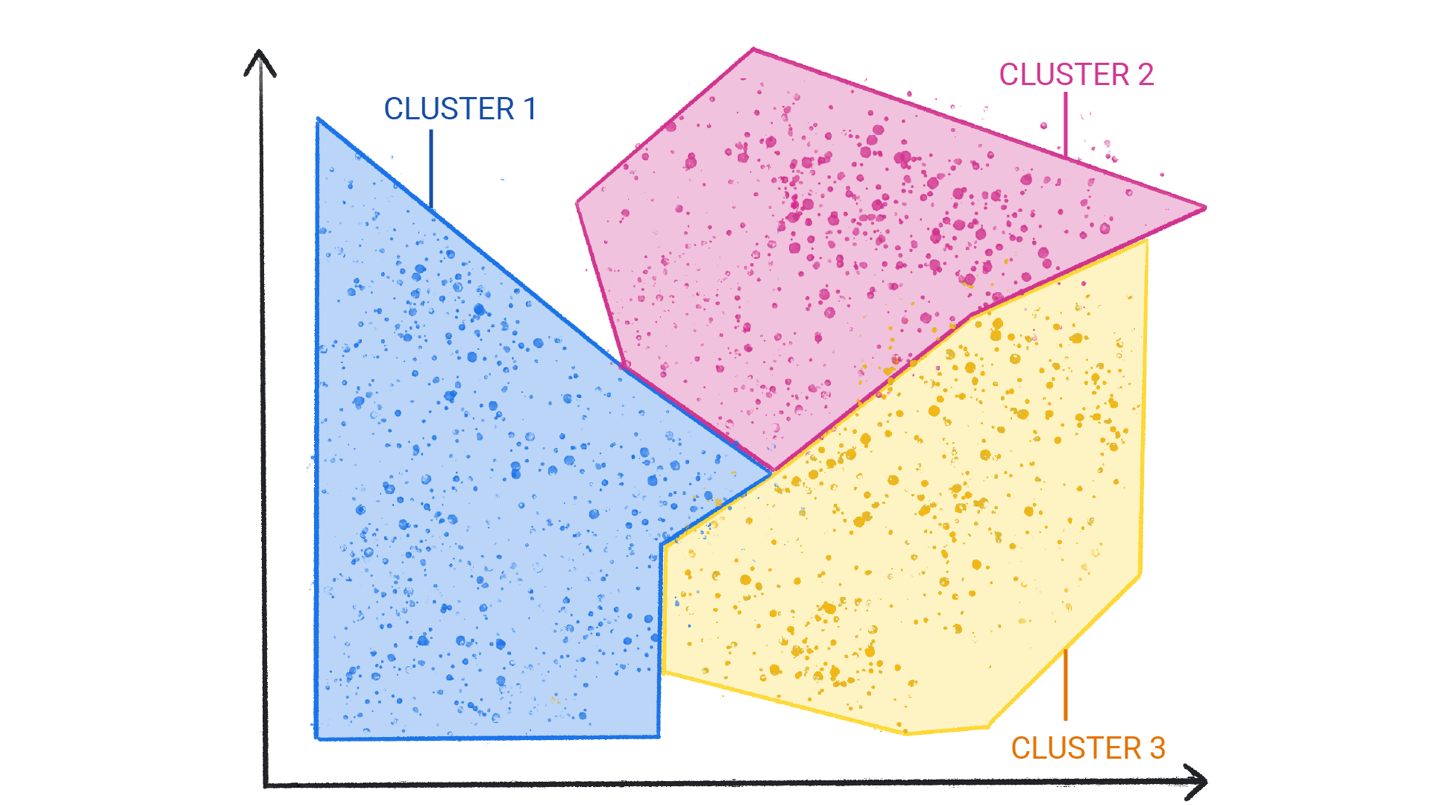
## **Aprendizado não supervisionado**

Modelos de [aprendizado não supervisionado](https://developers.google.com/machine-learning/glossary?hl=pt-br#unsupervised-machine-learning) fazem previsões por receber dados que não contêm respostas corretas. A meta de um modelo de aprendizado não supervisionado é identificar padrões significativos entre os dados. Em outras palavras, o modelo não tem dicas sobre como categorizar cada dado, mas precisa inferir as próprias regras.

Um modelo de aprendizado não supervisionado comumente usado emprega uma técnica chamada [clustering](https://developers.google.com/machine-learning/glossary?hl=pt-br#clustering). O modelo encontra pontos de dados que demarcam agrupamentos naturais.

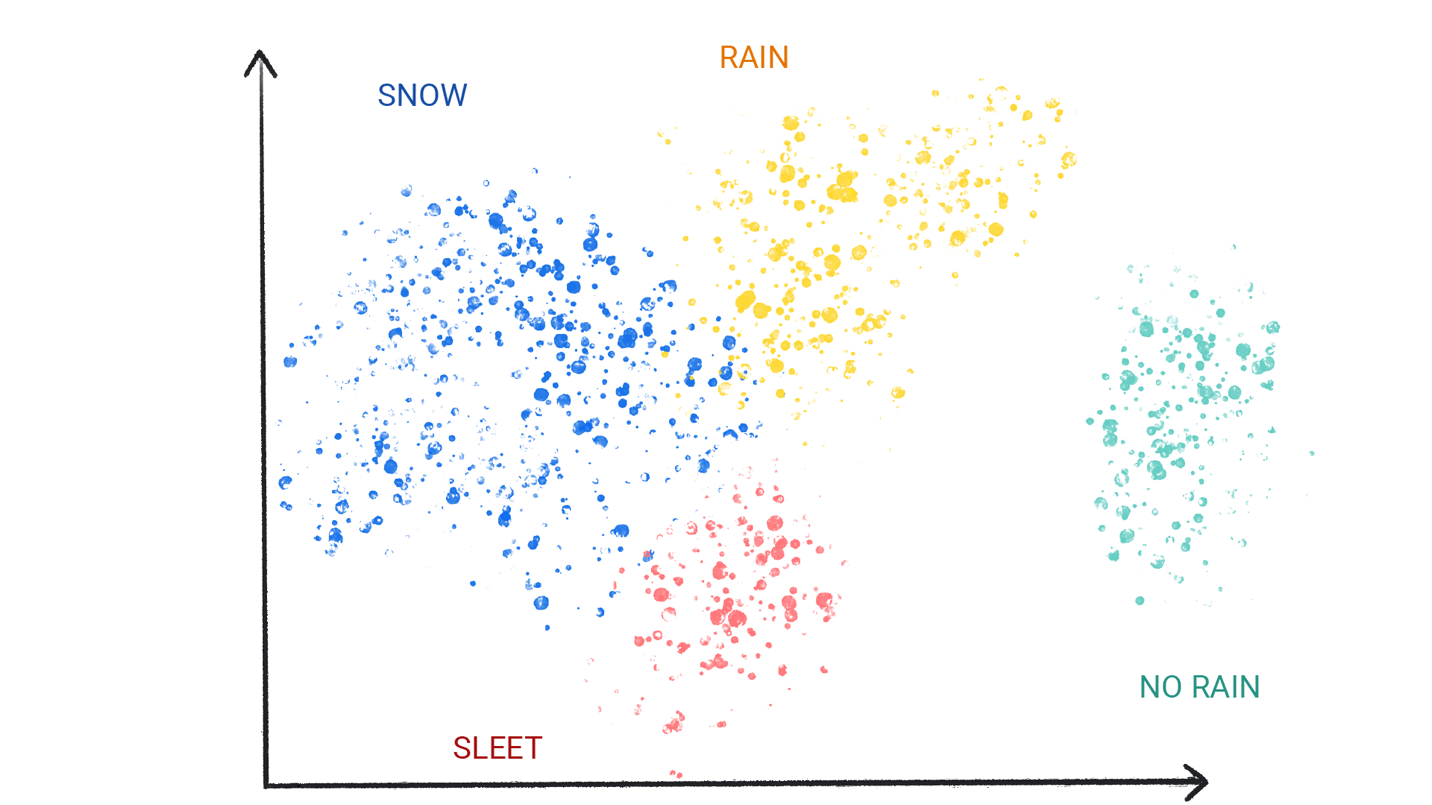


**Figura 1**. Um modelo de ML agrupando pontos de dados semelhantes.



**Figura 2**. Grupos de clusters com demarcações naturais.

O clustering difere da classificação porque as categorias não são definidas por você. Por exemplo, um modelo não supervisionado pode agrupar um conjunto de dados meteorológicos com base na temperatura, revelando segmentações que definem as estações do ano. Em seguida, você pode tentar nomear esses clusters com base na compreensão do conjunto de dados.



**Figura 3**. Um modelo de ML agrupando padrões climáticos semelhantes.



**Figura 4**. Clusters de padrões climáticos rotulados como neve, granizo, chuva e chuva.

## **Aprendizado por reforço**

Os modelos de [aprendizado por reforço](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "reinforcement-learning-rl) fazem previsões por meio de [prêmios](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "reward) ou penalidades com base nas ações realizadas em um ambiente. Um sistema de aprendizado por reforço gera uma [política](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "policy) que define a melhor estratégia para conseguir o máximo de recompensas.

O aprendizado por reforço é usado para treinar robôs a realizar tarefas, como andar ao redor de uma sala, e programas de software, como o [AlphaGo](https://deepmind.com/research/case-studies/alphago-the-story-so-far), para jogar o Go.

## **IA generativa**

A [IA generativa](https://developers.google.com/machine-learning/glossary?hl=pt-br#generative-ai) é uma classe de modelos que cria conteúdo a partir da entrada do usuário. Por exemplo, a IA generativa pode criar novas imagens, composições musicais e piadas. Ela pode resumir artigos, explicar como executar uma tarefa ou editar uma foto.

A IA generativa pode ter várias entradas e criar diversas saídas, como texto, imagens, áudio e vídeo. Ele também pode criar combinações. Por exemplo, um modelo pode usar uma imagem como entrada e criar uma imagem e um texto como saída ou criar uma imagem e um texto como entrada e criar um vídeo como saída.

É possível discutir modelos generativos pelas entradas e saídas, normalmente escritas como "tipo de entrada" para "tipo de saída". Por exemplo, veja a seguir uma lista parcial de algumas entradas e saídas para modelos generativos:

* Texto para texto
* Texto para imagem
* Texto para vídeo
* Conversão de texto em código
* Conversão de texto em voz
* Imagem e texto para imagem

A tabela a seguir lista exemplos de modelos generativos, as respectivas entradas e um exemplo da possível saída:

| **Modelo** | **Entrada** | **Exemplo de saída** |
| --- | --- | --- |
| Texto para texto | Quem inventou a corrida de Le Mans? | A 24 Hours de Le Mans foi fundada pela Automobile Club de l'Ouest (ACO), uma organização automobilística fundada em 1906. A primeira corrida foi realizada nos dias 26 e 27 de maio de 1923, no Circuito Sarthe, um circuito rodoviário de 13.426 quilômetros localizado perto da cidade de Le Mans, na França. Originalmente, a corrida foi criada como uma forma de promover a ACO e a região de Sarthe. Fonte: [Bard](https://bard.google.com/?hl=pt-br) |
| Texto para imagem | Um polvo alienígena flutua por um portal que lê um jornal. | Imagem de um polvo de desenho animado. Fonte: [Imagen](https://imagen.research.google/?hl=pt-br) |
| Texto para vídeo | Um urso de pelúcia fotorrealista nadando no mar em São Francisco. O urso de pelúcia vai debaixo d'água. O urso de pelúcia continua nadando embaixo d'água com peixes coloridos. Um urso-panda nadando debaixo d'água. | Vídeo de um urso de pelúcia nadando debaixo d&#39;água. Fonte: [Phenaki](https://phenaki.video/) |
| Conversão de texto em código | Escreva um loop do Python que passa por uma lista de números e imprime os números primos. | for number in numbers: # Check if the number is prime. is\_prime = True for i in range(2, number):   if number % i == 0:       is\_prime = False       break # If the number is prime, print it. if is\_prime:   print(number)  Fonte: [Bard](https://bard.google.com/?hl=pt-br) |
| Conversão de imagem em texto | Imagem de um flamingo. | Este é um flamingo. Eles são encontrados no Caribe. Fonte: [Google DeepMind](https://www.deepmind.com/blog/tackling-multiple-tasks-with-a-single-visual-language-model) |

Como funciona a IA generativa? Em um nível geral, modelos generativos aprendem padrões em dados com o objetivo de produzir dados novos, mas semelhantes. Os modelos generativos têm a seguinte aparência:

* Comediantes que aprendem a imitar os outros observando os comportamentos e o estilo das pessoas
* Artistas que aprendem a pintar em um estilo específico estudando muitas pinturas nesse estilo
* Covers que aprendem a soar como um grupo musical específico ouvindo muitas músicas desse grupo

Para produzir saídas únicas e criativas, os modelos generativos são inicialmente treinados usando uma abordagem não supervisionada, em que o modelo aprende a imitar os dados em que é treinado. Às vezes, o modelo é treinado mais detalhadamente usando aprendizado supervisionado ou de reforço em dados específicos relacionados a tarefas que o modelo pode ter que realizar, por exemplo, resumir um artigo ou editar uma foto.

A IA generativa é uma tecnologia em rápida evolução, com novos casos de uso sendo descobertos constantemente. Por exemplo, os modelos generativos estão ajudando as empresas a refinar as imagens de produtos de comércio eletrônico removendo automaticamente os planos de fundo que causam distração ou melhorando a qualidade das imagens de baixa resolução.

# **Aprendizado supervisionado**

As tarefas de aprendizado supervisionado são bem definidas e podem ser aplicadas a vários cenários, como identificar spam ou prever precipitação. O aprendizado supervisionado usa dados históricos melhor do que o de reforço.

## **Conceitos fundamentais de aprendizado supervisionado**

O machine learning supervisionado é baseado nos seguintes conceitos principais:

* Dados
* Modelo
* Treinamento
* Avaliando
* Inferência

**Dados**

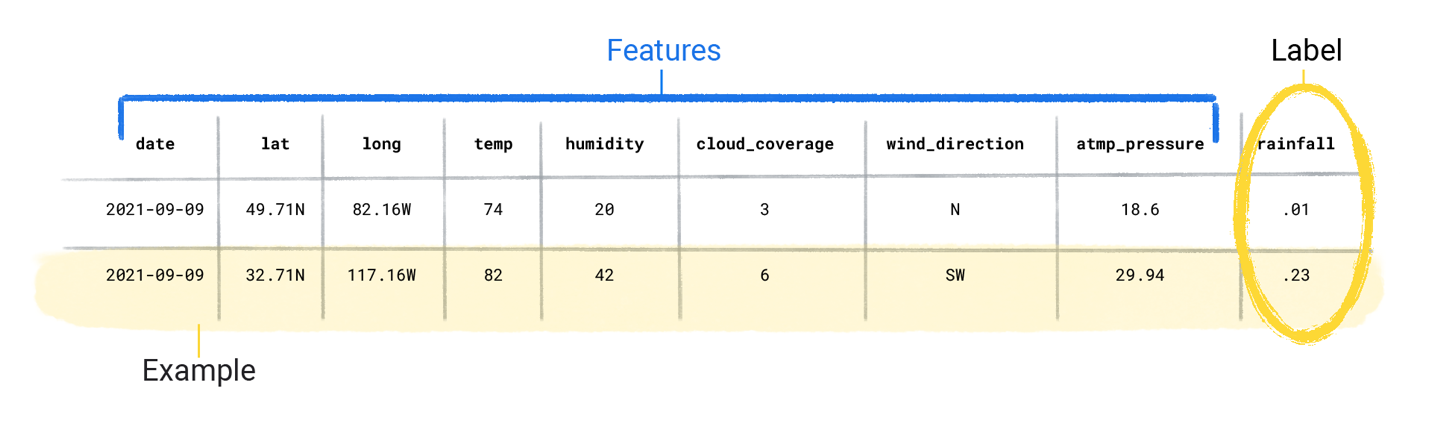
Os dados são a força motriz do ML. Os dados vêm na forma de palavras e números armazenados em tabelas ou como os valores de pixels e formas de onda capturados em arquivos de imagem e áudio. Armazenamos dados relacionados em conjuntos de dados. Por exemplo, podemos ter um conjunto de dados do seguinte tipo:

* Imagens de gatos
* Preços de imóveis
* Informações sobre a previsão do tempo

Os conjuntos de dados são compostos por [exemplos](https://developers.google.com/machine-learning/glossary?hl=pt-br#example) individuais que contêm [recursos](https://developers.google.com/machine-learning/glossary?hl=pt-br#feature) e um [rótulo](https://developers.google.com/machine-learning/glossary?hl=pt-br#label). Você pode pensar em um exemplo análogo a uma única linha em uma planilha. Recursos são os valores que um modelo supervisionado usa para prever o rótulo. O rótulo é a "resposta" ou o valor que queremos que o modelo preveja. Em um modelo de previsão do tempo de chuva, os atributos podem ser *latitude*, *longitude*, *temperatura*, *umidade*, *cobertura da nuvem*, *direção do vento* e *pressão atmosférica*. O rótulo seria *quantidade de precipitação*.

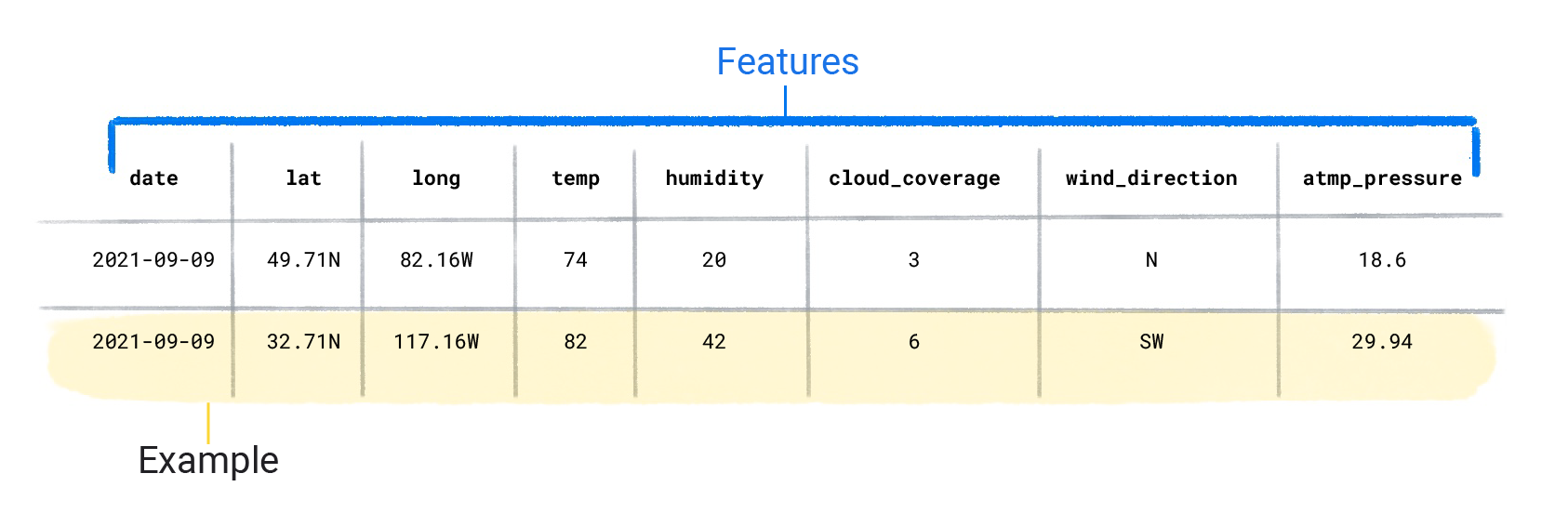
Os exemplos que contêm recursos e um rótulo são chamados de [exemplos rotulados](https://developers.google.com/machine-learning/glossary?hl=pt-br#labeled-example).

**Dois exemplos rotulados**



Por outro lado, os exemplos sem rótulos contêm atributos, mas nenhum rótulo. Depois de criar um modelo, ele prevê o rótulo a partir dos recursos.

**Dois exemplos sem rótulos**



### Características do conjunto de dados

Um conjunto de dados é caracterizado por seu tamanho e diversidade. O tamanho indica o número de exemplos. A diversidade indica a abrangência desses exemplos. Bons conjuntos de dados são grandes e diversificados.

Alguns conjuntos de dados são grandes e diversificados. No entanto, alguns conjuntos de dados são grandes, mas têm baixa diversidade, e alguns são pequenos, mas altamente diversificados. Em outras palavras, um conjunto de dados grande não garante diversidade suficiente, e um conjunto de dados altamente diversificado não garante exemplos suficientes.

Por exemplo, um conjunto de dados pode conter 100 anos de dados, mas apenas para o mês de julho. O uso desse conjunto de dados para prever a chuva em janeiro produziria previsões ruins. Por outro lado, um conjunto de dados pode abranger apenas alguns anos, mas conter todos os meses. Esse conjunto de dados pode produzir previsões ruins porque não contém anos suficientes para explicar a variabilidade.

Um conjunto de dados também pode ser caracterizado pelo número de atributos. Por exemplo, alguns conjuntos de dados meteorológicos podem conter centenas de recursos, desde imagens de satélite a valores de cobertura de nuvens. Outros conjuntos de dados podem conter apenas três ou quatro atributos, como umidade, pressão atmosférica e temperatura. Conjuntos de dados com mais recursos podem ajudar um modelo a descobrir padrões adicionais e fazer previsões melhores. No entanto, conjuntos de dados com mais recursos nem *sempre* produzem modelos que fazem previsões melhores, porque alguns recursos podem não ter relação causal com o rótulo.

### Modelo

No aprendizado supervisionado, um modelo é o conjunto complexo de números que definem a relação matemática de padrões de recursos de entrada específicos a valores de rótulos de saída específicos. O modelo descobre esses padrões por meio do treinamento.

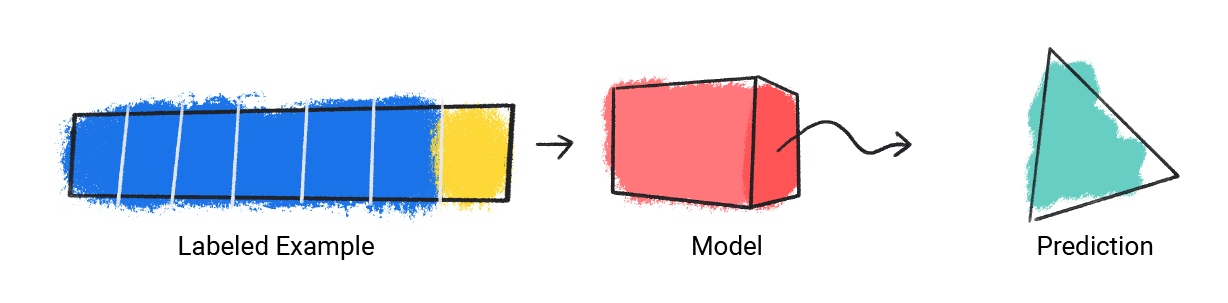
**Treinamento**

Para que um modelo supervisionado possa fazer previsões, ele precisa ser treinado. Para treinar um modelo, fornecemos a ele um conjunto de dados com exemplos rotulados. O objetivo do modelo é desenvolver a melhor solução para prever os rótulos dos atributos. O modelo encontra a melhor solução comparando o valor previsto com o valor real do rótulo. Com base na diferença entre os valores previstos e reais, definidos como a [perda](https://developers.google.com/machine-learning/glossary?hl=pt-br#loss), o modelo atualiza gradualmente a solução. Em outras palavras, o modelo aprende a relação matemática entre os atributos e o rótulo para que ele possa fazer as melhores previsões sobre dados não vistos.

Por exemplo, se o modelo previsse 1.15 inches de chuva, mas o valor real fosse .75 inches, o modelo modificará a solução para que a previsão seja mais próxima de .75 inches. Depois que o modelo analisa cada exemplo no conjunto de dados (em alguns casos, várias vezes), ele chega a uma solução que faz as melhores previsões, em média, para cada um dos exemplos.

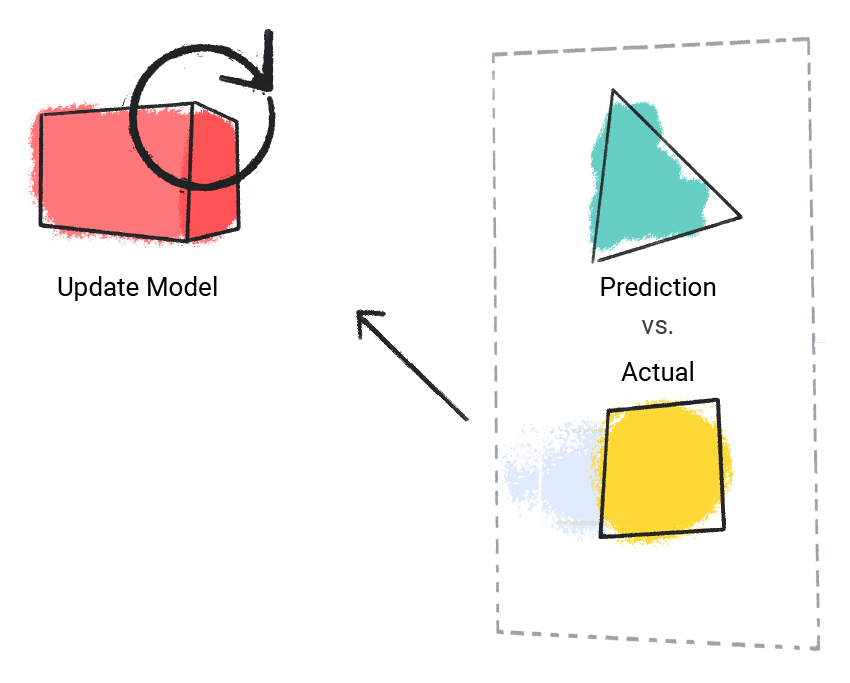
Veja a seguir como treinar um modelo:

1. O modelo recebe um único exemplo rotulado e fornece uma previsão.



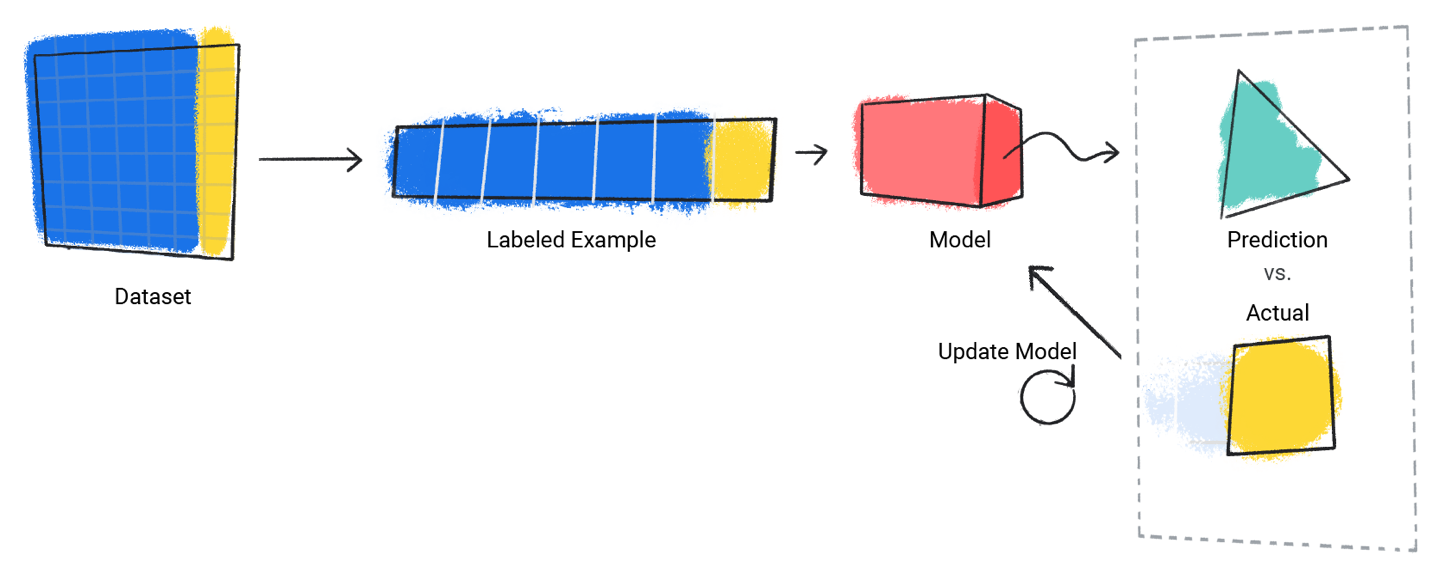
**Figura 1**. Um modelo de ML que faz uma previsão de um exemplo rotulado.

O modelo compara o valor previsto com o valor real e atualiza a solução.



**Figura 2**. Um modelo de ML que atualiza o valor previsto.

O modelo repete esse processo para cada exemplo rotulado no conjunto de dados.

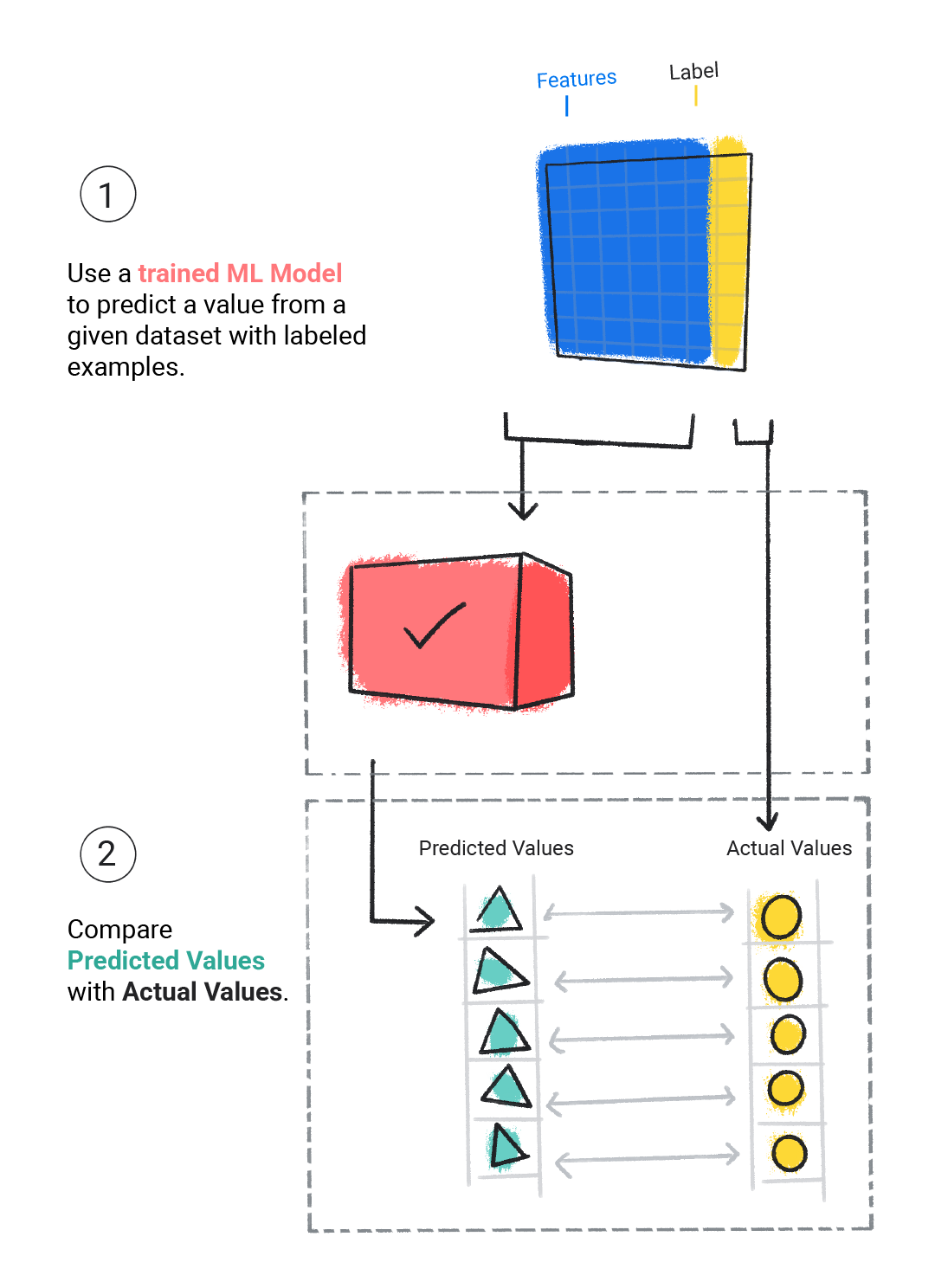


**Figura 3**. Um modelo de ML que atualiza as previsões para cada exemplo rotulado no conjunto de dados de treinamento.

Dessa forma, o modelo aprende gradualmente a relação correta entre os atributos e o rótulo. Essa compreensão gradual também é o motivo pelo qual conjuntos de dados grandes e diversos produzem um modelo melhor. O modelo viu mais dados com uma variedade maior de valores e refinou a compreensão da relação entre os atributos e o rótulo.

Durante o treinamento, os profissionais de ML podem fazer ajustes sutis às configurações e aos recursos que o modelo usa para fazer previsões. Por exemplo, alguns recursos têm mais poder preditivo do que outros. Portanto, os profissionais de ML podem selecionar quais atributos o modelo usa durante o treinamento. Por exemplo, suponha que um conjunto de dados meteorológicos contenha time\_of\_day como atributo. Nesse caso, um profissional de ML pode adicionar ou remover time\_of\_day durante o treinamento para ver se o modelo faz previsões melhores com ou sem ele.

### Avaliando

Avaliamos um modelo treinado para determinar se ele aprendeu muito bem. Quando avaliamos um modelo, usamos um conjunto de dados rotulado, mas fornecemos ao modelo apenas os atributos do conjunto de dados. Em seguida, comparamos as previsões do modelo com os valores verdadeiros do rótulo.  
**Figura 4**. Avaliar um modelo de ML comparando as previsões com os valores reais.

Dependendo das previsões do modelo, podemos fazer mais treinamento e avaliação antes de implantar o modelo em um aplicativo real

### Inferência

Quando estivermos satisfeitos com os resultados da avaliação do modelo, poderemos usá-lo para fazer predições, chamadas de [inferências](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "inference), em exemplos sem rótulos. No exemplo do app de clima, forneceríamos ao modelo as condições climáticas atuais, como temperatura, pressão atmosférica e umidade relativa, e ele prevê a quantidade de precipitação.

Como criar: principais terminologias de ML

O que é machine learning (supervisionado)? Em resumo, o seguinte é:

* Os sistemas de ML aprendem a combinar entradas para produzir previsões úteis sobre dados nunca vistos antes.

Vamos explorar a terminologia fundamental de machine learning.

## **Identificadores**

Um **marcador** é o que estamos prevendo, a variável y na regressão linear simples. O rótulo pode ser o preço futuro de trigo, o tipo de animal mostrado em uma imagem, o significado de um clipe de áudio ou qualquer outro item.

## **Recursos**

## Um **recurso** é uma variável de entrada: a variável x na regressão linear simples. Um projeto de machine learning simples pode usar um único recurso, enquanto um projeto de machine learning mais sofisticado pode usar milhões de atributos, especificados como: X1, X2,...Xn.

No exemplo do detector de spam, os recursos podem incluir o seguinte:

* palavras no texto do e-mail
* endereço do remetente
* hora do dia em que o e-mail foi enviado
* o e-mail contém a frase "um truque estranho".

## **Exemplos**

Um **exemplo** é uma instância específica de dados, **x**. Colocamos **x** em negrito para indicar que é um vetor. Dividimos os exemplos em duas categorias:

* exemplos rotulados
* exemplos sem rótulos

Um **exemplo rotulado** inclui os recursos e o rótulo. Ou seja:

  labeled examples: {features, label}: (x, y)

Use exemplos rotulados para **treinar** o modelo. Em nosso exemplo do detector de spam, os exemplos rotulados seriam e-mails individuais que os usuários marcaram explicitamente como "quot;spam" ou "não é spam".

Por exemplo, a tabela a seguir mostra cinco exemplos rotulados de um [conjunto de dados](https://developers.google.com/machine-learning/crash-course/california-housing-data-description?hl=pt-br) que contém informações sobre preços de imóveis na Califórnia:

|  |  |  |  |
| --- | --- | --- | --- |
| **habitMedianAge (recurso)** | **totalRooms (recurso)** | **totalBedrooms (recurso)** | **medianHouseValue (rótulo)** |
| 15 | 5612 | 1283 | 66900 |
| 19 | 7650 | 1901 | 80100 |
| 17 | 720 | 174 | 85700 |
| 14 | 1501 | 337 | 73400 |
| 20 | 1454 | 326 | 65500 |

Um **exemplo não rotulado** contém recursos, mas não o rótulo. Ou seja:

  unlabeled examples: {features, ?}: (x, ?)

Veja a seguir três exemplos não rotulados do mesmo conjunto de dados de imóveis, que excluem medianHouseValue:

|  |  |  |
| --- | --- | --- |
| **habitMedianAge (recurso)** | **totalRooms (recurso)** | **totalBedrooms (recurso)** |
| 42 | 1686 | 361 |
| 34 | 1226 | 180 |
| 33 | 1077 | 271 |

Depois de treinarmos nosso modelo com exemplos rotulados, usamos esse modelo para prever o rótulo em exemplos não rotulados. No detector de spam, os exemplos não rotulados são novos e-mails que ainda não foram identificados por humanos.

## **Modelos**

Um modelo define a relação entre os atributos e o rótulo. Por exemplo, um modelo de detecção de spam pode associar determinados recursos a "quot;spam"". Vamos destacar duas fases da vida de um modelo:

* **Treinamento** significa criar ou **aprender** o modelo. Ou seja, você mostra exemplos do modelo rotulado e permite que o modelo aprenda gradativamente as relações entre os atributos e o rótulo.
* **Inferência** significa aplicar o modelo treinado a exemplos não rotulados. Ou seja, use o modelo treinado para fazer previsões úteis (y'). Por exemplo, durante a inferência, é possível prever medianHouseValue para novos exemplos não rotulados.

## **regressão x classificação**

Um modelo de **regressão** prevê valores contínuos. Por exemplo, os modelos de regressão fazem previsões que respondem a perguntas como as seguintes:

* Qual é o valor de uma casa na Califórnia?
* Qual é a probabilidade de um usuário clicar nesse anúncio?

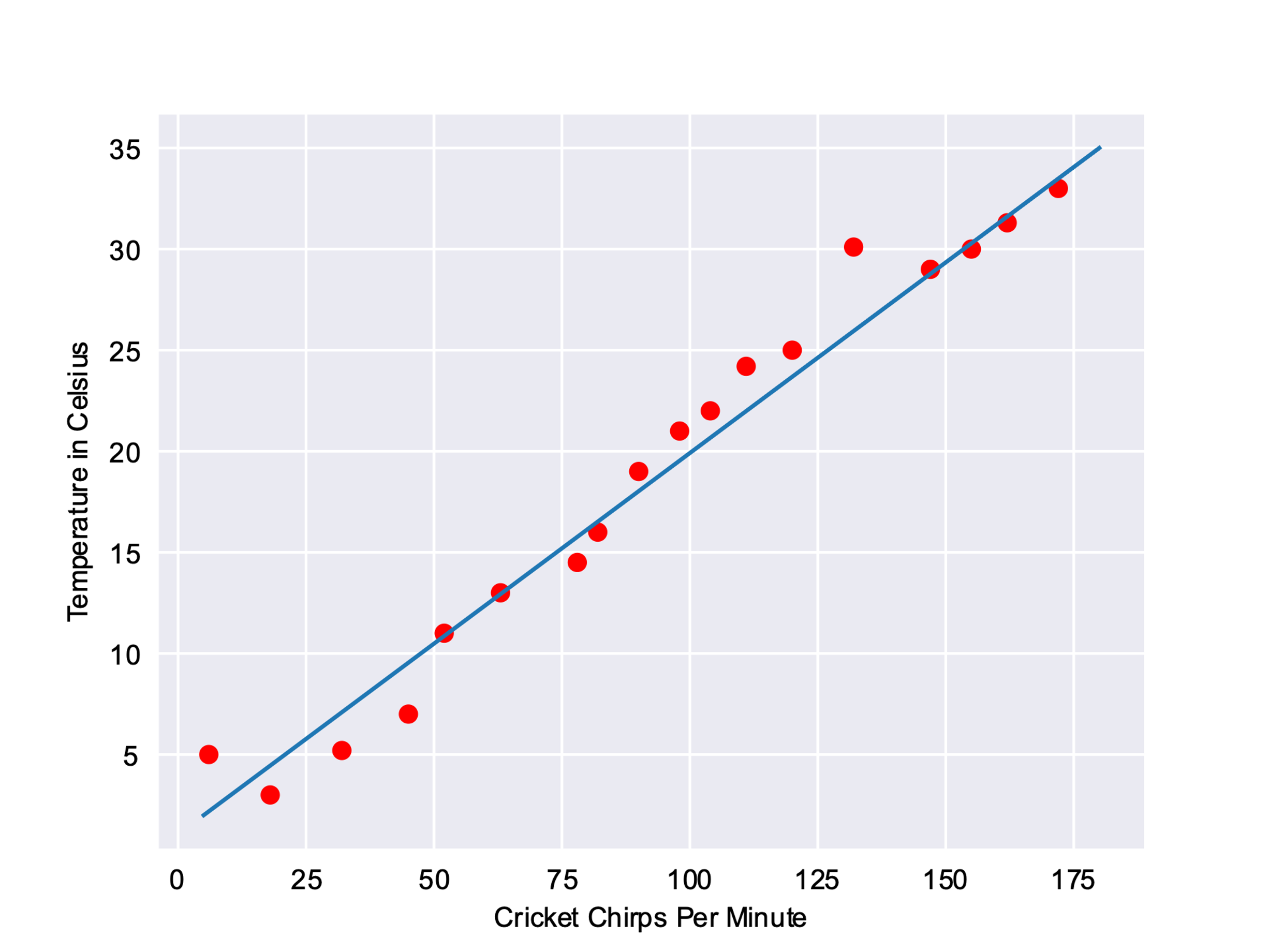
Um modelo de **classificação** prevê valores discretos. Por exemplo, os modelos de classificação fazem previsões que respondem a perguntas como as seguintes:

* Uma determinada mensagem de e-mail é spam ou não é spam?
* Esta é uma imagem de um cachorro, um gato ou um hamster?

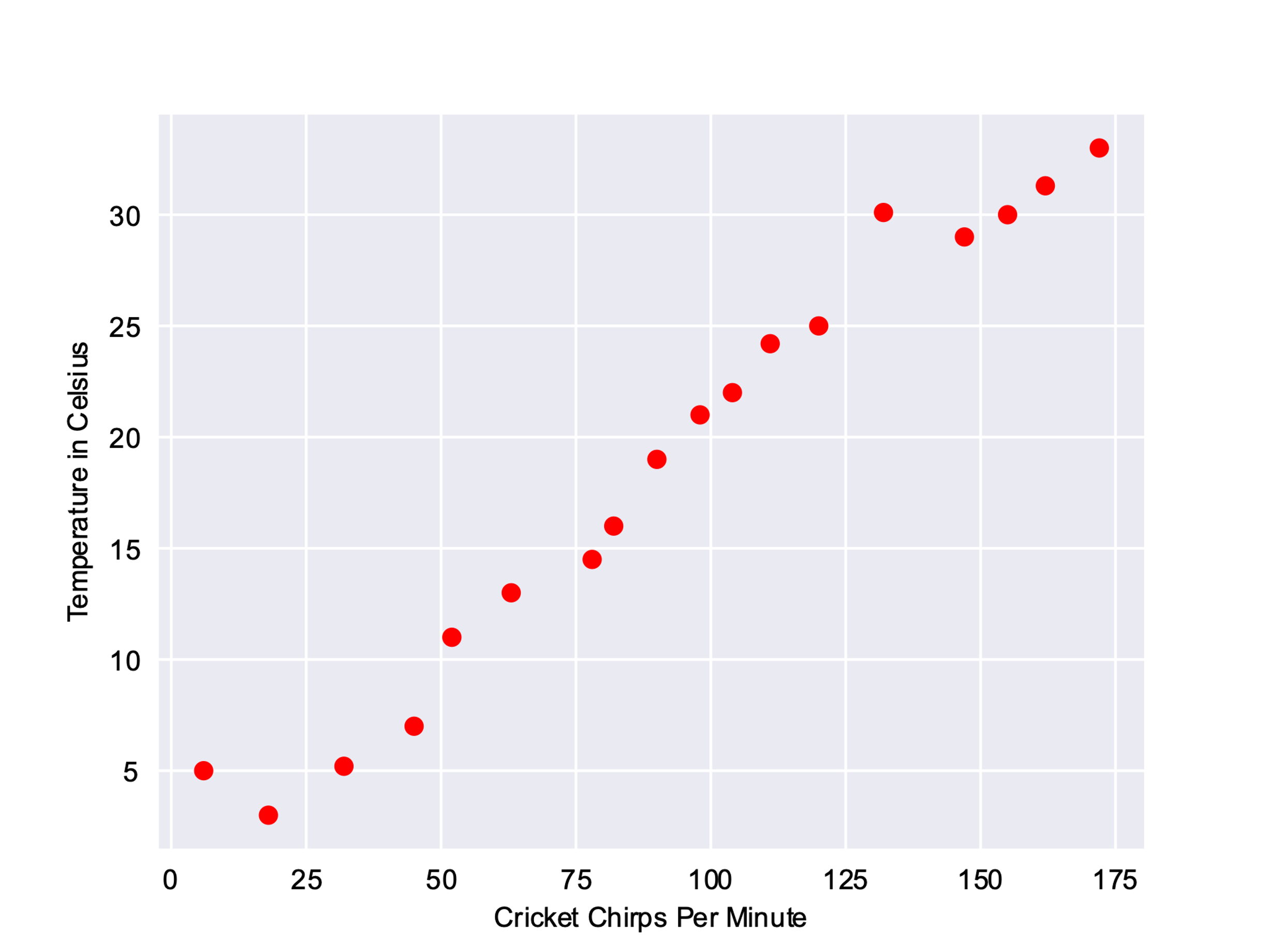
Redução no ML: regressão linear

Há muito tempo, sabe-se que os críquetes (uma espécie de inseto) chiram com mais frequência em dias mais quentes do que em dias mais frios. Por décadas, cientistas profissionais e amadores catalogam dados em chirps por minuto e temperatura. Como um presente de aniversário, sua tia Ruth oferece a ela um banco de dados de críquete e pede que você aprenda um modelo para prever esse relacionamento. Usando esses dados, você quer explorar essa relação.

Primeiro, examine os dados traçando-os:



**Figura 1. Chirps por minuto x temperatura em Celsius.**

Como esperado, o gráfico mostra a temperatura crescente com o número de toques. Esta relação entre chirps e temperatura linear? Sim, é possível desenhar uma única linha reta como a seguinte para aproximar essa relação:

**Figura 2. Uma relação linear.**

Verdadeiro, a linha não passa por todos os pontos, mas ela mostra claramente a relação entre chirps e temperatura. Usando a equação de uma linha, é possível escrever essa relação da seguinte maneira:

Y = mx + b

onde:

* y é a temperatura em Celsius, o valor que estamos tentando prever.
* m é a inclinação da linha.
* x é o número de toques por minuto, que é o valor do atributo de entrada.
* �bé a interseção em y.

Por convenção em machine learning, você escreverá a equação para um modelo um pouco diferente:

y′ = b + w1x1

onde:

* y′ é o [rótulo](https://developers.google.com/machine-learning/crash-course/framing/ml-terminology?hl=pt-br" \l "labels) previsto (uma saída esperada).
* b é o viés (a interseção em y), às vezes chamado de �0.
* w1 é o peso do atributo 1. Peso é o mesmo conceito da "inclinação" � na equação tradicional de uma linha.
* x1 é um [recurso](https://developers.google.com/machine-learning/crash-course/framing/ml-terminology?hl=pt-br" \l "features) (uma entrada conhecida).

Para **inferir** (prever) a temperatura y′ por um novo valor de Chips por minuto x1, basta substituir o valor x1 no modelo.

Embora esse modelo use apenas um recurso, um modelo mais sofisticado pode depender de vários atributos, cada um com um peso separado (�1, �2etc.). Por exemplo, um modelo que depende de três recursos pode ter a seguinte aparência:

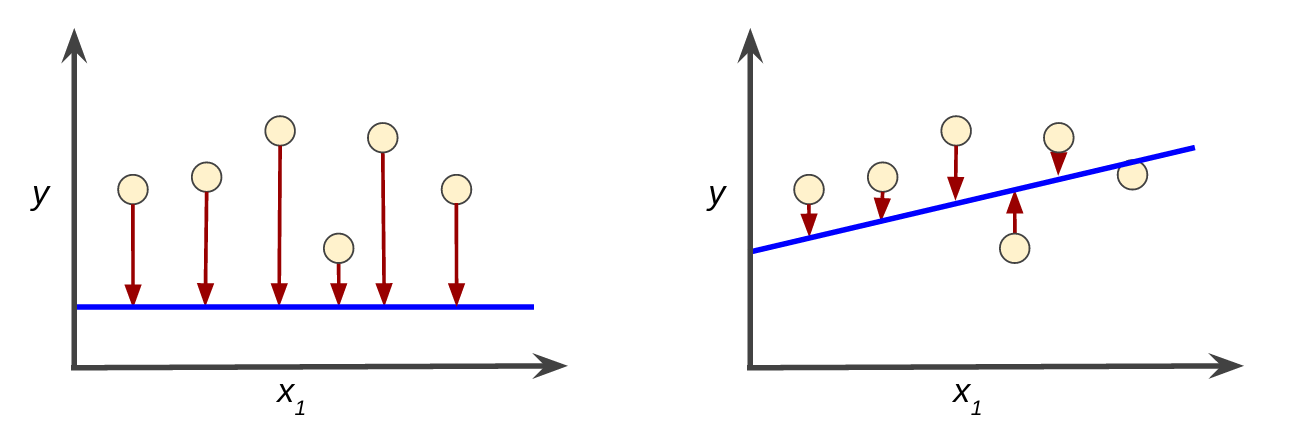
y′ = b + w1x1 + w2x2 + w3x3

Decrescente em ML: treinamento e perda

**Treinar** um modelo significa apenas aprender (determinar) bons valores para todos os pesos e o viés de exemplos rotulados. No aprendizado supervisionado, um algoritmo de machine learning cria um modelo examinando muitos exemplos e tentando encontrar um modelo que minimize a perda. Esse processo é chamado de **minimização do risco empírico**.

Perda é a penalidade para uma previsão ruim. Ou seja, **perda** é um número que indica quão ruim foi a previsão do modelo em um único exemplo. Se a previsão do modelo for perfeita, a perda será zero. Caso contrário, a perda será maior. O treinamento de um modelo visa encontrar um conjunto de ponderações e tendências com uma média de perda *baixa* em todos os exemplos. Por exemplo, a Figura 3 mostra um modelo de alta perda à esquerda e um modelo de baixa perda à direita. Observe o seguinte sobre a figura:

* As setas representam a perda.
* As linhas azuis representam previsões.



**Figura 3. Alta perda no modelo à esquerda e baixa no modelo à direita.**

Observe que as setas no gráfico à esquerda são muito mais compridas do que as que são exibidas no gráfico à direita. Claramente, a linha do gráfico à direita é um modelo preditivo muito melhor do que a linha no gráfico à esquerda.

Você pode estar se perguntando se é possível criar uma função matemática, uma função de perda, que agregaria as perdas individuais de maneira significativa.

### Perda quadrada: uma função de perda conhecida

Os modelos de regressão linear que vamos examinar aqui usam uma função de perda chamada **perda quadrada** (também conhecida como **L2perda**). A perda quadrada para um único exemplo é a seguinte:

= the square of the difference between the label and the prediction

= (observation - prediction(**x**))2

= (y - y')2

**Erro quadrático médio** (**MSE**) é a perda quadrada média por exemplo em todo o conjunto de dados. Para calcular a EQM, soma todas as perdas ao quadrado para exemplos individuais e, em seguida, divida pelo número de exemplos:

MSE = 1/N ∑(x,y)∈D (y − prediction(x))2

onde:

* (x,y) é um exemplo em que
  + x é o conjunto de atributos (por exemplo, chirps/minuto, idade, gênero) que o modelo usa para fazer previsões.
  + y é o rótulo do exemplo (por exemplo, temperatura).
* prediction(x) é uma função dos pesos e vieses em combinação com o conjunto de atributos x.
* D é um conjunto de dados que contém muitos exemplos rotulados, que são (x,y) pares.
* N é o número de exemplos em D.

Embora o MSE seja comumente usado em machine learning, ele não é a única função de perda prática nem a melhor função de perda para todas as circunstâncias.

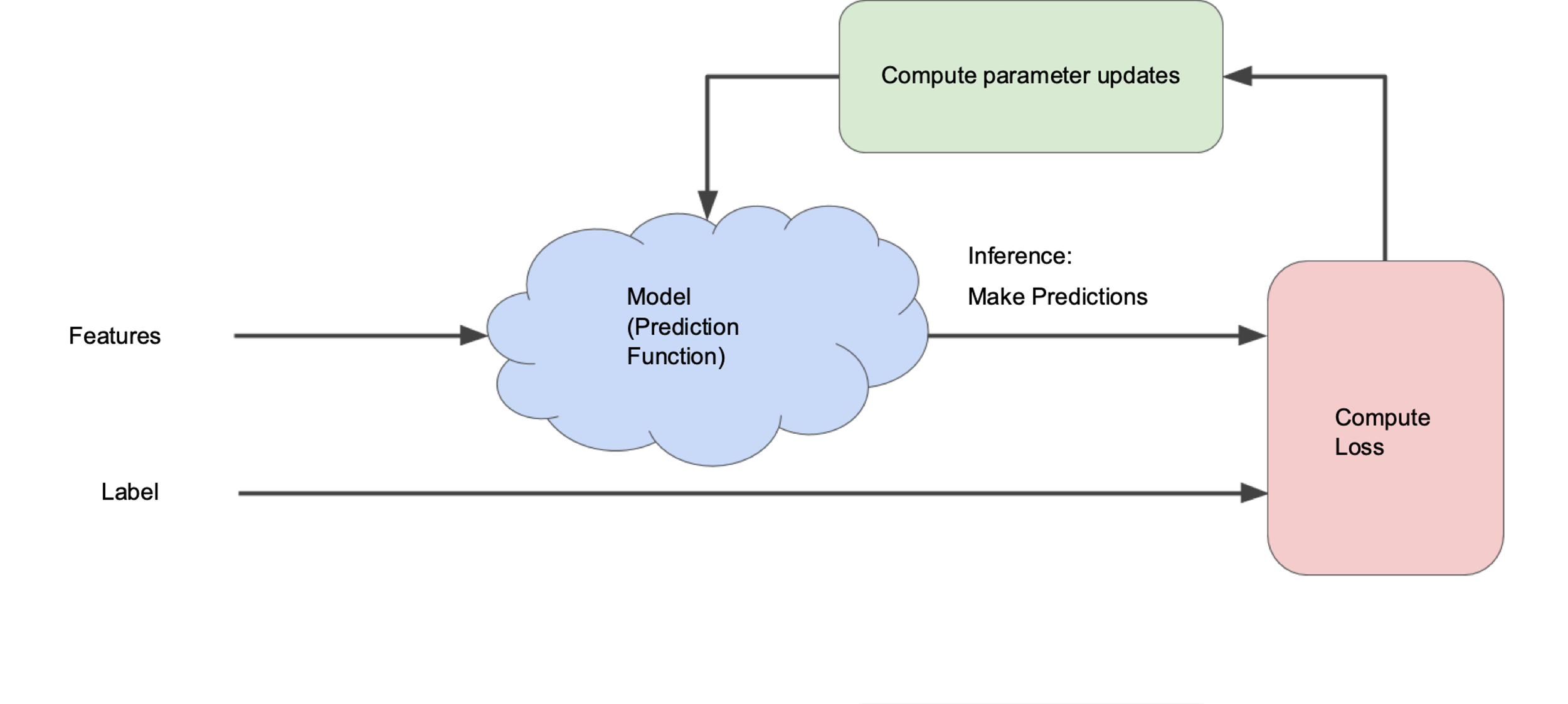
Reduzir Perda

Reduzir perdas: uma abordagem iterativa

O [módulo anterior](https://developers.google.com/machine-learning/crash-course/descending-into-ml?hl=pt-br) introduziu o conceito de perda. Neste módulo, você aprenderá como um modelo de machine learning reduz a perdas de forma iterativa.

O aprendizado iterativo pode lembrar você do jogo infantil ["Hot and Cold"](http://www.howcast.com/videos/258352-how-to-play-hot-and-cold/) para encontrar um objeto oculto, como um dedilhado. Nesse jogo, o "objeto oculto" é o melhor modelo possível. Você começará com uma suposição selvagem ("O valor de w1 é 0.") e esperará que o sistema diga qual foi a perda. Depois, você tentará outra adivinha ("O valor de w1 é 0.5.") e verá qual é a perda. Ah, você está se preparando. Na verdade, se você jogar bem, vai ficar mais quente. O verdadeiro truque para o jogo é tentar encontrar o melhor modelo possível da maneira mais eficiente possível.

A figura a seguir sugere o processo iterativo de tentativa e erro que os algoritmos de machine learning usam para treinar um modelo:



**Figura 1. Uma abordagem iterativa para treinar um modelo.**

Usaremos essa mesma abordagem iterativa em todo o curso "Crashing Machine Learning Crash", detalhando várias complicações, principalmente dentro dessa nuvem nublada rotulada como "quot;Model (função de previsão)". As estratégias iterativas são predominantes no aprendizado de máquina, principalmente porque são altamente escalonáveis em grandes conjuntos de dados.

O "model" usa um ou mais atributos como entrada e retorna uma previsão (y′) como saída. Para simplificar, considere um modelo que usa um recurso e retorna uma previsão:

y ′ = b + w1x1

Quais valores iniciais definir para b e w1? Para problemas de regressão linear, descobrimos que os valores iniciais não são importantes. Podemos escolher valores aleatórios, mas vamos usar apenas os seguintes valores triviais:

* b = 0
* w1 = 0

Suponha que o primeiro valor do recurso seja 10. Conectar esse valor de recurso à função de previsão gera:

y′ = 0 + 0 ⋅ 10 = 0

A parte "Perda de computação" do diagrama é a [função de perda](https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss?hl=pt-br) que o modelo usará. Suponha que usemos a função de perda quadrada. A função de perda recebe dois valores de entrada:

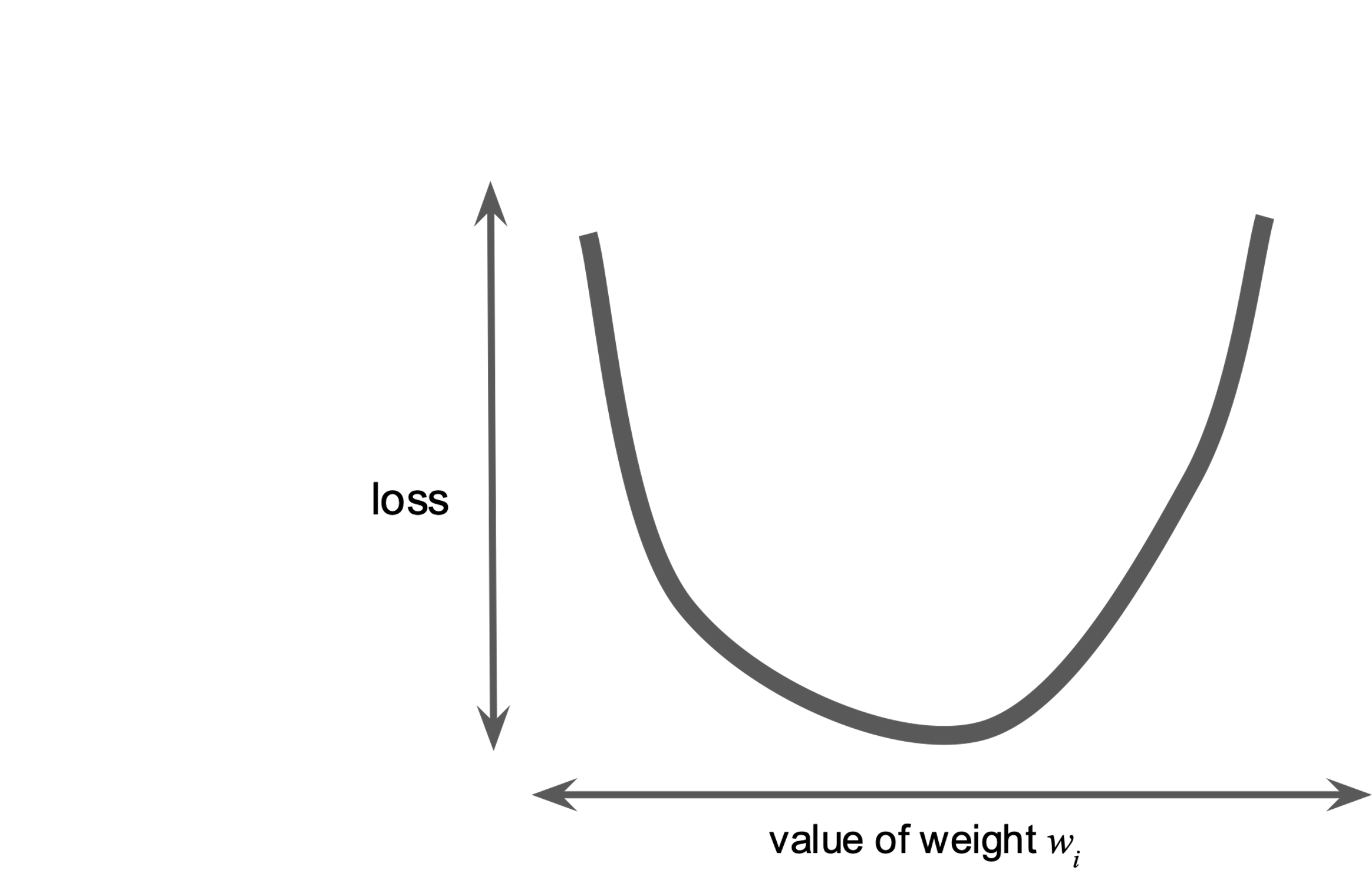
* y′: a previsão do modelo para atributos x
* y: o rótulo correto correspondente aos recursos x.

Por fim, chegamos à parte "Calcular atualizações de parâmetros" do diagrama. É aqui que o sistema de machine learning examina o valor da função de perda e gera novos valores para b e w1. Por enquanto, suponha que essa caixa misteriosa elabore novos valores e, em seguida, o sistema de machine learning reavalia todos esses recursos em comparação a todos esses rótulos, produzindo um novo valor para a função de perda, o que produz novos valores de parâmetros. O aprendizado continua a iterar até que o algoritmo descubra os parâmetros do modelo com a menor perda possível. Em geral, você faz uma iteração até que a perda geral pare de mudar ou pelo menos mude de forma muito lenta. Quando isso acontece, dizemos que o modelo convergiu .

Reduzir perda: gradiente descendente

O diagrama de abordagem iterativa ([Figura 1](https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach?hl=pt-br#ml-block-diagram)) continha uma caixa ondulada em verde chamada "Compute Updates updates." Substituímos essa poeira algorítmica por algo mais significativo.

Suponha que tivéssemos o tempo e os recursos de computação para calcular a perda para todos os valores possíveis de w1. Para o tipo de problemas de regressão que estamos analisando, o gráfico de perda vs. w1 resultante sempre será convexo. Em outras palavras, o gráfico sempre terá uma forma de tigela, assim:

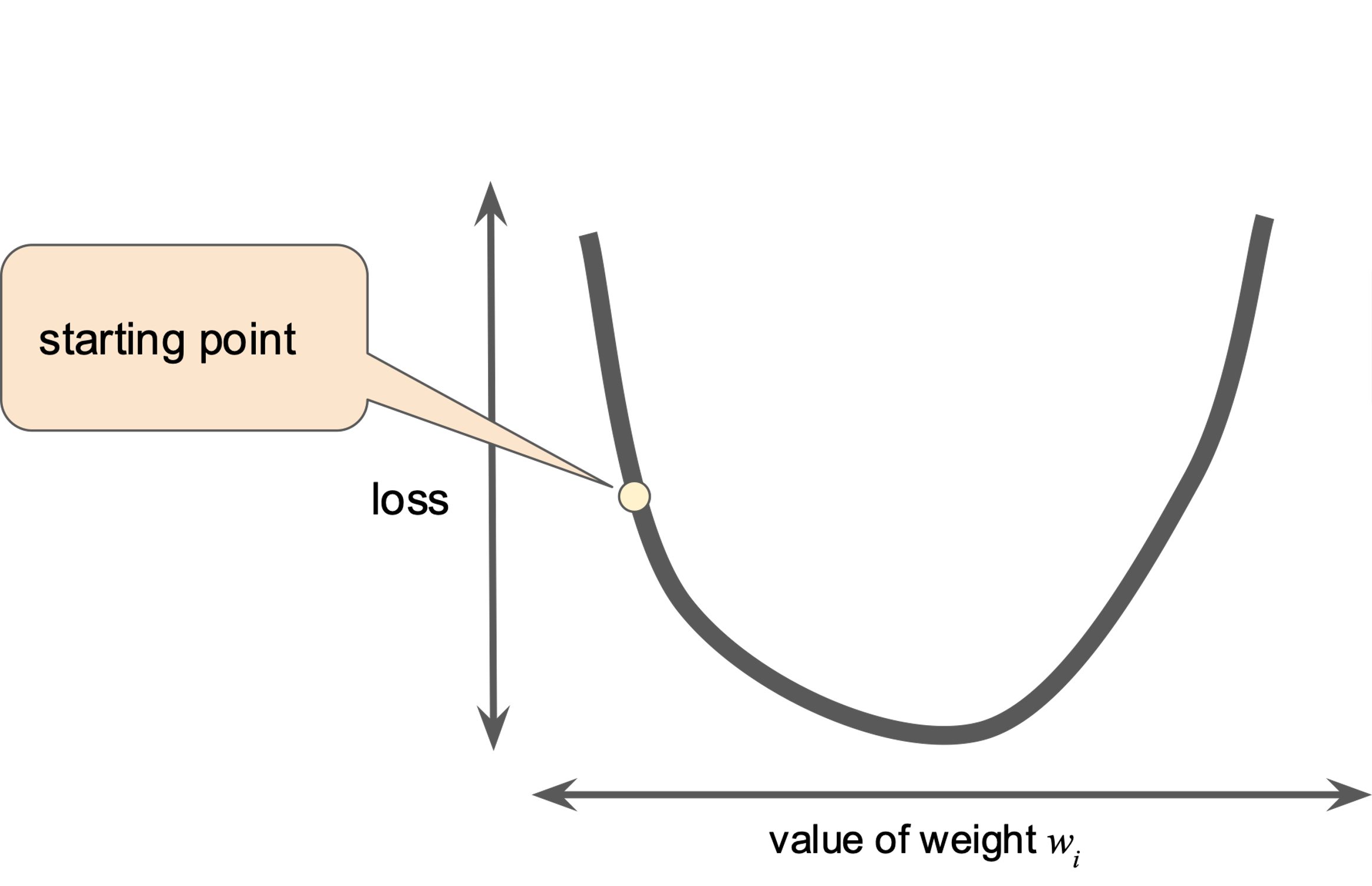


**Figura 2. Problemas de regressão geram perdas de convex em comparação com pesos.**

Os problemas de convex têm apenas um mínimo, ou seja, apenas um lugar em que a inclinação é exatamente 0. Esse mínimo é onde a função de perda converge.

Calcular a função de perda para cada valor concisível de w1sobre todo o conjunto de dados seria uma maneira ineficiente de encontrar o ponto de convergência. Vamos examinar um mecanismo melhor, muito conhecido em machine learning, chamado de **gradiente descendente**.

O primeiro estágio do gradiente descendente é escolher um valor inicial (um ponto de partida) para w1. O ponto de partida é pouco importante. Portanto, muitos algoritmos simplesmente definem w1 como 0 ou escolhem um valor aleatório. A figura a seguir mostra que escolhemos um ponto de partida ligeiramente maior que 0:



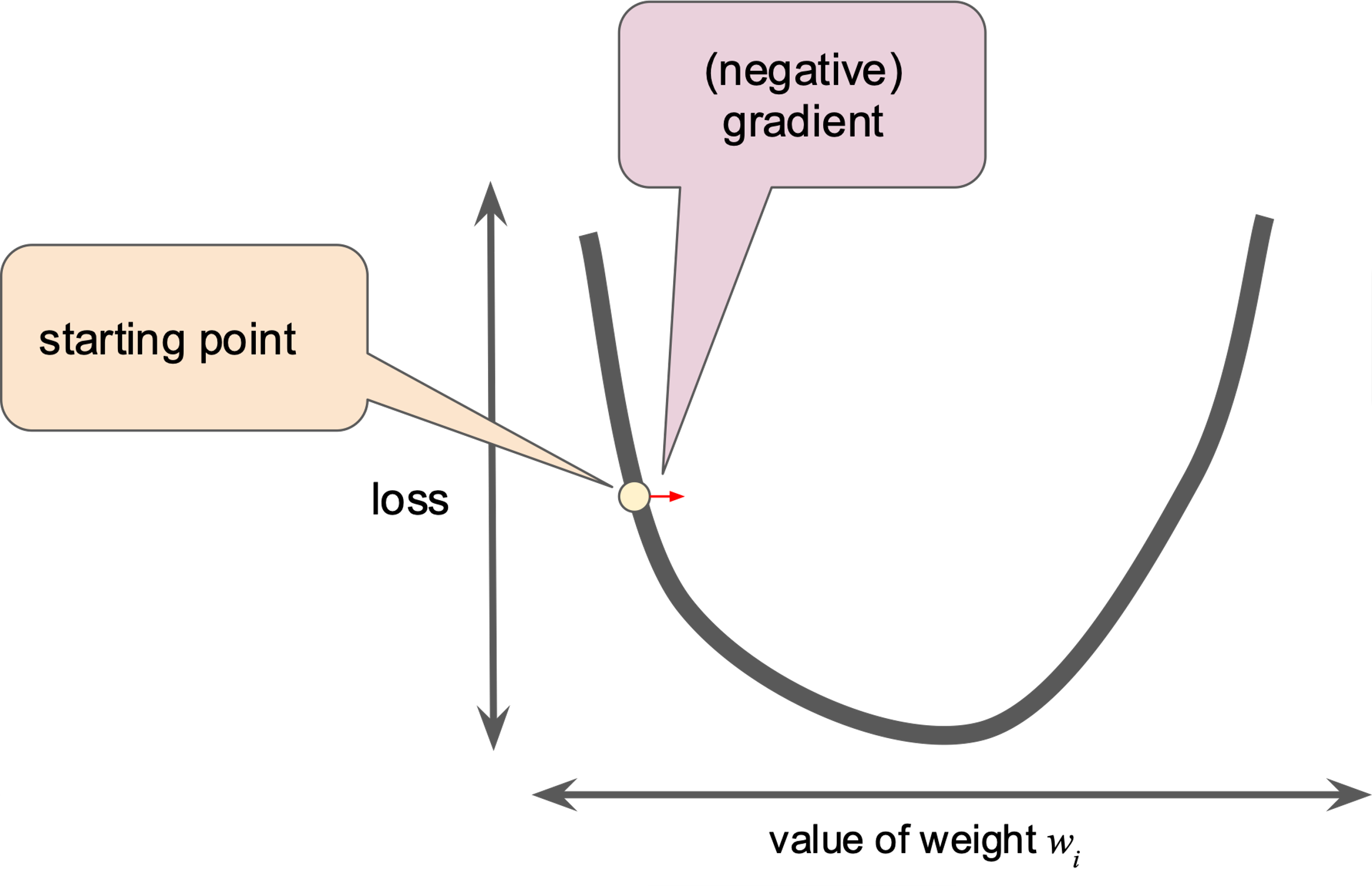
**Figura 3. Um ponto de partida para o gradiente descendente.**

O algoritmo de gradiente descendente então calcula o gradiente da curva de perda no ponto inicial. Na Figura 3, o gradiente da perda é igual à [derivativa](https://wikipedia.org/wiki/Differential_calculus#The_derivative) (inclinação) da curva e informa qual direção é "quot;warmer" ou "colder." Quando há vários pesos, o **gradiente** é um vetor de derivados parciais em relação aos pesos.

Um gradiente é um vetor, portanto, tem as duas características a seguir:

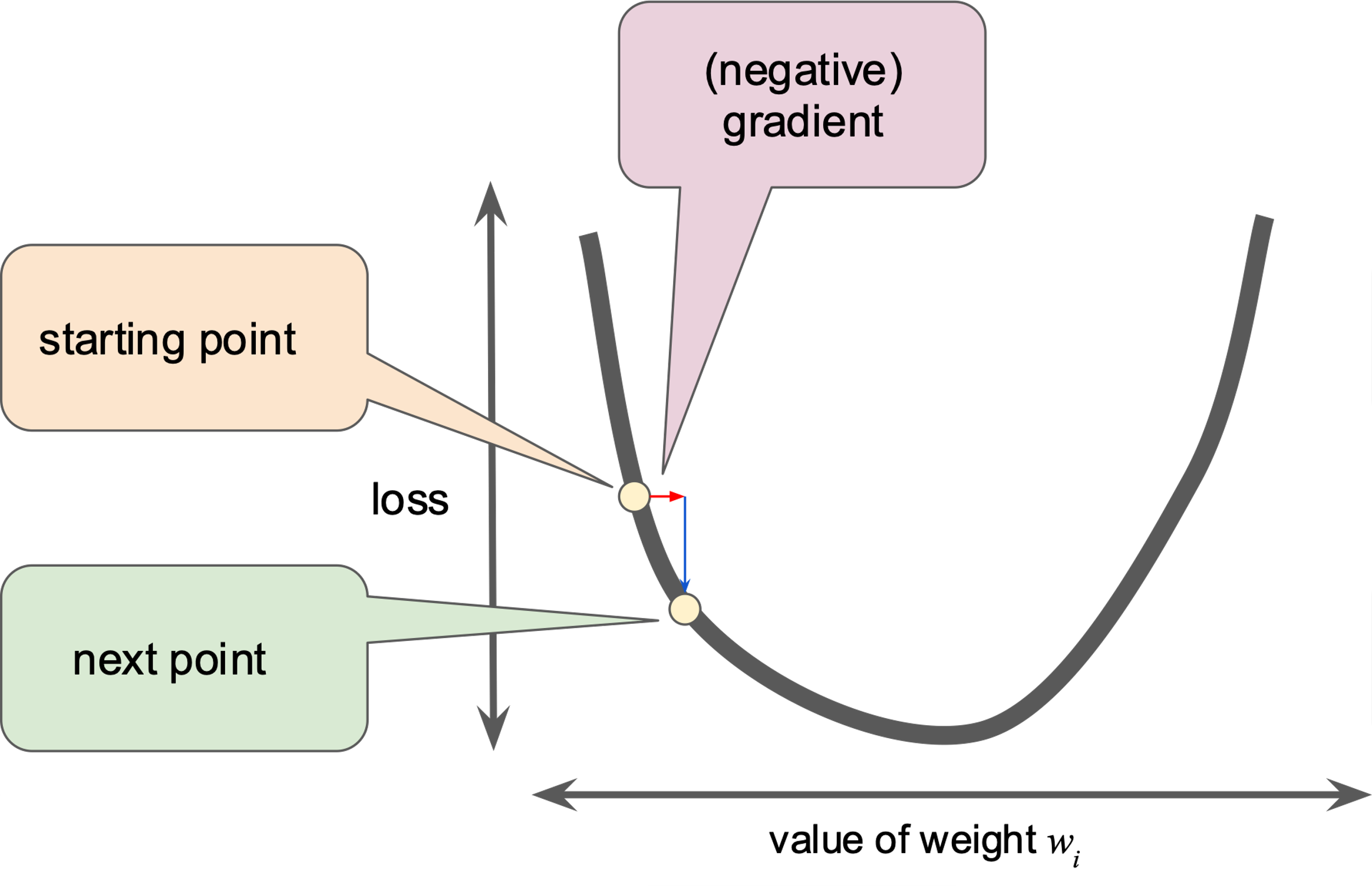
* uma direção
* uma magnitude

O gradiente sempre aponta na direção do maior aumento na função de perda. O algoritmo de gradiente descendente recebe uma etapa na direção do gradiente negativo para reduzir a perda o mais rápido possível.



**Figura 4. O gradiente descendente depende de gradientes negativos.**

Para determinar o próximo ponto ao longo da curva da função de perda, o algoritmo de gradiente descendente adiciona uma fração da magnitude de gradiente ao ponto inicial, como mostrado na figura a seguir:



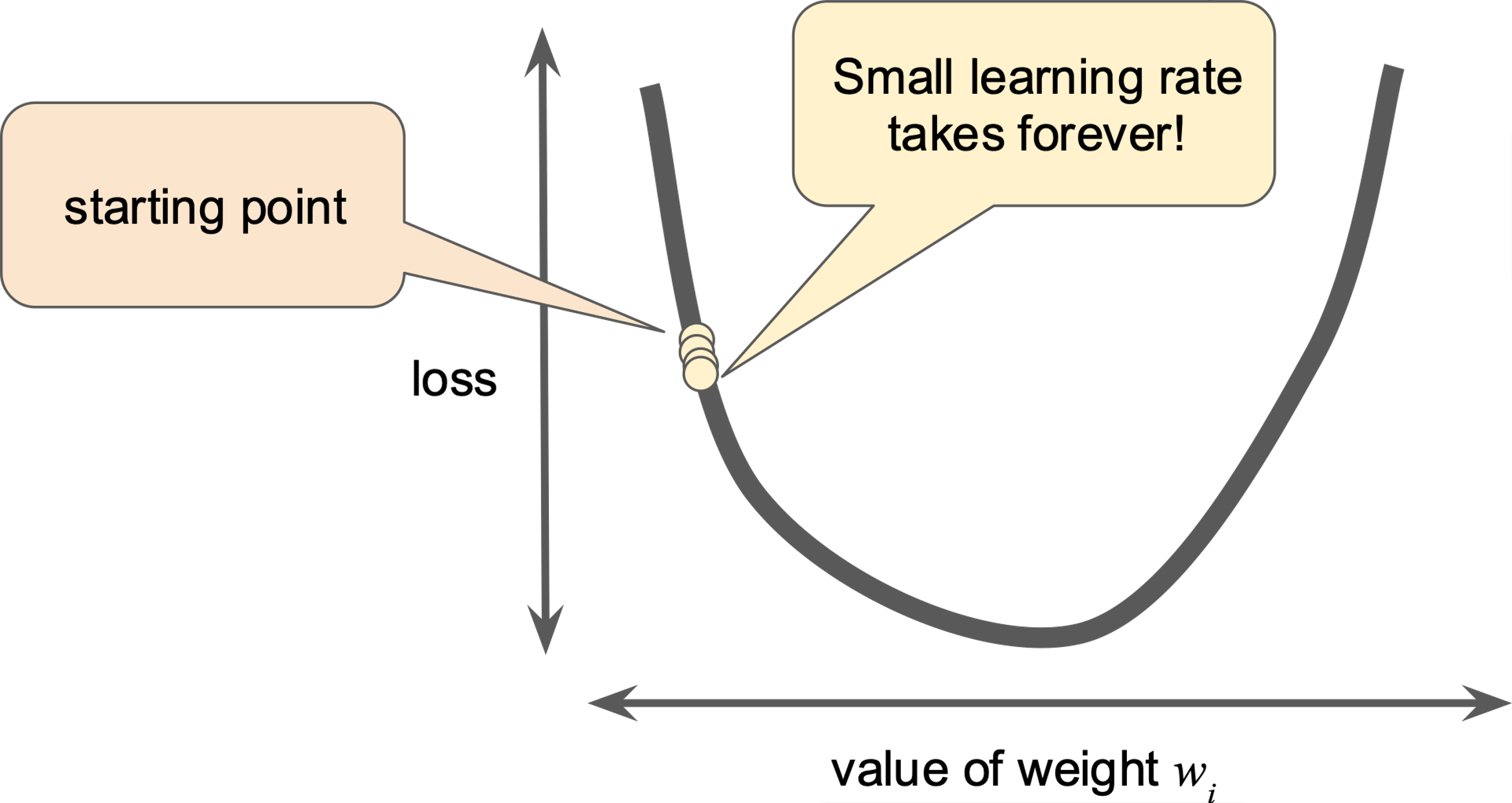
**Figura 5. A etapa de gradiente nos leva ao próximo ponto na curva de perda.**

O gradiente descendente então repete este processo, atingindo cada vez mais perto do mínimo.

# **Reduzir perdas: taxa de aprendizado**

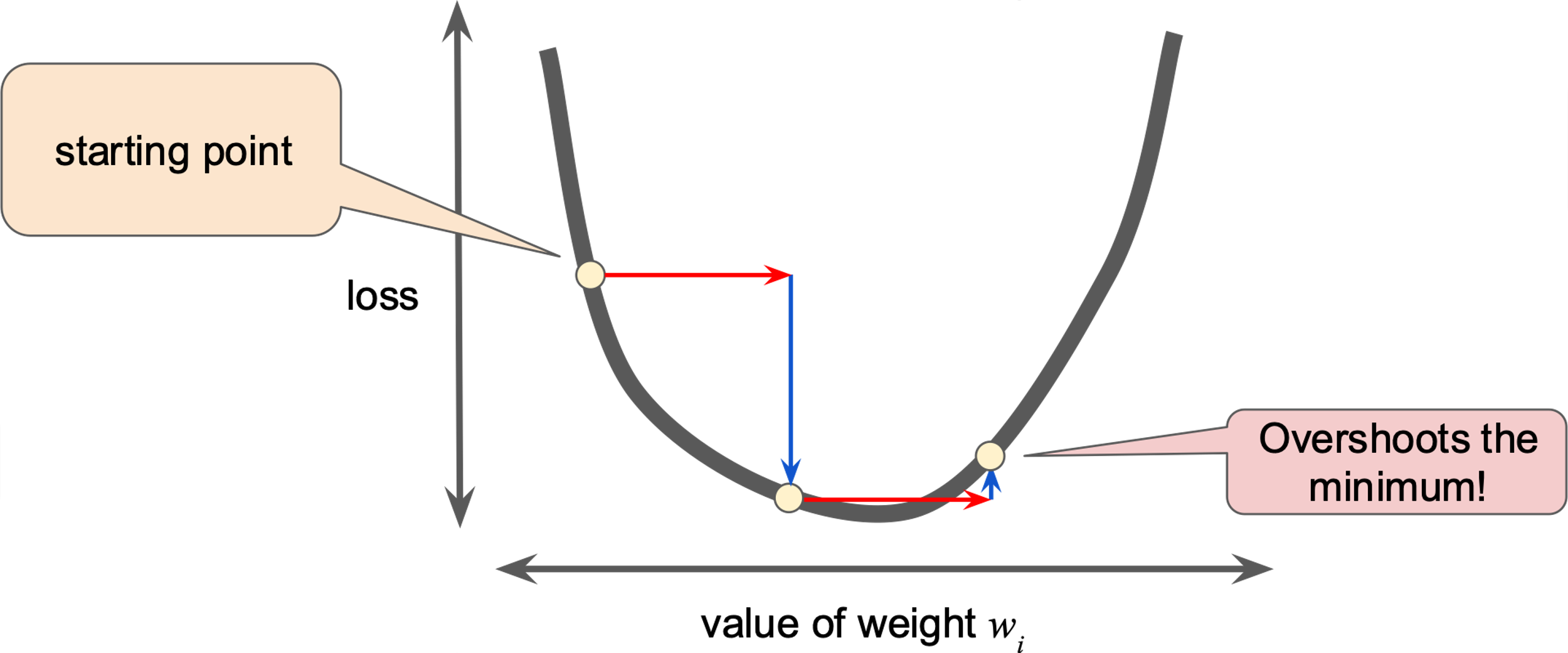
Conforme observado, o vetor de gradiente tem uma direção e uma magnitude. Os algoritmos de gradiente descendente multiplicam o gradiente por um escalar conhecido como **taxa de aprendizado**, também conhecida como **tamanho da etapa**, para determinar o próximo ponto. Por exemplo, se a magnitude do gradiente for 2,5 e a taxa de aprendizado for 0,01, o algoritmo do gradiente descendente vai escolher o próximo ponto a 0,025 do ponto anterior.

Os **hiperparâmetros** são os botões que os programadores ajustam nos algoritmos de aprendizado de máquina. A maioria dos programadores de aprendizado de máquina passa um bom tempo ajustando a taxa de aprendizado. Se você escolher uma taxa de aprendizado muito pequena, o aprendizado levará muito tempo:



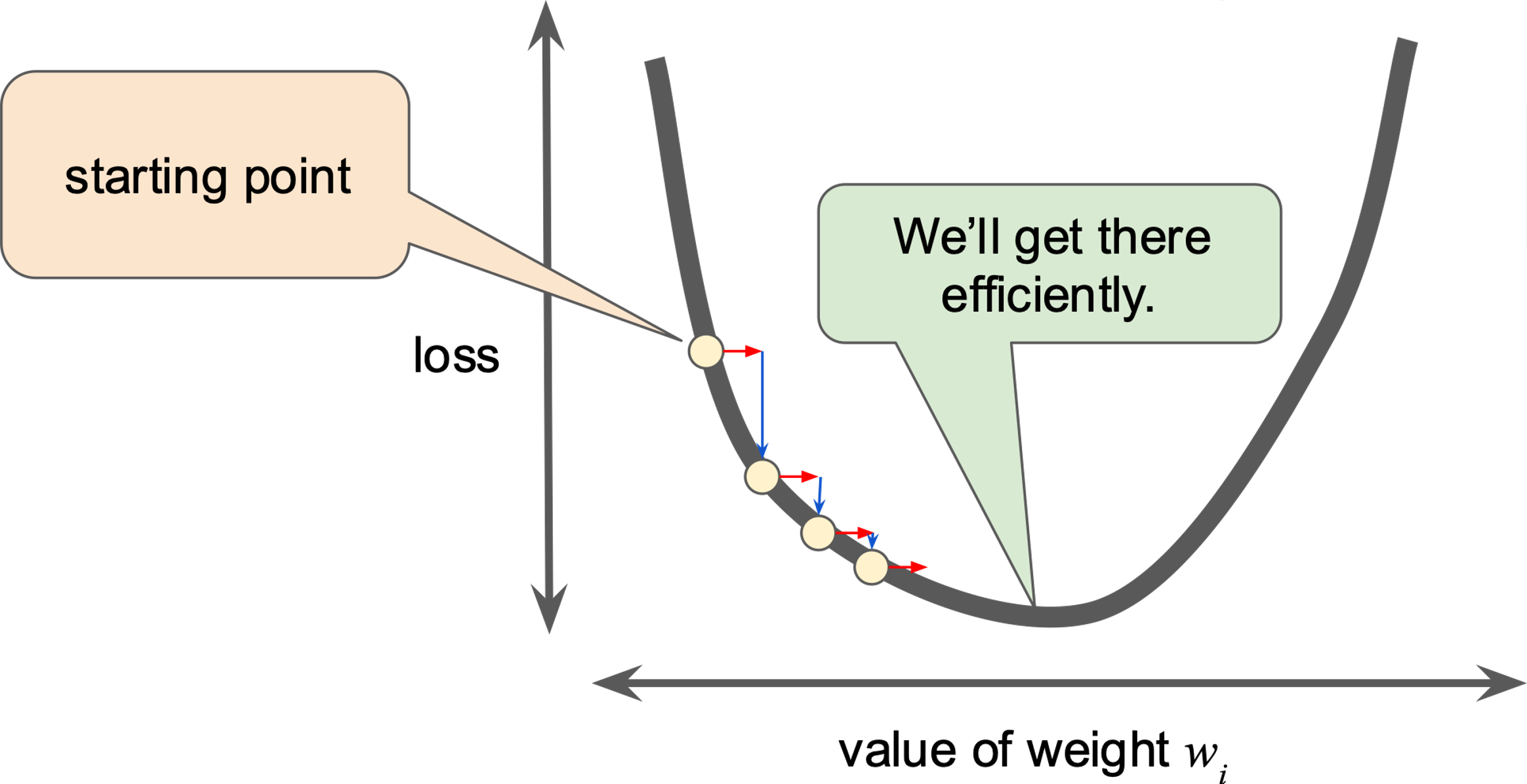
**Figura 6. A taxa de aprendizado é muito pequena.**

Por outro lado, se você especificar uma taxa de aprendizado muito grande, o próximo ponto vai saltar de modo perpétuo ao fundo do poço como um experimento de mecânica quântica muito errado:



**Figura 7. A taxa de aprendizado é muito alta.**

Há uma taxa de aprendizado de [Goldilocks](https://wikipedia.org/wiki/Goldilocks_principle) para cada problema de regressão. O valor de Cachinhos Dourados está relacionado à intensidade da função de perda. Se você souber que o gradiente da função de perda é pequeno, tente usar uma taxa de aprendizado maior com segurança. Isso compensa o pequeno gradiente e resulta em um tamanho de etapa maior.



**Figura 8. A taxa de aprendizado está certa.**

Reduzir perdas: como otimizar a taxa de aprendizado

## Exercício 1

Defina uma taxa de aprendizado de 0,03 no controle deslizante. Continue pressionando o botão ETAPA até que o algoritmo de gradiente descendente atinja o ponto mínimo da curva de perda. Quantos passos foram necessários?

## Exercício 2

Você pode alcançar o mínimo mais rapidamente com uma taxa de aprendizado mais alta? Defina uma taxa de aprendizado de 0, 1 e continue pressionando Step até que o gradiente descendente atinja o mínimo. Quantos passos você completou desta vez?

## Exercício 3

Que tal uma taxa de aprendizado ainda maior. Redefina o gráfico, defina uma taxa de aprendizado de 1 e tente alcançar o mínimo da curva de perda. O que aconteceu dessa vez?

Você pode encontrar a taxa de aprendizado de [Goldilocks](https://wikipedia.org/wiki/Goldilocks_principle) para essa curva, em que o gradiente descendente atinge o ponto mínimo no menor número de etapas? Qual é o menor número de etapas necessárias para alcançar o valor mínimo?

Reduzir perda: gradiente descendente estocástico

No gradiente descendente, um **lote** é o conjunto de exemplos usado para calcular o gradiente em uma única iteração de treinamento. Até agora, presumimos que o lote tenha sido todo o conjunto de dados. Ao trabalhar na escala do Google, os conjuntos de dados geralmente contêm bilhões ou até centenas de bilhões de exemplos. Além disso, os conjuntos de dados do Google geralmente contêm um grande número de recursos. Consequentemente, um lote pode ser enorme. Um lote muito grande pode fazer com que até mesmo uma única iteração demore muito para ser computada.

Um grande conjunto de dados com exemplos de amostra aleatória provavelmente contém dados redundantes. Na verdade, a redundância se torna mais provável à medida que o tamanho do lote aumenta. Algumas redundâncias podem ser úteis para suavizar gradientes barulhentos, mas lotes enormes tendem a não transportar muito mais valor preditivo do que lotes grandes.

E se pudéssemos conseguir o gradiente certo *em média* para muito menos computação? Ao escolher exemplos aleatoriamente do nosso conjunto de dados, poderíamos estimar (embora, com ruído) uma grande média a partir de uma muito menor. O **gradiente descendente estocástico** (**SGD**) leva essa ideia ao extremo. Ele usa apenas um único exemplo (tamanho de lote 1) por iteração. Devido a iterações suficientes, o SGD funciona, mas é muito barulhento. O termo "estocástica" indica que o exemplo que abrange cada lote é escolhido aleatoriamente.

O **gradiente descendente estocástico de minilote** (**SGD de minilote**) é um meio-termo entre a iteração de lote completo e o SGD. Um minilote normalmente fica entre 10 e 1.000 exemplos, escolhidos aleatoriamente. O SGD em lote reduz a quantidade de ruído no SGD, mas ainda é mais eficiente do que em lote completo.

Para simplificar a explicação, nos concentramos no gradiente descendente para um único recurso. O gradiente descendente também funciona em conjuntos de recursos que contêm vários recursos.

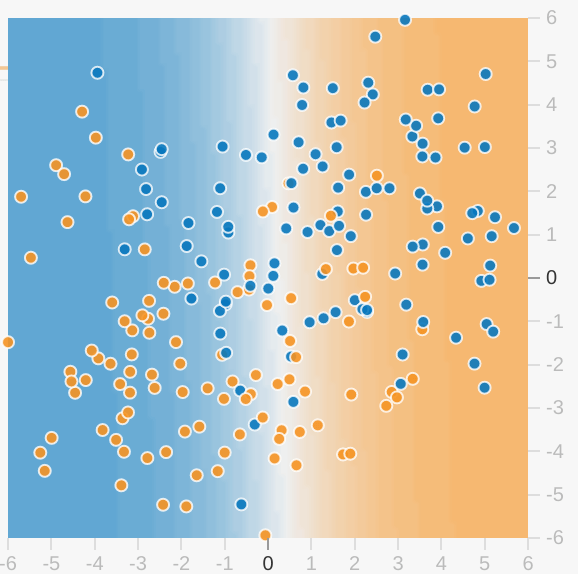
## Taxa de aprendizado e convergência

Este é o primeiro de vários exercícios do Playground. O [Playground](https://playground.tensorflow.org/?hl=pt-br) é um programa desenvolvido especialmente para este curso com o objetivo de ensinar princípios de machine learning. Cada exercício do Playground neste curso inclui uma instância de playground incorporada com predefinições.

Cada exercício do Playground gera um conjunto de dados. O rótulo para este conjunto de dados tem dois valores possíveis. Pense nesses dois valores possíveis como spam ou não spam ou talvez árvores saudáveis em vez de árvores doentes. O objetivo da maioria dos exercícios é ajustar vários hiperparâmetros para criar um modelo que classifica com êxito (separa ou diferencia) um valor de rótulo do outro. Observe que a maioria dos conjuntos de dados contém uma determinada quantidade de ruído que torna impossível classificar com sucesso *todos* os exemplos.

#### Clique no ícone de adição para ver uma explicação da visualização do modelo.

Cada exercício do Playground exibe uma visualização do estado atual do modelo. Por exemplo, veja uma visualização:



Observe o seguinte sobre a visualização do modelo:

* Cada eixo representa um recurso específico. No caso de spam x não spam, os recursos podem ser a contagem de palavras e o número de destinatários do e-mail.**Observação**:os valores apropriados para o eixo dependerão dos dados do recurso. Os valores dos eixos mostrados acima não faziam sentido na contagem de palavras nem no número de destinatários, já que nenhum deles pode ser negativo.
* Cada ponto representa os valores de atributo de um exemplo de dados, como um e-mail.
* A cor do ponto representa a classe a que o exemplo pertence. Por exemplo, os pontos azuis podem representar e-mails que não são spam, enquanto os pontos laranja podem representar e-mails de spam.
* A cor do plano de fundo representa a previsão do modelo de onde os exemplos dessa cor devem ser encontrados. Um plano de fundo azul ao redor de um ponto azul significa que o modelo está prevendo corretamente esse exemplo. Por outro lado, um plano de fundo laranja em torno de um ponto azul significa que o modelo está prevendo incorretamente esse exemplo.
* Os tons de azul e laranja do fundo são ajustados. Por exemplo, o lado esquerdo da visualização fica em azul sólido, mas gradualmente muda para branco no centro da visualização. Pense na força da cor como uma sugestão do modelo para garantir a confiança. Azul sólido significa que o modelo está muito confiante sobre a suposição, e azul-claro significa que o modelo está menos confiante. A visualização do modelo mostrada na figura está fazendo um job ruim de previsão.

Use a visualização para avaliar o progresso do seu modelo. Ótimo! A maioria dos pontos azuis tem um plano de fundo azul, ou “Ah, não! Os pontos azuis têm um plano de fundo laranja.") Além das cores, o Playground também exibe numericamente a perda atual do modelo. (Ah, não! A perda está subindo em vez de desistir.")

A interface deste exercício fornece três botões:

|  |  |  |
| --- | --- | --- |
| **Ícone** | **Nome** | **O que ele faz** |
| Botão de redefinição. | Redefinir | Redefine Iterations como 0. Redefine todos os pesos que o modelo já aprendeu. |
| Botão &quot;Step&quot;. | Etapa | Avançar uma iteração. A cada iteração, o modelo muda, às vezes de forma sutil e significativa, às vezes. |
|  | Gerar novamente | Gera um novo conjunto de dados. Não redefine Iterações. |

Neste primeiro exercício do Playground, você testará a taxa de aprendizado executando duas tarefas.

**Tarefa 1**:observe o menu **Taxa de aprendizado** no canto superior direito do Playground. A taxa de aprendizado fornecida (três) é muito alta. Observe como a alta taxa de aprendizado afeta seu modelo clicando no botão "Step" 10 ou 20 vezes. Após cada iteração inicial, observe como a visualização do modelo muda drasticamente. Você pode até notar alguma instabilidade *depois* que o modelo parece convergir. Observe também as linhas em execução de x1 e x2 na visualização do modelo. Os pesos dessas linhas indicam os pesos desses atributos no modelo. Ou seja, uma linha grossa indica um peso alto.

**Tarefa 2**:faça o seguinte:

1. Pressione o botão **Reset**.
2. Reduza a taxa de aprendizado.
3. Pressione o botão "Step" algumas vezes.

Como a taxa de aprendizado baixa afetou a convergência? Examine o número de etapas necessárias para a convergência do modelo e também se o modelo converge de maneira suave e estável. Faça experimentos com valores ainda menores da taxa de aprendizado. Você acha que uma taxa de aprendizado muito lenta para ser útil? Você verá uma discussão logo abaixo do exercício.

Introdução ao TensorFlow

**Objetivos de aprendizado**

* Aprenda sobre NumPy e pandas para entender o código tf.keras.
* Saiba como usar o Colab.
* Conhecer o código de regressão linear no tf.keras
* Avalie as curvas de perda.
* Ajuste os hiperparâmetros.

O TensorFlow é uma plataforma completa de código aberto para machine learning. O TensorFlow é um sistema avançado de gerenciar todos os aspectos de um sistema de machine learning. No entanto, esta aula se concentra no uso de uma API específica do TensorFlow para desenvolver e treinar modelos de machine learning. Consulte a [documentação do TensorFlow](https://tensorflow.org/?hl=pt-br) para ver detalhes completos sobre o sistema mais amplo do TensorFlow.

As APIs do TensorFlow são organizadas hierarquicamente com as APIs de alto nível criadas nas APIs de baixo nível. Os pesquisadores de machine learning usam as APIs de baixo nível para criar e explorar novos algoritmos de machine learning. Nesta aula, você usará uma API de alto nível chamada tf.keras para definir e treinar modelos de machine learning e fazer previsões. O tf.keras é a variante do TensorFlow da API de código aberto [Keras](https://keras.io/).

A figura a seguir mostra a hierarquia dos toolkits do TensorFlow:

Hierarquia simplificada de kits de ferramentas do TensorFlow. 
   A API tf.keras está na parte superior.

**Figura 1. Hierarquia de ferramentas do TensorFlow.**

Primeiros passos com o TensorFlow: exercícios de programação

À medida que avança no Curso intensivo de machine learning, você colocará conceitos de machine learning em prática codificando os modelos em tf.keras. Você usará o **Colab** como um ambiente de programação. O Colab é a versão do Google para o [Jupyter Notebook](https://jupyter.org/). Assim como o Jupyter Notebook, o Colab fornece um ambiente de programação interativo em Python que combina texto, código, gráficos e saída do programa.

## NumPy e pandas

O uso do tf.keras requer pelo menos um pouco de compreensão das duas bibliotecas Python de código aberto a seguir:

* [NumPy](https://numpy.org/), que simplifica a representação de matrizes e a execução de operações álgebras lineares.
* [pandas](https://pandas.pydata.org/), que fornece uma maneira fácil de representar conjuntos de dados na memória.

Se você não conhece o NumPy ou os pandas, comece fazendo os dois exercícios do Colab a seguir:

1. [Tutorial do NumPy ultrarrápido](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/numpy_ultraquick_tutorial.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=numpy_tf2-colab&hl=pt-br) Colab, que fornece todas as informações de NumPy necessárias para este curso.
2. [Tutorial do pandas ultrarrápido](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/pandas_dataframe_ultraquick_tutorial.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=pandas_tf2-colab&hl=pt-br) Colab, que fornece todas as informações de pandas necessárias para este curso.

## Regressão linear com o tf.keras

Depois de adquirir competência no NumPy e no pandas, faça os dois exercícios do Colab a seguir para explorar a regressão linear e o ajuste de hiperparâmetro em tf.keras:

1. Exercício de [regressão linear com dados sintéticos](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_synthetic_data.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=linear_regression_synthetic_tf2-colab&hl=pt-br) do Colab, que explora a regressão linear com um conjunto de dados de brinquedo.
2. [Regressão linear com um exercício real do Dataset](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_a_real_dataset.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=linear_regression_real_tf2-colab&hl=pt-br) no Colab, que orienta você pelos tipos de análise que precisa fazer em um conjunto de dados real.

Os exercícios de programação são executados diretamente no navegador (sem necessidade de configuração) usando a plataforma [Colaboratory](https://colab.research.google.com/?hl=pt-br). O Colaboratory é compatível com a maioria dos principais navegadores e é testado principalmente nas versões para computador do Chrome e do Firefox. Se preferir fazer o download e executar os exercícios off-line, consulte [estas instruções](https://developers.google.com/machine-learning/crash-course/running-exercises-locally?hl=pt-br) para configurar um ambiente local.

Exercicio 1- R:

# **Simple Linear Regression with Synthetic Data**

In this first Colab, you'll explore linear regression with a simple database.

Learning objectives:

After doing this exercise, you'll know how to do the following:

Run Colabs.

Tune the following [hyperparameters](https://developers.google.com/machine-learning/glossary/#hyperparameter):

[learning rate](https://developers.google.com/machine-learning/glossary/#learning_rate)

number of [epochs](https://developers.google.com/machine-learning/glossary/#epoch)

[batch size](https://developers.google.com/machine-learning/glossary/#batch_size)

Interpret different kinds of [loss curves](https://developers.google.com/machine-learning/glossary/#loss_curve).

Import relevant modules

The following cell imports the packages that the program requires:

import pandas as pd

import tensorflow as tf

from matplotlib import pyplot as plt

Define functions that build and train a model

The following code defines two functions:

build\_model(my\_learning\_rate), which builds an empty model.

train\_model(model, feature, label, epochs), which trains the model from the examples (feature and label) you pass.

Since you don't need to understand model building code right now, we've hidden this code cell. You may optionally double-click the headline to explore this code.

Define the functions that build and train a model

#@title Define the functions that build and train a model

def build\_model(my\_learning\_rate):

"""Create and compile a simple linear regression model."""

# Most simple tf.keras models are sequential.

# A sequential model contains one or more layers.

model = tf.keras.models.Sequential()

# Describe the topography of the model.

# The topography of a simple linear regression model

# is a single node in a single layer.

model.add(tf.keras.layers.Dense(units=1,

input\_shape=(1,)))

# Compile the model topography into code that

# TensorFlow can efficiently execute. Configure

# training to minimize the model's mean squared error.

model.compile(optimizer=tf.keras.optimizers.experimental.RMSprop(learning\_rate=my\_learning\_rate),

loss="mean\_squared\_error",

metrics=[tf.keras.metrics.RootMeanSquaredError()])

return model

def train\_model(model, feature, label, epochs, batch\_size):

"""Train the model by feeding it data."""

# Feed the feature values and the label values to the

# model. The model will train for the specified number

# of epochs, gradually learning how the feature values

# relate to the label values.

history = model.fit(x=feature,

y=label,

batch\_size=batch\_size,

epochs=epochs)

# Gather the trained model's weight and bias.

trained\_weight = model.get\_weights()[0]

trained\_bias = model.get\_weights()[1]

# The list of epochs is stored separately from the

# rest of history.

epochs = history.epoch

# Gather the history (a snapshot) of each epoch.

hist = pd.DataFrame(history.history)

# Specifically gather the model's root mean

# squared error at each epoch.

rmse = hist["root\_mean\_squared\_error"]

return trained\_weight, trained\_bias, epochs, rmse

print("Defined build\_model and train\_model")

Define plotting functions

We're using a popular Python library called [Matplotlib](https://developers.google.com/machine-learning/glossary/#matplotlib) to create the following two plots:

a plot of the feature values vs. the label values, and a line showing the output of the trained model.

a [loss curve](https://developers.google.com/machine-learning/glossary/#loss_curve).

We hid the following code cell because learning Matplotlib is not relevant to the learning objectives. Regardless, you must still run all hidden code cells.

Define the plotting functions

#@title Define the plotting functions

def plot\_the\_model(trained\_weight, trained\_bias, feature, label):

"""Plot the trained model against the training feature and label."""

# Label the axes.

plt.xlabel("feature")

plt.ylabel("label")

# Plot the feature values vs. label values.

plt.scatter(feature, label)

# Create a red line representing the model. The red line starts

# at coordinates (x0, y0) and ends at coordinates (x1, y1).

x0 = 0

y0 = trained\_bias

x1 = feature[-1]

y1 = trained\_bias + (trained\_weight \* x1)

plt.plot([x0, x1], [y0, y1], c='r')

# Render the scatter plot and the red line.

plt.show()

def plot\_the\_loss\_curve(epochs, rmse):

"""Plot the loss curve, which shows loss vs. epoch."""

plt.figure()

plt.xlabel("Epoch")

plt.ylabel("Root Mean Squared Error")

plt.plot(epochs, rmse, label="Loss")

plt.legend()

plt.ylim([rmse.min()\*0.97, rmse.max()])

plt.show()

print("Defined the plot\_the\_model and plot\_the\_loss\_curve functions.")

**Define the dataset**

The dataset consists of 12 [examples](https://developers.google.com/machine-learning/glossary/#example). Each example consists of one [feature](https://developers.google.com/machine-learning/glossary/#feature) and one [label](https://developers.google.com/machine-learning/glossary/#label).

my\_feature = ([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0])

my\_label = ([5.0, 8.8, 9.6, 14.2, 18.8, 19.5, 21.4, 26.8, 28.9, 32.0, 33.8, 38.2])

Specify the hyperparameters

The hyperparameters in this Colab are as follows:

[learning rate](https://developers.google.com/machine-learning/glossary/#learning_rate)

[epochs](https://developers.google.com/machine-learning/glossary/#epoch)

[batch\_size](https://developers.google.com/machine-learning/glossary/#batch_size)

The following code cell initializes these hyperparameters and then invokes the functions that build and train the model.

Task 1: Examine the graphs

Examine the top graph. The blue dots identify the actual data; the red line identifies the output of the trained model. Ideally, the red line should align nicely with the blue dots. Does it? Probably not.

A certain amount of randomness plays into training a model, so you'll get somewhat different results every time you train. That said, unless you are an extremely lucky person, the red line probably doesn't align nicely with the blue dots.

Examine the bottom graph, which shows the loss curve. Notice that the loss curve decreases but doesn't flatten out, which is a sign that the model hasn't trained sufficiently.

Task 2: Increase the number of epochs

Training loss should steadily decrease, steeply at first, and then more slowly. Eventually, training loss should stay steady (zero slope or nearly zero slope), which indicates that training has [converged](https://colab.research.google.com/corgiredirector?site=http%3A%2F%2Fdevelopers.google.com%2Fmachine-learning%2Fglossary%2F%23convergence" \t "_blank).

In Task 1, the training loss did not converge. One possible solution is to train for more epochs. Your task is to increase the number of epochs sufficiently to get the model to converge. However, it is inefficient to train past convergence, so don't just set the number of epochs to an arbitrarily high value.

Examine the loss curve. Does the model converge?

#@title Double-click to view a possible solution

learning\_rate=0.01

epochs=450

my\_batch\_size=12

my\_model = build\_model(learning\_rate)

trained\_weight, trained\_bias, epochs, rmse = train\_model(my\_model, my\_feature,

my\_label, epochs,

my\_batch\_size)

plot\_the\_model(trained\_weight, trained\_bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

# The loss curve suggests that the model does converge.

**Task 3: Increase the learning rate**

In Task 2, you increased the number of epochs to get the model to converge. Sometimes, you can get the model to converge more quickly by increasing the learning rate. However, setting the learning rate too high often makes it impossible for a model to converge. In Task 3, we've intentionally set the learning rate too high. Run the following code cell and see what happens.

# Increase the learning rate and decrease the number of epochs.

learning\_rate=100

epochs=500

my\_model = build\_model(learning\_rate)

trained\_weight, trained\_bias, epochs, rmse = train\_model(my\_model, my\_feature,

my\_label, epochs,

my\_batch\_size)

plot\_the\_model(trained\_weight, trained\_bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

The resulting model is terrible; the red line doesn't align with the blue dots. Furthermore, the loss curve oscillates like a [roller coaster](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.wikipedia.org%2Fwiki%2FRoller_coaster). An oscillating loss curve strongly suggests that the learning rate is too high.

Task 4: Find the ideal combination of epochs and learning rate

Assign values to the following two hyperparameters to make training converge as efficiently as possible:

learning\_rate

epochs

#@title Double-click to view a possible solution

learning\_rate=0.14

epochs=70

my\_model = build\_model(learning\_rate)

trained\_weight, trained\_bias, epochs, rmse = train\_model(my\_model, my\_feature,

my\_label, epochs,

my\_batch\_size)

plot\_the\_model(trained\_weight, trained\_bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

**Task 5: Adjust the batch size**

The system recalculates the model's loss value and adjusts the model's weights and bias after each **iteration**. Each iteration is the span in which the system processes one batch. For example, if the **batch size** is 6, then the system recalculates the model's loss value and adjusts the model's weights and bias after processing every 6 examples.

One **epoch** spans sufficient iterations to process every example in the dataset. For example, if the batch size is 12, then each epoch lasts one iteration. However, if the batch size is 6, then each epoch consumes two iterations.

It is tempting to simply set the batch size to the number of examples in the dataset (12, in this case). However, the model might actually train faster on smaller batches. Conversely, very small batches might not contain enough information to help the model converge.

Experiment with batch\_size in the following code cell. What's the smallest integer you can set for batch\_size and still have the model converge in a hundred epochs?

#@title Double-click to view a possible solution

learning\_rate=0.05

epochs=100

my\_batch\_size=1 # Wow, a batch size of 1 works!

my\_model = build\_model(learning\_rate)

trained\_weight, trained\_bias, epochs, rmse = train\_model(my\_model, my\_feature,

my\_label, epochs,

my\_batch\_size)

plot\_the\_model(trained\_weight, trained\_bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

**Summary of hyperparameter tuning**

Most machine learning problems require a lot of hyperparameter tuning. Unfortunately, we can't provide concrete tuning rules for every model. Lowering the learning rate can help one model converge efficiently but make another model converge much too slowly. You must experiment to find the best set of hyperparameters for your dataset. That said, here are a few rules of thumb:

Training loss should steadily decrease, steeply at first, and then more slowly until the slope of the curve reaches or approaches zero.

If the training loss does not converge, train for more epochs.

If the training loss decreases too slowly, increase the learning rate. Note that setting the learning rate too high may also prevent training loss from converging.

If the training loss varies wildly (that is, the training loss jumps around), decrease the learning rate.

Lowering the learning rate while increasing the number of epochs or the batch size is often a good combination.

Setting the batch size to a very small batch number can also cause instability. First, try large batch size values. Then, decrease the batch size until you see degradation.

For real-world datasets consisting of a very large number of examples, the entire dataset might not fit into memory. In such cases, you'll need to reduce the batch size to enable a batch to fit into memory.

Remember: the ideal combination of hyperparameters is data dependent, so you must always experiment and verify.

[Regressão linear com um exercício real do Dataset](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_a_real_dataset.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=linear_regression_real_tf2-colab&hl=pt-br)

Linear Regression with a Real Dataset

This Colab uses a real dataset to predict the prices of houses in California.

Learning Objectives:

After doing this Colab, you'll know how to do the following:

Read a .csv file into a [pandas](https://developers.google.com/machine-learning/glossary/" \l "pandas" \t "_blank) DataFrame.

Examine a [dataset](https://developers.google.com/machine-learning/glossary/" \l "data_set" \t "_blank).

Experiment with different [features](https://developers.google.com/machine-learning/glossary/" \l "feature" \t "_blank) in building a model.

Tune the model's [hyperparameters](https://developers.google.com/machine-learning/glossary/" \l "hyperparameter" \t "_blank).

CodeText

The Dataset

The [dataset for this exercise](https://developers.google.com/machine-learning/crash-course/california-housing-data-description" \t "_blank) is based on 1990 census data from California. The dataset is old but still provides a great opportunity to learn about machine learning programming.

Import relevant modules

#@title Import relevant modules

import pandas as pd

import tensorflow as tf

from matplotlib import pyplot as plt

# The following lines adjust the granularity of reporting.

pd.options.display.max\_rows = 10

pd.options.display.float\_format = "{:.1f}".format

The dataset

Datasets are often stored on disk or at a URL in [.csv format](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwikipedia.org%2Fwiki%2FComma-separated_values).

A well-formed .csv file contains column names in the first row, followed by many rows of data. A comma divides each value in each row. For example, here are the first five rows of the .csv file holding the California Housing Dataset:

"longitude","latitude","housing\_median\_age","total\_rooms","total\_bedrooms","population","households","median\_income","median\_house\_value"  
-114.310000,34.190000,15.000000,5612.000000,1283.000000,1015.000000,472.000000,1.493600,66900.000000  
-114.470000,34.400000,19.000000,7650.000000,1901.000000,1129.000000,463.000000,1.820000,80100.000000  
-114.560000,33.690000,17.000000,720.000000,174.000000,333.000000,117.000000,1.650900,85700.000000  
-114.570000,33.640000,14.000000,1501.000000,337.000000,515.000000,226.000000,3.191700,73400.000000

Load the .csv file into a pandas DataFrame

This Colab, like many machine learning programs, gathers the .csv file and stores the data in memory as a pandas Dataframe. Pandas is an open source Python library. The primary datatype in pandas is a DataFrame. You can imagine a pandas DataFrame as a spreadsheet in which each row is identified by a number and each column by a name. Pandas is itself built on another open source Python library called NumPy. If you aren't familiar with these technologies, please view these two quick tutorials:

[NumPy](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/numpy_ultraquick_tutorial.ipynb?utm_source=linearregressionreal-colab&utm_medium=colab&utm_campaign=colab-external&utm_content=numpy_tf2-colab&hl=en)

[Pandas DataFrames](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/pandas_dataframe_ultraquick_tutorial.ipynb?utm_source=linearregressionreal-colab&utm_medium=colab&utm_campaign=colab-external&utm_content=pandas_tf2-colab&hl=en)

The following code cell imports the .csv file into a pandas DataFrame and scales the values in the label (median\_house\_value):

# Import the dataset.

training\_df = pd.read\_csv(filepath\_or\_buffer="https://download.mlcc.google.com/mledu-datasets/california\_housing\_train.csv")

# Scale the label.

training\_df["median\_house\_value"] /= 1000.0

# Print the first rows of the pandas DataFrame.

training\_df.head()

Scaling median\_house\_value puts the value of each house in units of thousands. Scaling will keep loss values and learning rates in a friendlier range.

Although scaling a label is usually not essential, scaling features in a multi-feature model usually is essential.

Examine the dataset

A large part of most machine learning projects is getting to know your data. The pandas API provides a describe function that outputs the following statistics about every column in the DataFrame:

count, which is the number of rows in that column. Ideally, count contains the same value for every column.

mean and std, which contain the mean and standard deviation of the values in each column.

min and max, which contain the lowest and highest values in each column.

25%, 50%, 75%, which contain various [quantiles](https://developers.google.com/machine-learning/glossary/#quantile).

# Get statistics on the dataset.

training\_df.describe()

Define functions that build and train a model

The following code defines two functions:

build\_model(my\_learning\_rate), which builds a randomly-initialized model.

train\_model(model, feature, label, epochs), which trains the model from the examples (feature and label) you pass.

Since you don't need to understand model building code right now, we've hidden this code cell. You may optionally double-click the following headline to see the code that builds and trains a model.

#@title Define the functions that build and train a model

def build\_model(my\_learning\_rate):

"""Create and compile a simple linear regression model."""

# Most simple tf.keras models are sequential.

model = tf.keras.models.Sequential()

# Describe the topography of the model.

# The topography of a simple linear regression model

# is a single node in a single layer.

model.add(tf.keras.layers.Dense(units=1,

input\_shape=(1,)))

# Compile the model topography into code that TensorFlow can efficiently

# execute. Configure training to minimize the model's mean squared error.

model.compile(optimizer=tf.keras.optimizers.experimental.RMSprop(learning\_rate=my\_learning\_rate),

loss="mean\_squared\_error",

metrics=[tf.keras.metrics.RootMeanSquaredError()])

return model

def train\_model(model, df, feature, label, epochs, batch\_size):

"""Train the model by feeding it data."""

# Feed the model the feature and the label.

# The model will train for the specified number of epochs.

history = model.fit(x=df[feature],

y=df[label],

batch\_size=batch\_size,

epochs=epochs)

# Gather the trained model's weight and bias.

trained\_weight = model.get\_weights()[0]

trained\_bias = model.get\_weights()[1]

# The list of epochs is stored separately from the rest of history.

epochs = history.epoch

# Isolate the error for each epoch.

hist = pd.DataFrame(history.history)

# To track the progression of training, we're going to take a snapshot

# of the model's root mean squared error at each epoch.

rmse = hist["root\_mean\_squared\_error"]

return trained\_weight, trained\_bias, epochs, rmse

print("Defined the build\_model and train\_model functions.")

Define plotting functions

The following [matplotlib](https://developers.google.com/machine-learning/glossary/#matplotlib) functions create the following plots:

a scatter plot of the feature vs. the label, and a line showing the output of the trained model

a loss curve

You may optionally double-click the headline to see the matplotlib code, but note that writing matplotlib code is not an important part of learning ML programming.

#@title Define the plotting functions

def plot\_the\_model(trained\_weight, trained\_bias, feature, label):

"""Plot the trained model against 200 random training examples."""

# Label the axes.

plt.xlabel(feature)

plt.ylabel(label)

# Create a scatter plot from 200 random points of the dataset.

random\_examples = training\_df.sample(n=200)

plt.scatter(random\_examples[feature], random\_examples[label])

# Create a red line representing the model. The red line starts

# at coordinates (x0, y0) and ends at coordinates (x1, y1).

x0 = 0

y0 = trained\_bias

x1 = random\_examples[feature].max()

y1 = trained\_bias + (trained\_weight \* x1)

plt.plot([x0, x1], [y0, y1], c='r')

# Render the scatter plot and the red line.

plt.show()

def plot\_the\_loss\_curve(epochs, rmse):

"""Plot a curve of loss vs. epoch."""

plt.figure()

plt.xlabel("Epoch")

plt.ylabel("Root Mean Squared Error")

plt.plot(epochs, rmse, label="Loss")

plt.legend()

plt.ylim([rmse.min()\*0.97, rmse.max()])

plt.show()

print("Defined the plot\_the\_model and plot\_the\_loss\_curve functions.")

Call the model functions

An important part of machine learning is determining which [features](https://developers.google.com/machine-learning/glossary/#feature) correlate with the [label](https://developers.google.com/machine-learning/glossary/#label). For example, real-life home-value prediction models typically rely on hundreds of features and synthetic features. However, this model relies on only one feature. For now, you'll arbitrarily use total\_rooms as that feature.

CodeText

# The following variables are the hyperparameters.

learning\_rate = 0.01

epochs = 30

batch\_size = 30

# Specify the feature and the label.

my\_feature = "total\_rooms" # the total number of rooms on a specific city block.

my\_label="median\_house\_value" # the median value of a house on a specific city block.

# That is, you're going to create a model that predicts house value based

# solely on total\_rooms.

# Discard any pre-existing version of the model.

my\_model = None

# Invoke the functions.

my\_model = build\_model(learning\_rate)

weight, bias, epochs, rmse = train\_model(my\_model, training\_df,

my\_feature, my\_label,

epochs, batch\_size)

print("\nThe learned weight for your model is %.4f" % weight)

print("The learned bias for your model is %.4f\n" % bias )

plot\_the\_model(weight, bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

A certain amount of randomness plays into training a model. Consequently, you'll get different results each time you train the model. That said, given the dataset and the hyperparameters, the trained model will generally do a poor job describing the feature's relation to the label.

Use the model to make predictions

You can use the trained model to make predictions. In practice, [you should make predictions on examples that are not used in training](https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data). However, for this exercise, you'll just work with a subset of the same training dataset. A later Colab exercise will explore ways to make predictions on examples not used in training.

First, run the following code to define the house prediction function:

def predict\_house\_values(n, feature, label):

"""Predict house values based on a feature."""

batch = training\_df[feature][10000:10000 + n]

predicted\_values = my\_model.predict\_on\_batch(x=batch)

print("feature label predicted")

print(" value value value")

print(" in thousand$ in thousand$")

print("--------------------------------------")

for i in range(n):

print ("%5.0f %6.0f %15.0f" % (training\_df[feature][10000 + i],

training\_df[label][10000 + i],

predicted\_values[i][0] ))

Now, invoke the house prediction function on 10 examples:

[ ]

predict\_house\_values(10, my\_feature, my\_label)

Task 2: Judge the predictive power of the model

Look at the preceding table. How close is the predicted value to the label value? In other words, does your model accurately predict house values?

#@title Double-click to view the answer.

# Most of the predicted values differ significantly

# from the label value, so the trained model probably

# doesn't have much predictive power. However, the

# first 10 examples might not be representative of

# the rest of the examples.

Task 3: Try a different feature

The total\_rooms feature had only a little predictive power. Would a different feature have greater predictive power? Try using population as the feature instead of total\_rooms.

Note: When you change features, you might also need to change the hyperparameters.

#@title Double-click to view a possible solution.

my\_feature = "population" # Pick a feature other than "total\_rooms"

# Possibly, experiment with the hyperparameters.

learning\_rate = 0.05

epochs = 18

batch\_size = 3

# Don't change anything below.

my\_model = build\_model(learning\_rate)

weight, bias, epochs, rmse = train\_model(my\_model, training\_df,

my\_feature, my\_label,

epochs, batch\_size)

plot\_the\_model(weight, bias, my\_feature, my\_label)

plot\_the\_loss\_curve(epochs, rmse)

predict\_house\_values(10, my\_feature, my\_label)

#@title Double-click to view the answer.

# Training is not entirely deterministic, but population

# typically converges at a slightly higher RMSE than

# total\_rooms. So, population appears to be about

# the same or slightly worse at making predictions

# than total\_rooms.

Task 4: Define a synthetic feature

You have determined that total\_rooms and population were not useful features. That is, neither the total number of rooms in a neighborhood nor the neighborhood's population successfully predicted the median house price of that neighborhood. Perhaps though, the ratio of total\_rooms to population might have some predictive power. That is, perhaps block density relates to median house value.

To explore this hypothesis, do the following:

Create a [synthetic feature](https://developers.google.com/machine-learning/glossary/#synthetic_feature) that's a ratio of total\_rooms to population. (If you are new to pandas DataFrames, please study the [Pandas DataFrame Ultraquick Tutorial](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/pandas_dataframe_ultraquick_tutorial.ipynb?utm_source=linearregressionreal-colab&utm_medium=colab&utm_campaign=colab-external&utm_content=pandas_tf2-colab&hl=en).)

Tune the three hyperparameters.

Determine whether this synthetic feature produces a lower loss value than any of the single features you tried earlier in this exercise.

#@title Double-click to view a possible solution to Task 4.

# Define a synthetic feature

training\_df["rooms\_per\_person"] = training\_df["total\_rooms"] / training\_df["population"]

my\_feature = "rooms\_per\_person"

# Tune the hyperparameters.

learning\_rate = 0.06

epochs = 24

batch\_size = 30

# Don't change anything below this line.

my\_model = build\_model(learning\_rate)

weight, bias, epochs, mae = train\_model(my\_model, training\_df,

my\_feature, my\_label,

epochs, batch\_size)

plot\_the\_loss\_curve(epochs, mae)

predict\_house\_values(15, my\_feature, my\_label)

Based on the loss values, this synthetic feature produces a better model than the individual features you tried in Task 2 and Task 3. However, the model still isn't creating great predictions.

Task 5. Find feature(s) whose raw values correlate with the label

So far, we've relied on trial-and-error to identify possible features for the model. Let's rely on statistics instead.

A **correlation matrix** indicates how each attribute's raw values relate to the other attributes' raw values. Correlation values have the following meanings:

1.0: perfect positive correlation; that is, when one attribute rises, the other attribute rises.

-1.0: perfect negative correlation; that is, when one attribute rises, the other attribute falls.

0.0: no correlation; the two columns [are not linearly related](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FCorrelation_and_dependence%23%2Fmedia%2FFile%3ACorrelation_examples2.svg).

In general, the higher the absolute value of a correlation value, the greater its predictive power. For example, a correlation value of -0.8 implies far more predictive power than a correlation of -0.2.

The following code cell generates the correlation matrix for attributes of the California Housing Dataset:

# Generate a correlation matrix.

training\_df.corr()

The correlation matrix shows nine potential features (including a synthetic feature) and one label (median\_house\_value). A strong negative correlation or strong positive correlation with the label suggests a potentially good feature.

**Your Task:** Determine which of the nine potential features appears to be the best candidate for a feature?

#@title Double-click here for the solution to Task 5

# The median\_income correlates 0.7 with the label

# (median\_house\_value), so median\_income might be a

# good feature. The other seven potential features

# all have a correlation relatively close to 0.

# If time permits, try median\_income as the feature

# and see whether the model improves.

Correlation matrices don't tell the entire story. In later exercises, you'll find additional ways to unlock predictive power from potential features.

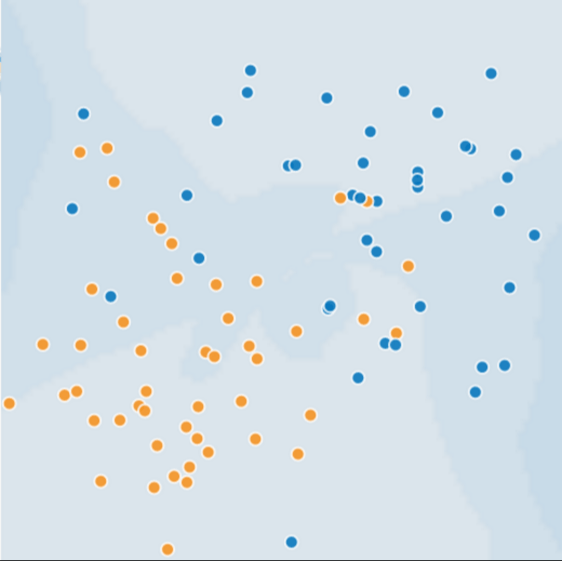
**Note:** Using median\_income as a feature may raise some ethical and fairness issues. Towards the end of the course, we'll explore ethical and fairness issues.

Generalization: Peril of Overfitting

This module focuses on generalization. In order to develop some intuition about this concept, you're going to look at three figures. Assume that each dot in these figures represents a tree's position in a forest. The two colors have the following meanings:

* The blue dots represent sick trees.
* The orange dots represent healthy trees.

With that in mind, take a look at Figure 1.

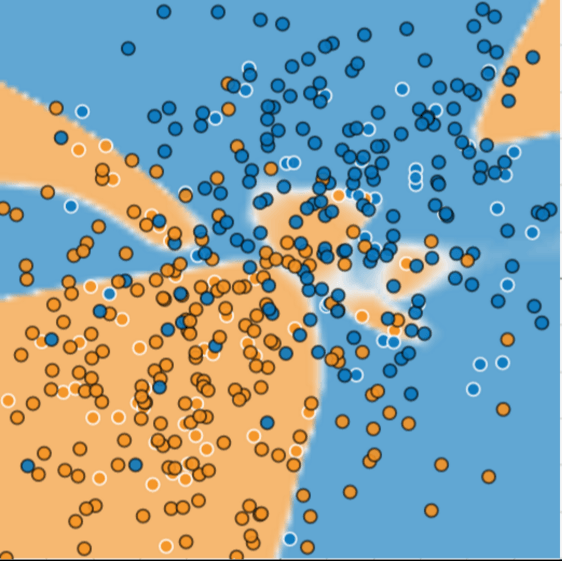


**Figure 1. Sick (blue) and healthy (orange) trees.**

Can you imagine a good model for predicting subsequent sick or healthy trees? Take a moment to mentally draw an arc that divides the blues from the oranges, or mentally lasso a batch of oranges or blues. Then, look at Figure 2, which shows how a certain machine learning model separated the sick trees from the healthy trees. Note that this model produced a very low loss.

## Low loss, but still a bad model?

Figure 3 shows what happened when we added new data to the model. It turned out that the model adapted very poorly to the new data. Notice that the model miscategorized much of the new data.



**Figure 3. The model did a bad job predicting new data.**

The model shown in Figures 2 and 3 **overfits** the peculiarities of the data it trained on. An overfit model gets a low loss during training but does a poor job predicting new data. If a model fits the current sample well, how can we trust that it will make good predictions on new data? As you'll see [later on](https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization), overfitting is caused by making a model more complex than necessary. The fundamental tension of machine learning is between fitting our data well, but also fitting the data as simply as possible.

Machine learning's goal is to predict well on new data drawn from a (hidden) true probability distribution. Unfortunately, the model can't see the whole truth; the model can only sample from a training data set. If a model fits the current examples well, how can you trust the model will also make good predictions on never-before-seen examples?

William of Ockham, a 14th century friar and philosopher, loved simplicity. He believed that scientists should prefer simpler formulas or theories over more complex ones. To put Ockham's razor in machine learning terms:

The less complex an ML model, the more likely that a good empirical result is not just due to the peculiarities of the sample.

In modern times, we've formalized Ockham's razor into the fields of **statistical learning theory** and **computational learning theory**. These fields have developed **generalization bounds**--a statistical description of a model's ability to generalize to new data based on factors such as:

* the complexity of the model
* the model's performance on training data

While the theoretical analysis provides formal guarantees under idealized assumptions, they can be difficult to apply in practice. Machine Learning Crash Course focuses instead on empirical evaluation to judge a model's ability to generalize to new data.

A machine learning model aims to make good predictions on new, previously unseen data. But if you are building a model from your data set, how would you get the previously unseen data? Well, one way is to divide your data set into two subsets:

* **training set**—a subset to train a model.
* **test set**—a subset to test the model.

Good performance on the test set is a useful indicator of good performance on the new data in general, assuming that:

* The test set is large enough.
* You don't cheat by using the same test set over and over.

## The ML fine print

The following three basic assumptions guide generalization:

* We draw examples **independently and identically** (**i.i.d**) at random from the distribution. In other words, examples don't influence each other. (An alternate explanation: i.i.d. is a way of referring to the randomness of variables.)
* The distribution is **stationary**; that is the distribution doesn't change within the data set.
* We draw examples from partitions from the **same distribution.**

In practice, we sometimes violate these assumptions. For example:

* Consider a model that chooses ads to display. The i.i.d. assumption would be violated if the model bases its choice of ads, in part, on what ads the user has previously seen.
* Consider a data set that contains retail sales information for a year. User's purchases change seasonally, which would violate stationarity.

When we know that any of the preceding three basic assumptions are violated, we must pay careful attention to metrics.

Conjuntos de treinamento e teste

Um **conjunto de teste** é um conjunto de dados usado para avaliar o modelo desenvolvido com base em um conjunto de treinamento.

como dividir dados

O módulo anterior introduziu a ideia de dividir seu conjunto de dados em dois subconjuntos:

* **conjunto de treinamento**: um subconjunto para treinar um modelo.
* **conjunto de teste**: um subconjunto para testar o modelo treinado.

Você pode imaginar o corte do único conjunto de dados da seguinte maneira:

Verifique se o conjunto de teste atende às duas condições a seguir:

* É grande o suficiente para gerar resultados estatisticamente significativos.
* Representa o conjunto de dados como um todo. Em outras palavras, não escolha um conjunto de teste com características diferentes do conjunto de treinamento.

Supondo que seu conjunto de teste atenda às duas condições anteriores, o objetivo é criar um modelo que generalize bem para novos dados. Nosso conjunto de teste serve como um proxy para novos dados. Por exemplo, considere a figura a seguir. Observe que o modelo aprendido com os dados de treinamento é muito simples. Esse modelo não é perfeito. Algumas previsões estão erradas. No entanto, esse modelo funciona tão bem quanto os dados de teste e de treinamento. Em outras palavras, esse modelo simples não supera os dados de treinamento.

**Figura 2. Validando o modelo treinado em relação aos dados de teste.**

**Nunca treine em dados de teste.** Se você estiver vendo resultados surpreendentemente bons nas suas métricas de avaliação, pode ser um sinal de que você está treinando acidentalmente no conjunto de teste. Por exemplo, a alta precisão pode indicar que os dados de teste vazaram para o conjunto de treinamento.

Por exemplo, considere um modelo que prevê se um e-mail é spam, usando a linha de assunto, o corpo do e-mail e o endereço de e-mail do remetente como recursos. Dividimos os dados em conjuntos de treinamento e teste, com uma divisão de 80 a 20. Após o treinamento, o modelo atinge 99% de precisão nos conjuntos de treinamento e de teste. Esperamos ter uma precisão menor no conjunto de teste. Portanto, analisamos os dados e descobrimos que muitos dos exemplos no conjunto de teste são cópias de exemplos do conjunto de treinamento (não reescrevemos entradas duplicadas para o mesmo e-mail de spam do banco de dados de entrada antes de dividir os dados). Nós, inadvertidamente, treinamos em alguns de nossos dados de teste e, como resultado, não estamos mais medindo a precisão da generalização do nosso modelo com novos dados.

Representação: Engenharia de atributos

Na programação tradicional, o foco está no código. Em projetos de machine learning, o foco muda para representação. Ou seja, uma maneira de os desenvolvedores aprimorarem um modelo é adicionando e melhorando os recursos dele.

## Como mapear dados brutos para atributos

O lado esquerdo da Figura 1 ilustra dados brutos de uma fonte de dados de entrada. O lado direito ilustra um **vetor de atributos**, que é o conjunto de valores de ponto flutuante que compõem os exemplos do seu conjunto de dados. **Engenharia de atributos** significa transformar dados brutos em um vetor de atributos. Espere passar um tempo significativo fazendo engenharia de atributos.

Muitos modelos de machine learning precisam representar os atributos como vetores de número real, já que os valores dos atributos precisam ser multiplicados pelos pesos do modelo.

**Figura 1. A engenharia de atributos mapeia dados brutos para atributos de ML.**

### Como mapear valores numéricos

Dados inteiros e de ponto flutuante não precisam de codificação especial, porque podem ser multiplicados por um peso numérico. Como sugerido na Figura 2, a conversão do valor inteiro bruto 6 para o valor do recurso 6.0 é trivial:

**Figura 2. Mapeando valores inteiros para valores de ponto flutuante.**

### Mapear valores categóricos

Os [atributos categóricos](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "categorical_data" \t "C) têm um conjunto distinto de valores possíveis. Por exemplo, pode ter um recurso chamado street\_name com opções que incluem:

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Como os modelos não podem multiplicar strings pelos pesos aprendidos, usamos engenharia de atributos para converter strings em valores numéricos.

Isso pode ser feito definindo um mapeamento com base nos valores do atributo, que vamos chamar de **vocabulário** de valores possíveis para números inteiros. Como nem todas as ruas do mundo aparecerão no nosso conjunto de dados, podemos agrupar todas as outras ruas em uma categoria completa, conhecida como **bucket OOV (fora do vocabulário)**.

Usando essa abordagem, veja como podemos mapear nossos nomes de ruas para números:

* mapear a Rua Charleston para 0
* mapear o North Shoreline Boulevard para 1
* mapa Shorebird Way para 2
* mapear Rengstorff Avenue para 3
* mapear todo o restante (OOV) para 4

No entanto, se incorporarmos esses números de índice diretamente ao nosso modelo, ele imporá algumas restrições que podem ser problemáticas:

* Vamos aprender um único peso que se aplica a todas as ruas. Por exemplo, se aprendermos um peso de 6 para street\_name, vamos multiplicá-lo por 0 para a Charleston Road, para 1 para a North Shoreline Boulevard, 2 para Shorebird Way e assim por diante. Considere um modelo que prevê preços de imóveis usando street\_name como atributo. É improvável que haja um ajuste de preço linear com base no nome da rua. Além disso, isso pressupõe que você fez o pedido das ruas com base no preço médio da casa. Nosso modelo precisa da flexibilidade de aprender diferentes pesos para cada rua que serão adicionados ao preço estimado usando os outros atributos.
* Não estamos considerando casos em que street\_name pode ter vários valores. Por exemplo, muitas casas estão localizadas na esquina de duas ruas, e não há como codificar essas informações no valor de street\_name se elas contiverem um único índice.

Para remover essas duas restrições, podemos criar um vetor binário para cada atributo categórico em nosso modelo que representa os valores da seguinte maneira:

* Para valores que se aplicam ao exemplo, defina os elementos vetoriais correspondentes como 1.
* Defina todos os outros elementos como 0.

A duração do vetor é igual ao número de elementos no vocabulário. Essa representação é chamada de **codificação one-hot** quando um único valor é 1, e uma **codificação multi-hot** quando vários valores são 1.

A Figura 3 ilustra uma codificação one-hot de uma rua específica: Shorebird Way. O elemento no vetor binário da maneira Shorebird tem um valor de 1, enquanto os elementos de todas as outras ruas têm valores de 0.

**Figura 3. mapear o endereço em codificação one-hot.**

Essa abordagem cria efetivamente uma variável booleana para cada valor do recurso (por exemplo, nome da rua). Aqui, se uma casa está na Shorebird Way, o valor binário é 1 apenas para Shorebird Way. Assim, o modelo usa apenas o peso para o caminho Shorebird.

Da mesma forma, se uma casa estiver no canto de duas ruas, dois valores binários serão definidos como 1, e o modelo usará os respectivos pesos.

### Representação esparsa

Suponha que você tenha 1.000.000 de nomes de rua diferentes no conjunto de dados que queira incluir como valores para street\_name. Criar explicitamente um vetor binário de 1.000.000 elementos em que apenas um ou dois elementos são verdadeiros é uma representação muito ineficiente em termos de tempo de armazenamento e computação ao processar esses vetores. Nessa situação, uma abordagem comum é usar uma [representação esparsa](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "sparse_representation" \t "C) em que apenas valores diferentes de zero são armazenados. Nas representações esparsas, um peso de modelo independente ainda é aprendido para cada valor de atributo, conforme descrito acima.

Representação: qualidades de bons atributos

Nós exploramos maneiras de mapear dados brutos em vetores de atributos adequados, mas isso é apenas parte do trabalho. Agora precisamos explorar que tipos de valores fazem bons atributos dentro desses vetores de atributos.

**Evite valores de recursos discretos raramente usados**

Bons valores de atributo precisam aparecer mais de cinco vezes em um conjunto de dados. Isso permite que um modelo aprenda como esse valor de atributo se relaciona com o rótulo. Ou seja, com muitos exemplos com o mesmo valor distinto, o modelo tem chance de ver o recurso em configurações diferentes e, por sua vez, determinar quando é um bom preditor para o rótulo. Por exemplo, um recurso house\_type provavelmente conteria muitos exemplos em que o valor dele era victorian:

✔house\_type: victorian

Por outro lado, se o valor de um recurso aparecer apenas uma vez ou muito raramente, o modelo não poderá fazer previsões com base nesse recurso. Por exemplo, unique\_house\_id é um recurso ruim, porque cada valor seria usado apenas uma vez, então o modelo não conseguiu aprender nada com ele:

✘unique\_house\_id: 8SK982ZZ1242Z

**Prefira significados claros e óbvios**

Cada recurso precisa ter um significado claro e óbvio para qualquer pessoa no projeto. Por exemplo, o nome bom do recurso a seguir é claramente nomeado e o valor faz sentido em relação ao nome:

✔house\_age\_years: 27

Por outro lado, o significado do valor do atributo a seguir é quase indecifrável para qualquer pessoa, menos para o engenheiro que o criou:

✘house\_age: 851472000

Em alguns casos, os dados ruidosos (em vez das escolhas erradas de engenharia) causam valores confusos. Por exemplo, a seguinte user\_age\_years veio de uma fonte que não verificou valores apropriados:

✘user\_age\_years: 277

**Não misturar valores "mágicos" com dados reais**

Atributos bons de ponto flutuante não contêm descontinuidades esforadas ou fora do intervalo, ou valores "mágicos" Por exemplo, suponha que um recurso contenha um valor de ponto flutuante entre 0 e 1. Portanto, os valores abaixo são bons:

✔quality\_rating: 0.82

quality\_rating: 0.37

No entanto, se um usuário não tiver inserido um quality\_rating, talvez o conjunto de dados representasse a ausência dele com um valor mágico como o seguinte:

✘quality\_rating: -1

Para marcar explicitamente os valores mágicos, crie um recurso booleano que indique se um quality\_rating foi fornecido ou não. Dê um nome a esse recurso booleano como is\_quality\_rating\_defined.

No recurso original, substitua os valores mágicos da seguinte maneira:

* Para variáveis que assumem um conjunto finito de valores (variáveis discretas), adicione um novo valor ao conjunto e use-o para indicar que está faltando o valor do recurso.
* Para variáveis contínuas, verifique se os valores ausentes não afetam o modelo usando o valor médio dos dados do recurso.

**Considere a instabilidade upstream.**

A definição de um recurso não muda com o tempo. Por exemplo, o valor a seguir é útil porque o nome da cidade *provavelmente* não mudará. Ainda é necessário converter uma string como "br/sao\_paulo" em um vetor one-hot.

✔city\_id: "br/sao\_paulo"

Mas coletar um valor inferido por outro modelo tem custos adicionais. Talvez o valor "quot;219" representa atualmente São Paulo, mas essa representação pode mudar facilmente em uma execução futura do outro modelo:

✘inferred\_city\_cluster: "219"

Representação: dados de limpeza

As árvores de maçã produzem uma mistura de ótimas frutas de frutas e vermes. No entanto, as maçãs em supermercados sofisticados exibem 100% de frutas perfeitas. Entre pomar e mercearia, alguém passa um tempo significativo removendo as maçãs ruins ou jogando uma pequena cera nas que podem ser recuperadas. Como engenheiro de ML, você passará muito tempo dedicando exemplos ruins e limpando aqueles que podem ser resgatados. Mesmo algumas maçãs" podem estragar um grande conjunto de dados.

### Como escalonar valores de atributos

**Escalonamento** significa converter valores de recursos de ponto flutuante do seu intervalo natural (por exemplo, 100 para 900) em um intervalo padrão (por exemplo, de 0 para 1 ou -1 para +1). Se um conjunto de atributos consistir em apenas um atributo, o escalonamento fornece pouco ou nenhum benefício prático. No entanto, se um conjunto de recursos consistir em vários atributos, o escalonamento de atributos fornecerá os seguintes benefícios:

* Ajuda o gradiente descendente a convergir mais rapidamente.
* Ajuda a evitar a armadilha de "NaN," em que um número no modelo se torna um [NaN](https://wikipedia.org/wiki/NaN) (por exemplo, quando um valor excede o limite de precisão de ponto flutuante durante o treinamento) e, devido a operações matemáticas, todos os outros números no modelo também se tornam um NaN.
* Ajuda o modelo a aprender os pesos apropriados para cada atributo. Sem o escalonamento de atributos, o modelo prestará muita atenção aos atributos que têm um intervalo maior.

Não é necessário atribuir a mesma escala a todos os recursos de pontos flutuantes. Nada muito ruim vai acontecer se o Recurso A for dimensionado de -1 a +1, enquanto o Recurso B for escalonado de -3 para +3. No entanto, o modelo reagirá mal se o Recurso B for escalonado de 5000 para 100000.

#### Clique no ícone de adição para saber mais sobre o escalonamento.

### Como lidar com outliers extremos

O gráfico a seguir representa um recurso chamado roomsPerPerson do [conjunto de dados de imóveis da Califórnia](https://developers.google.com/machine-learning/crash-course/california-housing-data-description?hl=pt-br). O valor de roomsPerPerson foi calculado dividindo o número total de salas de uma área pela população dessa área. O gráfico mostra que a grande maioria das áreas na Califórnia tem um ou dois quartos por pessoa. Veja o eixo X.

**Figura 4. Uma cauda lorny lonnnnnnng.**

Como podemos minimizar a influência desses outliers extremos? Uma maneira é usar o registro de cada valor:

**Figura 5. O escalonamento logarítmico ainda deixa uma cauda.**

O escalonamento de registros faz um trabalho um pouco melhor, mas ainda há uma cauda significativa de valores outliers. Vamos escolher outra abordagem. E se nós simplesmente "quot;cap" ou "clip" o valor máximo de roomsPerPerson em um valor arbitrário, digamos 4,0?

**Figura 6. Valores de recursos de corte em 4.0**

Cortar o valor do recurso em 4.0 não significa que ignoramos todos os valores maiores que 4.0. Em vez disso, isso significa que todos os valores maiores que 4,0 agora se tornam 4,0. Isso explica o engraçado morro em 4.0. Apesar dessa montanha, o conjunto de atributos em escala agora é mais útil que os dados originais.

### Agrupamento por classes

O gráfico a seguir mostra a prevalência relativa de casas em diferentes latitudes na Califórnia. Observe o clustering. Los Angeles está na latitude 34, e São Francisco está na latitude 38.

**Figura 7. Casas por latitude.**

No conjunto de dados, latitude é um valor de ponto flutuante. No entanto, não faz sentido representar latitude como um atributo de ponto flutuante no nosso modelo. Isso porque não há relação linear entre a latitude e os valores das casas. Por exemplo, as casas na latitude 35 não 3534 são mais (ou menos) caras que as casas na latitude 34. Ainda assim, as latitudes individuais provavelmente são um preditor muito bom de valores internos.

Para tornar a latitude um preditor útil, vamos dividir as latitudes em "bins" conforme sugerido pela figura a seguir:

**Figura 8. Valores de agrupamento.**

Em vez de ter um atributo de ponto flutuante, agora temos 11 atributos booleanos diferentes (LatitudeBin1, LatitudeBin2, ..., LatitudeBin11). Ter 11 atributos separados é um pouco insignificante, então vamos unificá-los em um único vetor de 11 elementos. Ao fazer isso, será possível representar a latitude 37.4 da seguinte forma:

[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

Graças à vinculação, nosso modelo pode aprender pesos completamente diferentes para cada latitude.

**Refinamento**

Até agora, presumimos que todos os dados usados para treinamento e teste fossem confiáveis. Na vida real, muitos exemplos em conjuntos de dados não são confiáveis devido a um ou mais dos seguintes motivos:

* **Valores omitidos.** Por exemplo, uma pessoa se esqueceu de inserir um valor para a idade da casa.
* **Exemplos duplicados.** Por exemplo, um servidor fez upload dos mesmos registros duas vezes por engano.
* **Ruim.** Por exemplo, uma pessoa rotulou incorretamente uma foto de uma árvore de carvalho como um bordo.
* **Valores de recurso inválidos.** Por exemplo, alguém digitou um dígito extra ou um termômetro escondido no sol.

Após a detecção, você normalmente os "corrigir" maus exemplos os remove do conjunto de dados. Para detectar valores omitidos ou exemplos duplicados, escreva um programa simples. Detectar valores de recursos ou rótulos inadequados pode ser muito mais difícil.

Além de detectar exemplos individuais ruins, você também precisa detectar dados inválidos na agregação. Os histogramas são um ótimo mecanismo para visualizar seus dados de maneira agregada. Além disso, coletar estatísticas como as seguintes pode ajudar:

* Máximo e mínimo
* Média e mediana
* Desvio padrão

Considere gerar listas dos valores mais comuns para atributos discretos. Por exemplo, faça a correspondência entre o número de exemplos com country:uk e o número esperado. O language:jp precisa ser a linguagem mais comum no conjunto de dados?

**Conheça seus dados**

Siga estas regras:

* Lembre-se de como você acha que seus dados devem ser.
* Verifique se os dados atendem a essas expectativas ou se você pode explicar por que eles não atendem a essas expectativas.
* Verifique se os dados de treinamento estão de acordo com outras fontes (por exemplo, painéis).

Trate seus dados com todos os cuidados que você faria com qualquer código essencial. Um bom ML depende de bons dados.

Cruzamentos de atributos

Um **cruzamento de atributos** é um **atributo sintético** formado pela multiplicação (cruzamento) de dois ou mais atributos. Combinações de cruzamentos de atributos podem fornecer habilidades preditivas melhores do que esses atributos conseguem fornecer individualmente.

Nas Figuras 1 e 2, imagine o seguinte:

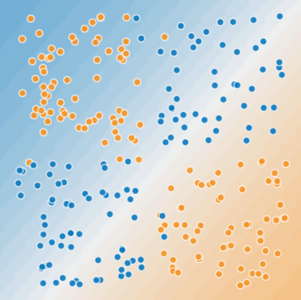
* Os pontos azuis representam árvores doentes.
* Os pontos laranja representam árvores saudáveis.



**Figura 1. Isso é um problema linear?**

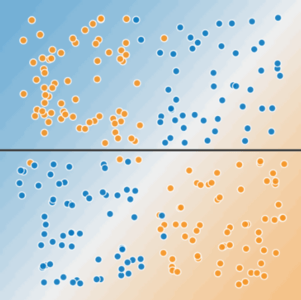
Vocês conseguem traçar uma linha que separa perfeitamente as árvores doentes das árvores saudáveis? Claro. Isso é um problema linear. A linha não será perfeita. Uma ou duas árvores doentes podem estar do lado "quo positivo", íntegro, mas sua linha será um bom preditor.

Veja a figura a seguir:



**Figura 2. Isso é um problema linear?**

Você consegue desenhar uma única linha reta que separa claramente as árvores doentes das árvores saudáveis? Não, isso não é possível. Esse problema não é linear. Qualquer linha desenhada será um preditor ruim da saúde da árvore.



**Figura 3. Uma única linha não consegue separar as duas classes.**

Para resolver o problema não linear mostrado na Figura 2, crie um cruzamento de atributos. Um **cruzamento de atributos** é um atributo sintético que codifica a não linearidade no espaço de atributos multiplicando dois ou mais atributos de entrada. O termo cruzado vem de [produto cruzado](https://wikipedia.org/wiki/Cross_product). Vamos criar um cruzamento de atributos com o nome x3 cruzando x1 e x2:

x3=x1x2

Tratamos esse cruzamento de atributos �3 recém-formado como qualquer outro atributo. A fórmula linear fica assim:

Y = b + w1x1 + w2x2 + w3x3

Um algoritmo linear pode aprender um peso para w3 da mesma forma que para w1 e w2. Em outras palavras, mesmo que w3 codifique informações não lineares, você não precisa mudar como o modelo linear é treinado para determinar o valor de w3.

### Tipos de cruzamentos de atributos

Podemos criar muitos tipos diferentes de cruzamentos de atributos. Exemplo:

* [A X B]: um cruzamento de atributos formado pela multiplicação de valores de dois atributos.
* [A x B x C x D x E]: um cruzamento de atributos formado pela multiplicação de valores de cinco atributos.
* [A x A]: um cruzamento de atributos formado pelo multiplicação do valor de um único atributo por si mesmo.

Graças ao [gradiente estocástico](https://developers.google.com/machine-learning/crash-course/reducing-loss/stochastic-gradient-descent?hl=pt-br), os modelos lineares podem ser treinados de forma eficiente. Consequentemente, complementar os modelos lineares escalonados com cruzamentos de atributos tem sido uma maneira eficiente de treinar em conjuntos de dados em grande escala.

Cruzamentos de atributos: cruzando vetores one-hot

Até agora, nosso foco se concentrou na interseção de dois recursos individuais de ponto flutuante. Na prática, os modelos de machine learning raramente são de atributos contínuos. No entanto, os modelos de machine learning geralmente fazem cruzamentos de vetores de atributos one-hot. Pense nos cruzamentos de atributos de vetores de atributo one-hot como conjunções lógicas. Por exemplo, suponha que temos dois recursos: país e idioma. Uma codificação one-hot de cada gera vetores com atributos binários que podem ser interpretados como country=USA, country=France ou language=English, language=Spanish. Depois, se fizer um cruzamento de atributos dessas codificações one-hot, você terá recursos binários que podem ser interpretados como conjunções lógicas, como:

  country:usa AND language:spanish

Como outro exemplo, suponha que você vincule a latitude e a longitude, produzindo vetores separados de recursos de cinco elementos one-hot. Por exemplo, determinada latitude e longitude podem ser representadas da seguinte maneira:

binned\_latitude = [0, 0, 0, 1, 0]

binned\_longitude = [0, 1, 0, 0, 0]

Suponha que você crie um cruzamento de atributos desses dois vetores de atributos:

binned\_latitude X binned\_longitude

Este cruzamento de atributos é um vetor one-hot de 25 elementos (24 zeros e 1). O 1 exclusivo na cruz identifica uma conjunção específica de latitude e longitude. Assim, seu modelo pode aprender associações específicas sobre essa conjunção.

Digamos que a latitude e a longitude sejam muito mais abrangentes, da seguinte maneira:

binned\_latitude(lat) = [

0 < lat <= 10

10 < lat <= 20

20 < lat <= 30

]

binned\_longitude(lon) = [

0 < lon <= 15

15 < lon <= 30

]

A criação de um cruzamento de atributos dessas classes aproximadas gera um atributo sintético que tem os seguintes significados:

binned\_latitude\_X\_longitude(lat, lon) = [

0 < lat <= 10 AND 0 < lon <= 15

0 < lat <= 10 AND 15 < lon <= 30

10 < lat <= 20 AND 0 < lon <= 15

10 < lat <= 20 AND 15 < lon <= 30

20 < lat <= 30 AND 0 < lon <= 15

20 < lat <= 30 AND 15 < lon <= 30

]

Agora, suponha que nosso modelo precise prever o nível de satisfação dos donos com os cães com base em dois recursos:

* Tipo de comportamento (latindo, choro, aconchego etc.)
* Hora do dia

Se criarmos um cruzamento de atributos desses dois atributos:

[behavior type X time of day]

acabaremos com uma capacidade de previsão muito maior que um dos atributos por conta própria. Por exemplo, se um cachorro chorar (felizmente) às 17h, quando o proprietário retorna do trabalho, provavelmente será um bom preditor de satisfação do proprietário. Chorar, talvez, às 3h, quando o proprietário dormia profundamente, provavelmente será um forte fator negativo de satisfação do proprietário.

Os alunos lineares se adaptam bem a dados enormes. O uso de cruzamentos de atributos em grandes conjuntos de dados é uma estratégia eficiente para aprender modelos altamente complexos. As [redes neurais](https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks?hl=pt-br) oferecem outra estratégia.

Cruzamentos de atributos: exercício de programação

No exercício a seguir, você analisará os cruzamentos de atributos no TensorFlow:

 [Representação com o cruzamento de atributos](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/representation_with_a_feature_cross.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=representation_tf2-colab&hl=pt-br) do Colab.

regularização para manter a simplicidade

**Regularização** significa penalizar a complexidade de um modelo para reduzir o overfitting.

regularização para manter a simplicidade: regularização L2

Considere a seguinte **curva de generalização**, que mostra a perda para o conjunto de treinamento e o conjunto de validação em relação ao número de iterações de treinamento.

**Figura 1. Perda do conjunto de treinamento e de validação.**

A Figura 1 mostra um modelo em que a perda de treinamento diminui gradualmente, mas a perda de validação acaba aumentando. Em outras palavras, essa curva de generalização mostra que o modelo está [sobreajustando](https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting?hl=pt-br) aos dados no conjunto de treinamento. Ao canalizar nossa [Ockham](https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting?hl=pt-br" \l "ockham) interna, talvez possa evitar o overfitting penalizando modelos complexos, um princípio chamado de **regularização** (link em inglês).

Em outras palavras, em vez de simplesmente tentar minimizar a perda (minimização do risco empírico):

minimize(Loss(Data|Model))

Agora, minimizaremos a perda e a complexidade, o que é chamado de **minimização do risco estrutural**:

minimize(Loss(Data|Model) + complexity(Model))

Nosso algoritmo de otimização de treinamento agora é uma função de dois termos: o **termo de perda**, que mede se o modelo atende aos dados, e o **termo de regularização**, que mede a complexidade do modelo.

O curso intensivo de aprendizado de máquina se concentra em duas maneiras comuns (e relacionadas) de pensar na complexidade do modelo:

* Complexidade do modelo como uma função dos pesos de todos os atributos no modelo.
* Complexidade do modelo como uma função do número total de atributos com pesos diferentes de zero. Em um [módulo posterior](https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization?hl=pt-br), abordamos essa abordagem.

Se a complexidade do modelo for uma função dos pesos, um peso de atributo com um alto valor absoluto é mais complexo que um peso de atributo com um valor absoluto baixo.

Podemos quantificar a complexidade usando a fórmula de regularização **L2**, que define o termo de regularização como a soma dos quadrados de todos os pesos de atributos:

L2 regularization term=||w||2^2=w1^2 + w2^2 +...+wn^2

Nessa fórmula, os pesos próximos de zero têm pouco efeito na complexidade do modelo, enquanto os pesos atípicos podem ter um impacto enorme.

Por exemplo, um modelo linear com os seguintes pesos:

{w1=0.2, w2=0.5, w3=5, w4=1, w5=0.25, w6=0.75}

Tem um termo de regularização *L2* de 26.915:

w1^2 + w2^2 + w3^2 + w4^2 + w5^2 + w6^2

=0.22+0.52+52+12+0.252+0.752

=0.04+0.25+25+1+0.0625+0.5625

=26.915

No entanto, w3 (negrito acima), com um valor quadrado de 25, contribui para quase toda a complexidade. A soma dos quadrados de todos os outros cinco pesos é somada a 1,915 ao termo de regularização *L2*.

regularização para manter a simplicidade: lambda

Os desenvolvedores de modelos ajustam o impacto geral do termo de regularização multiplicando o valor por um escalar conhecido como **lambda**, também chamado de **taxa de regularização**. Ou seja, os desenvolvedores de modelos querem fazer o seguinte:

minimize(Loss(Data|Model)+� complexity(Model))

A regularização de *L2* tem o seguinte efeito em um modelo

* Encoraja valores de peso a 0 (mas não exatamente a 0)
* Encoraja a média dos pesos a 0, com uma distribuição normal (em forma de sino ou gaussiana).

Aumentar o valor da lambda dá força ao efeito de regularização. Por exemplo, o histograma de pesos de um alto valor de lambda pode ter a aparência mostrada na Figura 2.

**Figura 2. Histograma de pesos.**

A redução do valor da lambda tende a produzir um histograma mais plano, como mostrado na Figura 3.

**Figura 3. Histograma de pesos produzido por um valor lambda menor.**

Ao escolher um valor lambda, o objetivo é encontrar o equilíbrio certo entre simplicidade e ajuste dos dados de treinamento:

* Se o valor da lambda for muito alto, seu modelo será simples, mas você correrá o risco de *subajustar* seus dados. Seu modelo não aprenderá o suficiente sobre os dados de treinamento para fazer previsões úteis.
* Se o valor da lambda for muito baixo, seu modelo será mais complexo e você correrá o risco de *overfitting* com seus dados. Seu modelo aprenderá muito sobre as particularidades dos dados de treinamento e não poderá generalizar para novos dados.

**Observação**: definir lambda como zero remove a regularização completamente. Nesse caso, o treinamento se concentra exclusivamente na minimização da perda, o que representa o maior risco de overfitting possível.

O valor ideal de lambda produz um modelo que generaliza bem para dados novos e nunca vistos. Infelizmente, esse valor ideal de lambda depende dos dados, portanto, você precisará fazer alguns ajustes de.

regularização para manter a simplicidade: exercício do Playground (Regularização de L2)

## Examinar a regularização de *L2*

Este exercício contém um conjunto de dados de treinamento com ruído. Nesse tipo de ambiente, o overfitting é uma preocupação real. Felizmente, a regularização pode ajudar.

Este exercício consiste em três tarefas relacionadas. Para simplificar as comparações entre as três tarefas, execute cada tarefa em uma guia separada.

* **Tarefa 1**:execute o modelo conforme fornecido por pelo menos 500 períodos. Atenção:
  + Perda de teste.
  + O delta entre a perda de teste e a perda de treinamento.
  + Os pesos aprendidos dos atributos e dos cruzamentos de atributos. A espessura relativa de cada linha executada de FEATURES a OUTPUT representa o peso aprendido desse atributo ou cruzamento de atributos. Para encontrar os valores exatos, passe o cursor sobre cada linha.
* **Tarefa 2**: *faça essa tarefa em uma guia separada*. Aumente a taxa de regularização de 0 para 0,3. Em seguida, execute o modelo por pelo menos 500 períodos e encontre respostas para as seguintes perguntas:
  + Como a perda de teste na tarefa 2 difere da perda de teste na tarefa 1?
  + Como o delta entre a perda de teste e a perda de treinamento na tarefa 2 é diferente do da tarefa 1?
  + Como os pesos aprendidos de cada atributo e cruzamento de atributos são diferentes da tarefa 2 para a tarefa 1?
  + O que dizem os resultados sobre a complexidade do modelo?
* **Tarefa 3:** tente a taxa de regularização para encontrar o valor ideal.

Regressão logística

Em vez de prever *exatamente* 0 ou 1, a **regressão logística** gera uma probabilidade, um valor *entre* 0 e 1, exclusivo. Por exemplo, considere um modelo de regressão logística para detecção de spam. Se o modelo inferir um valor de 0,932 em uma determinada mensagem de e-mail, isso implica uma probabilidade de 93,2% de que a mensagem seja spam. Mais precisamente, isso significa que, no limite de exemplos de treinamento *infinito*, o conjunto de exemplos para os quais o modelo prevê 0,932 será, na verdade, spam 93,2% das vezes, e os 6,8% restantes não.

Regressão logística: como calcular uma probabilidade

Muitos problemas exigem uma estimativa de probabilidade como saída. A regressão logística é um mecanismo extremamente eficiente para calcular probabilidades. Na prática, é possível usar a probabilidade retornada de uma destas duas maneiras:

* "Como está"
* Convertido para uma categoria binária.

Vamos analisar como podemos usar a probabilidade "no estado em que se encontram." Suponha que criemos um modelo de regressão logística para prever a probabilidade de um cachorro latir durante a noite. Vamos chamá-lo de probabilidade:

p(bark|rigℎt)

Se o modelo de regressão logística previr p(bark|rigℎt)=0.05, ao longo de um ano, os proprietários dos cães precisarão ser acordados aproximadamente 18 vezes:

Startled = p(bark|rigℎt) ⋅ nigℎts

= 0.05⋅365

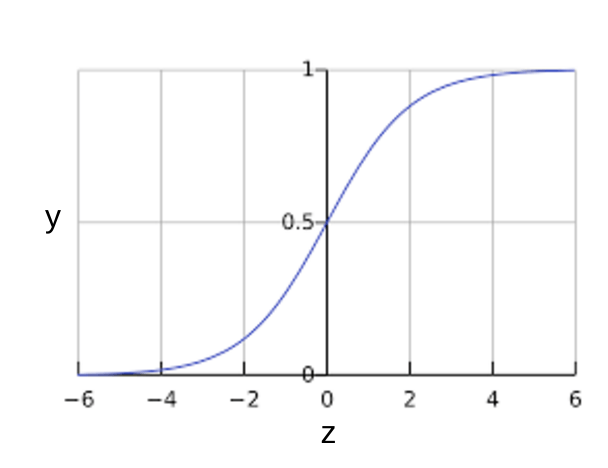
=18

Em muitos casos, você mapeará a saída da regressão logística na solução para um problema de classificação binária em que o objetivo é prever corretamente um dos dois rótulos possíveis (por exemplo, "spam" ou "não é spam"). Um próximo [módulo](https://developers.google.com/machine-learning/crash-course/classification/video-lecture?hl=pt-br) se concentra nisso.

Você pode estar se perguntando como um modelo de regressão logística pode garantir uma saída que sempre fique entre 0 e 1. Como acontece, uma **função sigmoide**, definida a seguir, produz um resultado com as mesmas características:

y = 1 / 1 + e ˆ−z

A função sigmoide gera o seguinte gráfico:



**Figura 1: função sigmoide.**

Se z representar a saída da camada linear de um modelo treinado com regressão logística, sigmoid(z) vai gerar um valor (uma probabilidade) entre 0 e 1. Em termos matemáticos:

y′ =1 / 1 + e ˆ−z

onde:

* y′ é a saída do modelo de regressão logística de um exemplo específico.
* Z = b + w1x1 + w2x2 +…+ WnXn
  + Os valores w são os pesos aprendidos do modelo, e b é o viés.
  + Os valores x são os valores de recurso de um exemplo específico.

Observe que z também é conhecido como *log-odds*, porque o inverso dos sigmoides z pode ser definido como o registro da probabilidade do rótulo 1 (por exemplo, "latidos de cachorro") dividido pela probabilidade do identificador 0(por exemplo, "cachorro não/#39;não latir"):

z = log(y / 1 − y)

Veja a função sigmoide com identificadores de ML:

**Figura 2: saída da regressão logística**.

Regressão logística: perda e regularização

## Função de perda para regressão logística

A função de perda para regressão linear é a perda ao quadrado. A função de perda para regressão logística é a **Log Loss**, definida da seguinte maneira:

Log Loss = ∑(x, y)∈D [− ylog(y′) − (1−y)log(1−y′)]

onde:

* (x, y)∈D é o conjunto de dados que contém muitos exemplos rotulados, que são (x, y) pares.
* y é o rótulo em um exemplo rotulado. Como esta é uma regressão logística, todos os valores de y precisam ser 0 ou 1.
* y′ é o valor previsto (algo entre 0 e 1), dado o conjunto de atributos em x.

## **Regularização na regressão logística**

A [regularização](https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/video-lecture?hl=pt-br) (link em inglês) é extremamente importante na modelagem de regressão logística. Sem a regularização, a natureza assintotônica da regressão logística continuaria a perder em direção a 0 em dimensões elevadas. Consequentemente, a maioria dos modelos de regressão logística usa uma das duas estratégias a seguir para reduzir a complexidade do modelo:

* L2 regularização.
* Parada antecipada, ou seja, limitar o número de etapas de treinamento ou a taxa de aprendizado.

Falaremos sobre uma terceira estratégia, a L1 regularização, em um [módulo posterior](https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/video-lecture?hl=pt-br).

Imagine que você atribua um ID exclusivo a cada exemplo e mapeie esses IDs para um atributo próprio. Se você não especificar uma função de regularização, o modelo ficará completamente overfitting. Isso porque o modelo tentaria gerar perda para zero em todos os exemplos e nunca chegaria lá, levando os pesos para cada atributo indicador para +infinito ou -infinito. Isso pode acontecer em dados de alta dimensão com cruzamentos de atributos, quando há uma enorme massa de cruzamentos raros que acontecem apenas em um exemplo cada.

Felizmente, o uso de L2 ou parada antecipada vai evitar esse problema.

Classificação

Este módulo mostra como a regressão logística pode ser usada em tarefas de classificação e explora como avaliar a eficácia dos modelos de classificação.

Classificação: limite

A regressão logística retorna uma probabilidade. Você pode usar a probabilidade retornada "como está quot; (por exemplo, a probabilidade de o usuário clicar neste anúncio é 0,00023) ou converter a probabilidade retornada em um valor binário (por exemplo, este e-mail é spam).

Um modelo de regressão logística que retorna 0,9995 para uma mensagem de e-mail específica prevê que é muito provável que seja spam. Por outro lado, outra mensagem de e-mail com uma pontuação de previsão de 0,0003 nesse mesmo modelo de regressão logística provavelmente não é spam. No entanto, o que acontece com uma mensagem de e-mail com uma pontuação de previsão de 0,6? Para mapear um valor de regressão logística para uma categoria binária, é preciso definir um **limite de classificação**, também chamado de **limite de decisão**. Um valor acima desse limite indica "spam" um valor abaixo indica "não é spam." É tentador presumir que o limite de classificação sempre deve ser 0,5, mas os limites dependem do problema e, portanto, são valores que precisam ser ajustados.

As seções a seguir examinam melhor as métricas que podem ser usadas para avaliar as previsões de um modelo de classificação, bem como o impacto de alterar o limite de classificação nessas previsões.

**Observação:** "Tuning" um limite para regressão logística é diferente de ajuste de hiperparâmetros, como taxa de aprendizado. Parte de um limite é avaliar quanto você sofrerá por cometer um erro. Por exemplo, rotular uma mensagem que não é spam por engano é muito ruim. No entanto, rotular uma mensagem de spam por engano é desagradável, mas raramente é o fim do job.

Classificação: verdadeiro x falso x positivo ou negativo

Nesta seção, vamos definir os elementos básicos das métricas que serão usadas para avaliar os modelos de classificação. Mas primeiro, uma fábula:

**Uma fábula de Aesop: The Boy Who Created Wolf (compactada)**

Um menino pastando está entediado cuidando do bando da cidade. Para se divertir, ele grita: "Lobo!", mesmo que nenhum lobo esteja visível. Os aldeões correm para proteger o bando, mas ficam muito bravos quando percebem que o menino estava brincando.

[Repetir o parágrafo anterior N vezes].

Uma noite, o menino pastor viu um lobo real se aproximando do bando e chama, "quot;Wolf!" Os vilarejos se recusam a se enganar e permanecem em casa. O lobo faminto transforma o bando em costeletas de cordeiro. A cidade fica faminta. O pânico chega.

Vamos fazer as seguintes definições:

* "Wolf" é uma **classe positiva**.
* "Sem lobo" é uma **classe negativa**.

Podemos resumir nosso modelo de previsão e previsão usando uma [matriz de confusão](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "confusion_matrix) 2x2 que representa os quatro resultados possíveis:

|  |  |
| --- | --- |
| Verdadeiro positivo (VP):   * Realidade: um lobo ameaçado. * O Shepherd disse: "quot;Wolf." * Resultado: Shepherd é um herói. | Falso positivo (FP):   * Realidade: nenhum lobo está ameaçado. * O Shepherd disse: "quot;Wolf." * Resultado: os moradores estão bravos com o pastor por acordá-los. |
| Falso negativo (FN):   * Realidade: um lobo ameaçado. * O Shepherd disse: "Sem lobo." * Resultado: o lobo comeu todas as ovelhas. | Verdadeiro negativo (VN):   * Realidade: nenhum lobo está ameaçado. * O Shepherd disse: "Sem lobo." * Resultado: todos estão bem. |

Um **verdadeiro positivo** é um resultado em que o modelo prevê corretamente a classe positiva. Da mesma forma, um **verdadeiro negativo** é um resultado em que o modelo corretamente prevê a classe negativo.

Um **falso positivo** é um resultado em que o modelo prevê incorretamente a classe positiva. E um **falso negativo** é um resultado em que o modelo prevê incorretamente a classe negativa.

Nas seções a seguir, veremos como avaliar modelos de classificação usando métricas derivadas desses quatro resultados.

Classificação: acurácia

A acurácia é uma métrica para avaliar modelos de classificação. Internamente, a **precisão** é a fração das previsões que o modelo acertou. Formalmente, a precisão tem a seguinte definição:

Accuracy = Number of correct predictions / Total number of predictions

Para classificação binária, a acurácia também pode ser calculada em termos de positivos e negativos, da seguinte maneira:

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

Em que TP = Verdadeiros positivos, TN = Verdadeiros negativos, FP = Falsos positivos e FN = Falsos negativos.

Vamos tentar calcular a acurácia do modelo a seguir que classificou 100 tumores como [malignante](https://wikipedia.org/wiki/Malignancy) (a classe positiva) ou [benign](https://wikipedia.org/wiki/Benign_tumor) (a classe negativa):

|  |  |
| --- | --- |
| Verdadeiro positivo (VP):   * Realidade: maligna * Modelo de ML previsto: maligno * **Número de resultados de VP: 1** | Falso positivo (FP):   * Realidade: benigno * Modelo de ML previsto: maligno * **Número de resultados de FP: 1** |
| Falso negativo (FN):   * Realidade: maligna * Modelo de ML previsto: Benign * **Número de resultados de FN: 8** | Verdadeiro negativo (VN):   * Realidade: benigno * Modelo de ML previsto: Benign * **Número de resultados de TN: 90** |

Accuracy = TP + TN / TP + TN + FP + FN = 1 + 901 + 90 + 1 + 8 = 0.91

A acurácia é de 0,91, ou 91% (91 previsões corretas de 100 exemplos totais). Isso significa que o classificador de tumor está fazendo um ótimo trabalho para identificar malignidades, certo?

Na verdade, vamos fazer uma análise mais precisa dos positivos e negativos para obter mais insights sobre o desempenho do nosso modelo.

Dos 100 exemplos de tumor, 91 são benignos (90 TNs e 1 FP) e 9 são malignos (1 TP e 8 FNs).

Dos 91 tumores benignos, o modelo identifica corretamente 90 como beninos. Isso é positivo. No entanto, dos 9 tumores malignos, o modelo identifica apenas 1 como maligno, um resultado terrível, já que 8 de 9 malignidades não são diagnosticadas.

Embora a precisão de 91% possa parecer boa à primeira vista, outro modelo de classificador de tumor que sempre prevê o benigno alcançaria exatamente a mesma precisão (91/100 previsões corretas) nos nossos exemplos. Em outras palavras, nosso modelo não é melhor do que um que não tem capacidade preditiva de distinguir os tumors malignos dos tumores benignos.

A precisão por si só não conta a história completa quando você está trabalhando com um **conjunto de dados desequilibrado**, como este, em que há uma diferença significativa entre o número de rótulos positivos e negativos.

Na próxima seção, veremos duas métricas melhores para avaliar problemas de desequilíbrio de classes: precisão e recall.

Classificação: precisão e recall

## **Precisão**

A **precisão** tenta responder à seguinte pergunta:

Qual a proporção de identificações positivas estava correta?

A precisão é definida da seguinte maneira:

Precision = TP / TP +FP

**Observação**: um modelo que não produz falsos positivos tem precisão de 1,0.

Vamos calcular a precisão do nosso modelo de ML na [seção anterior](https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=pt-br) que analisa tumores:

|  |  |
| --- | --- |
| Verdadeiros positivos (VPs): 1 | Falsos positivos (FPs): 1 |
| Falsos negativos (FNs): 8 | Verdadeiros negativos (VNs): 90 |

Precision = TP / TP + FP = 1 / 1 + 1 = 0.5

Nosso modelo tem uma precisão de 0,5.Em outras palavras, quando prevê que um tumor é maligno, é correto em 50% do tempo.

## **Recall**

O **recall** tenta responder à seguinte pergunta:

Qual proporção de positivos verdadeiros foi identificada corretamente?

Matematicamente, o recall é definido da seguinte forma:

Recall = TP / TP + FN

**Observação**:um modelo que não produz falsos negativos tem um recall de 1,0.

Vamos calcular o recall do nosso classificador de tumor:

|  |  |
| --- | --- |
| Verdadeiros positivos (VPs): 1 | Falsos positivos (FPs): 1 |
| Falsos negativos (FNs): 8 | Verdadeiros negativos (VNs): 90 |

Recall=TP / TP + FN = 1 / 1 + 8 = 0.11

Nosso modelo teve um recall de 0,11.Em outras palavras, identifica corretamente 11% de todos os tumores malignos.

## Precisão e recall: um combate

Para avaliar totalmente a eficácia de um modelo, é necessário examinar **ambos** a precisão e o recall. Infelizmente, a precisão e o recall geralmente estão sobrecarregados. Ou seja, melhorar a precisão geralmente reduz o recall e vice-versa. Explore essa ideia analisando a figura a seguir, que mostra 30 previsões feitas por um modelo de classificação de e-mail. Aquelas à direita do limite de classificação são classificadas como spam, enquanto aquelas à esquerda são classificadas como spam, não.

**Figura 1. classificar mensagens de e-mail como spam ou não.**

Vamos calcular a precisão e o recall com base nos resultados mostrados na Figura 1:

|  |  |
| --- | --- |
| Verdadeiros positivos (VP): 8 | Falsos positivos (FP): 2 |
| Falsos negativos (FN): 3 | Verdadeiros negativos (VN): 17 |

A precisão mede a porcentagem de **e-mails sinalizados como spam** que foram classificados corretamente, ou seja, a porcentagem de pontos à direita da linha de limite que é verde na Figura 1:

Precision = TP / TP + FP = 8 / 8 + 2 = 0.8

O recall mede a porcentagem de **e-mails de spam reais** que foram classificados corretamente, ou seja, a porcentagem de pontos verdes à direita da linha de limite na Figura 1:

Recall = TP / TP + FN = 8 / 8 + 3 = 0.73

A Figura 2 ilustra o efeito de aumento do limite de classificação.

**Figura 2. Aumentar o limite de classificação.**

O número de falsos positivos diminui, mas os falsos negativos aumentam. Como resultado, a precisão aumenta, enquanto o recall diminui:

|  |  |
| --- | --- |
| Verdadeiros positivos (VP): 7 | Falsos positivos (FP): 1 |
| Falsos negativos (FN): 4 | Verdadeiros negativos (VN): 18 |

Precision = TP / TP + FP = 7 / 7 + 1 = 0.88

Recall = TP / TP + FN = 7 / 7 + 4 = 0.64

Por outro lado, a Figura 3 ilustra o efeito de diminuição do limite de classificação (da posição original na Figura 1).

**Figura 3. Diminuindo o limite de classificação.**

Os falsos positivos aumentam e os falsos negativos diminuem. Como resultado, dessa vez, a precisão diminui e o recall aumenta:

|  |  |
| --- | --- |
| Verdadeiros positivos (VP): 9 | Falsos positivos (FP): 3 |
| Falsos negativos (FN): 2 | Verdadeiros negativos (VN): 16 |

Precision = TP / TP + FP = 9 / 9 + 3 = 0.75

Recall = TP / TP + FN = 9 / 9 + 2 = 0.82

Diversas métricas foram desenvolvidas que dependem da precisão e do recall. Por exemplo, consulte [Pontuação F1](https://wikipedia.org/wiki/F1_score).

Classificação: curva ROC e AUC

## Curva ROC

Uma **curva de ROC** (**curva de característica de operação do receptor**) é um gráfico que mostra o desempenho de um modelo de classificação em todos os limiares de classificação. Essa curva representa dois parâmetros:

* Taxa de verdadeiro positivo
* Taxa de falso positivo

**Taxa de verdadeiro positivo** (**TPR**) é um sinônimo de recall e, portanto, é definida da seguinte maneira:

TPR = TP / TP + FN

A **Taxa de falso positivo** (**FPR**) é definida da seguinte maneira:

FPR = FP / FP + TN

Uma curva ROC mostra TPR x FPR em diferentes limiares de classificação. Reduzir o limite de classificação classifica mais itens como positivos, o que aumenta os falsos positivos e verdadeiros positivos. A figura a seguir mostra uma curva típica de ROC.

**Figura 4. Taxas de VP e FP em diferentes limites de classificação.**

Para calcular os pontos em uma curva ROC, poderíamos avaliar um modelo de regressão logística muitas vezes com limites de classificação diferentes, mas isso seria ineficiente. Felizmente, há um algoritmo eficiente e baseado em classificação que pode fornecer essas informações para nós, chamado AUC.

Para calcular os pontos em uma curva ROC, poderíamos avaliar um modelo de regressão logística muitas vezes com limites de classificação diferentes, mas isso seria ineficiente. Felizmente, há um algoritmo eficiente e baseado em classificação que pode fornecer essas informações para nós, chamado AUC.

## AUC: Área sob a curva ROC

**AUC** significa "área" sob a curva ROC. Ou seja, a AUC mede toda a área bidimensional abaixo da curva ROC inteira (pense no cálculo integral) de (0,0) a (1,1).

A AUC oferece uma medida agregada de desempenho em todos os limites de classificação possíveis. Uma maneira de interpretar a AUC é a probabilidade de o modelo classificar um exemplo positivo aleatório mais alto do que um exemplo negativo aleatório. Por exemplo, considerando os exemplos a seguir, que são organizados da esquerda para a direita em ordem crescente de previsões de regressão logística:

A AUC representa a probabilidade de um exemplo aleatório positivo (verde) estar posicionado à direita de um exemplo negativo aleatório (vermelho).

O AUC varia no valor de 0 a 1. Um modelo com previsões 100% incorretas tem uma AUC de 0,0, e uma com previsões 100% corretas tem uma AUC de 1,0.

A AUC é recomendável pelos seguintes motivos:

* A AUC é **invariante em escala**. Ele mede a precisão das classificações em vez dos valores absolutos.
* A AUC é **classification-threshold-invariant**. Ela mede a qualidade das previsões do modelo, independentemente do limite de classificação escolhido.

No entanto, ambos os motivos vêm com ressalvas, que podem limitar a utilidade da AUC em determinados casos de uso:

* **Nem sempre uma variação é adequada quando se trata de escalonamentos.** Por exemplo, às vezes precisamos de resultados de probabilidade bem calibrados, e a AUC não vai nos contar isso.
* **A incompatibilidade de limite de classificação nem sempre é desejada.** Nos casos em que há amplas disparidades no custo de falsos negativos em comparação a falsos positivos, pode ser essencial minimizar um tipo de erro de classificação. Por exemplo, ao fazer a detecção de spam de e-mail, é provável que você queira priorizar a minimização dos falsos positivos, mesmo que isso resulte em um aumento significativo dos falsos negativos. A AUC não é uma métrica útil para esse tipo de otimização.

Classificação: viés de previsão

Previsões de regressão logística precisam ser imparciais. Ou seja:

"média de previsões" ≈ "média de observações"

O **viés de previsão** é uma quantidade que mede a distância entre essas duas médias. Ou seja:

prediction bias=average of predictions − average of labels in data set

**Observação**: "Viés de previsão" é uma quantidade diferente de [viés](https://developers.google.com/machine-learning/crash-course/descending-into-ml?hl=pt-br) (b em wx + b).

Um viés de previsão significativo que não seja zero diz que há um bug em algum lugar do seu modelo, porque indica que ele está errado sobre a frequência com que os identificadores positivos.

Por exemplo, digamos que saibamos que, em média, 1% de todos os e-mails são spam. Se não soubermos nada sobre um determinado e-mail, devemos prever que ele tenha 1% de probabilidade de ser spam. Da mesma forma, um bom modelo de spam deve prever em média que os e-mails têm 1% de probabilidade de serem spam. Em outras palavras, se calcularmos a probabilidade estimada de cada e-mail ser spam, o resultado será 1%. Se, em vez disso, a previsão média do modelo tiver 20% de probabilidade de ser spam, poderemos concluir que ele exibe um viés de previsão.

As possíveis causas da tendência de previsão são as seguintes:

* Conjunto de recursos incompleto
* Conjunto de dados com ruído
* Pipeline com bugs
* Amostra de treinamento tendenciosa
* Regularização muito forte

Um viés de previsão pode ser tentado corrigir o pós-processamento do modelo aprendido, ou seja, adicionando uma **camada de calibração** que ajusta a saída do modelo para reduzir o viés de previsão. Por exemplo, se o modelo tiver um viés de +3%, será possível adicionar uma camada de calibração que diminua a previsão média em 3%. No entanto, adicionar uma camada de calibração não é uma boa ideia pelos seguintes motivos:

* Você está corrigindo o sintoma em vez da causa.
* Você criou um sistema mais frágil que precisa manter atualizado.

Se possível, evite camadas de calibração. Projetos que usam camadas de calibração tendem a se tornar dependente delas, usando camadas de calibração para corrigir todos os sins do modelo. Por fim, manter as camadas de calibração pode ser um pesadelo.

**Observação**: um bom modelo geralmente tem um viés próximo de zero. Dito isso, um viés de previsão baixo não prova que o modelo é bom. Um modelo muito ruim pode ter um viés de previsão zero. Por exemplo, um modelo que apenas prevê o valor médio de todos os exemplos seria um modelo ruim, apesar de ter viés zero.

## viés na previsão e no bucket

A regressão logística prevê um valor entre 0 e 1. No entanto, todos os exemplos rotulados são exatamente 0 (que significa, por exemplo, "não é spam") ou exatamente 1 (ou seja, "quot;spam""). Portanto, ao examinar o viés de previsão, não é possível determinar o viés de previsão com base em apenas um exemplo. É necessário examinar o viés de previsão em um exemplo de bucket. Ou seja, o viés de previsão para regressão logística faz sentido apenas ao agrupar exemplos suficientes para poder comparar um valor previsto (por exemplo, 0,392) com os valores observados (por exemplo, 0,394).

É possível formar buckets das seguintes maneiras:

* Dividir linearmente as previsões de destino
* Como formar quantis.

Considere o seguinte gráfico de calibração de um modelo específico. Cada ponto representa um bucket de 1.000 valores. Os eixos têm os seguintes significados:

* O eixo x representa a média dos valores previstos pelo modelo para esse bucket.
* O eixo y representa a média real de valores no conjunto de dados desse bucket.

Os dois eixos são balanças logarítmicas.

**Figura 8. Curva de viés de previsão (escalas logarítmicas)**

Por que as previsões são tão ruins para apenas parte do modelo? Veja algumas possibilidades:

* O conjunto de treinamento não representa adequadamente determinados subconjuntos do espaço de dados.
* Alguns subconjuntos do conjunto de dados são mais ruidosos que outros.
* O modelo é [regularmente](https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/video-lecture?hl=pt-br). Considere reduzir o valor de [lambda](https://developers.google.com/machine-learning/glossary?hl=pt-br" \l "lambda).

Classificação binária: exercício de programação

No exercício a seguir, você aprenderá a classificação binária no TensorFlow:

 Exercício de [classificação binária](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/binary_classification.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=binary_classification_tf2-colab&hl=pt-br" \t "C) do Colab

Os exercícios de programação são executados diretamente no navegador, sem necessidade de configuração, usando a plataforma [Colaboratory](https://colab.research.google.com/?hl=pt-br) (em inglês). O Colaboratory é compatível com a maioria dos principais navegadores e é testado mais detalhadamente em versões do Chrome para computador e Chrome. Se você preferir fazer o download e executar os exercícios off-line, consulte [estas instruções](https://developers.google.com/machine-learning/crash-course/running-exercises-locally?hl=pt-br) para configurar um ambiente local.

Binary Classification

So far, you've only created regression models. That is, you created models that produced floating-point predictions, such as, "houses in this neighborhood costs N thousand dollars." In this Colab, you'll create and evaluate a binary [classification model](https://developers.google.com/machine-learning/glossary/" \l "classification_model" \t "_blank). That is, you'll create a model that answers a binary question. In this exercise, the binary question will be, "Are houses in this neighborhood above a certain price?"

Learning Objectives:

After doing this Colab, you'll know how to:

Convert a regression question into a classification question.

Modify the classification threshold and determine how that modification influences the model.

Experiment with different classification metrics to determine your model's effectiveness.

The Dataset

Like several of the previous Colabs, this Colab uses the [California Housing Dataset](https://developers.google.com/machine-learning/crash-course/california-housing-data-description" \t "_blank).

Call the import statements

The following code imports the necessary modules.

Load the imports

#@title Load the imports

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import layers

from matplotlib import pyplot as plt

# The following lines adjust the granularity of reporting.

pd.options.display.max\_rows = 10

pd.options.display.float\_format = "{:.1f}".format

# tf.keras.backend.set\_floatx('float32')

print("Ran the import statements.")

Load the datasets from the internet

The following code cell loads the separate .csv files and creates the following two pandas DataFrames:

train\_df, which contains the training set

test\_df, which contains the test set

train\_df = pd.read\_csv("https://download.mlcc.google.com/mledu-datasets/california\_housing\_train.csv")

test\_df = pd.read\_csv("https://download.mlcc.google.com/mledu-datasets/california\_housing\_test.csv")

train\_df = train\_df.reindex(np.random.permutation(train\_df.index)) # shuffle the training set

Unlike some of the previous Colabs, the preceding code cell did not scale the label (median\_house\_value). The following section ("Normalize values") provides an alternative approach.

Normalize values

When creating a model with multiple features, the values of each feature should cover roughly the same range. For example, if one feature's range spans 500 to 100,000 and another feature's range spans 2 to 12, then the model will be difficult or impossible to train. Therefore, you should [normalize](https://developers.google.com/machine-learning/glossary/" \l "normalization" \t "_blank) features in a multi-feature model.

The following code cell normalizes datasets by converting each raw value (including the label) to its Z-score. A **Z-score** is the number of standard deviations from the mean for a particular raw value. For example, consider a feature having the following characteristics:

The mean is 60.

The standard deviation is 10.

The raw value 75 would have a Z-score of +1.5:

  Z-score = (75 - 60) / 10 = +1.5

The raw value 38 would have a Z-score of -2.2:

  Z-score = (38 - 60) / 10 = -2.2

# Calculate the Z-scores of each column in the training set and

# write those Z-scores into a new pandas DataFrame named train\_df\_norm.

train\_df\_mean = train\_df.mean()

train\_df\_std = train\_df.std()

train\_df\_norm = (train\_df - train\_df\_mean)/train\_df\_std

# Examine some of the values of the normalized training set. Notice that most

# Z-scores fall between -2 and +2.

train\_df\_norm.head()

# Calculate the Z-scores of each column in the test set and

# write those Z-scores into a new pandas DataFrame named test\_df\_norm.

test\_df\_norm = (test\_df - train\_df\_mean) / train\_df\_std

# Note that we transform the test data with the values calculated from the training set,

# as you should always transform your datasets with exactly the same values.

Task 1: Create a binary label

In classification problems, the label for every example must be either 0 or 1. Unfortunately, the natural label in the California Housing Dataset, median\_house\_value, contains floating-point values like 80,100 or 85,700 rather than 0s and 1s, while the normalized version of median\_house\_values contains floating-point values primarily between -3 and +3.

Your task is to create a new column named median\_house\_value\_is\_high in both the training set and the test set . If the median\_house\_value is higher than a certain arbitrary value (defined by threshold), then set median\_house\_value\_is\_high to 1. Otherwise, set median\_house\_value\_is\_high to 0.

**Hint:** The cells in the median\_house\_value\_is\_high column must each hold 1 and 0, not True and False. To convert True and False to 1 and 0, call the pandas DataFrame function astype(float).

#@title Double-click for possible solutions.

# We arbitrarily set the threshold to 265,000, which is

# the 75th percentile for median house values. Every neighborhood

# with a median house price above 265,000 will be labeled 1,

# and all other neighborhoods will be labeled 0.

threshold = 265000

train\_df\_norm["median\_house\_value\_is\_high"] = (train\_df["median\_house\_value"] > threshold).astype(float)

test\_df\_norm["median\_house\_value\_is\_high"] = (test\_df["median\_house\_value"] > threshold).astype(float)

train\_df\_norm["median\_house\_value\_is\_high"].head(8000)

# Alternatively, instead of picking the threshold

# based on raw house values, you can work with Z-scores.

# For example, the following possible solution uses a Z-score

# of +1.0 as the threshold, meaning that no more

# than 16% of the values in median\_house\_value\_is\_high

# will be labeled 1.

# threshold\_in\_Z = 1.0

# train\_df\_norm["median\_house\_value\_is\_high"] = (train\_df\_norm["median\_house\_value"] > threshold\_in\_Z).astype(float)

# test\_df\_norm["median\_house\_value\_is\_high"] = (test\_df\_norm["median\_house\_value"] > threshold\_in\_Z).astype(float)

Represent features as input layers

This code cell specifies the features, median\_income and total\_rooms, that you'll ultimately train the model on. These [Input](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.tensorflow.org%2Fapi_docs%2Fpython%2Ftf%2Fkeras%2FInput" \t "_blank) objects are instantiated as Keras tensors.

inputs = {

# Features used to train the model on.

'median\_income': tf.keras.Input(shape=(1,)),

'total\_rooms': tf.keras.Input(shape=(1,))

}

Define functions that build and train a model

The following code cell defines two functions:

create\_model(inputs, learning\_rate, METRICS), which defines the model's topography.

train\_model(model, dataset, epochs, label\_name, batch\_size, shuffle), uses input features and labels to train the model.

Prior exercises used [ReLU](https://developers.google.com/machine-learning/glossary" \l "ReLU" \t "_blank) as the [activation function](https://developers.google.com/machine-learning/glossary" \l "activation-function" \t "_blank). By contrast, this exercise uses [sigmoid](https://developers.google.com/machine-learning/glossary" \l "sigmoid-function" \t "_blank) as the activation function.

Define the functions that create and train a model.

#@title Define the functions that create and train a model.

def create\_model(my\_inputs, my\_learning\_rate, METRICS):

# Use a Concatenate layer to concatenate the input layers into a single tensor.

# as input for the Dense layer. Ex: [input\_1[0][0], input\_2[0][0]]

concatenated\_inputs = tf.keras.layers.Concatenate()(my\_inputs.values())

dense = layers.Dense(units=1, name='dense\_layer', activation=tf.sigmoid)

dense\_output = dense(concatenated\_inputs)

"""Create and compile a simple classification model."""

my\_outputs = {

'dense': dense\_output,

}

model = tf.keras.Model(inputs=my\_inputs, outputs=my\_outputs)

# Call the compile method to construct the layers into a model that

# TensorFlow can execute. Notice that we're using a different loss

# function for classification than for regression.

model.compile(optimizer=tf.keras.optimizers.experimental.RMSprop(learning\_rate=my\_learning\_rate),

loss=tf.keras.losses.BinaryCrossentropy(),

metrics=METRICS)

return model

def train\_model(model, dataset, epochs, label\_name,

batch\_size=None, shuffle=True):

"""Feed a dataset into the model in order to train it."""

# The x parameter of tf.keras.Model.fit can be a list of arrays, where

# each array contains the data for one feature. Here, we're passing

# every column in the dataset. Note that the feature\_layer will filter

# away most of those columns, leaving only the desired columns and their

# representations as features.

features = {name:np.array(value) for name, value in dataset.items()}

label = np.array(features.pop(label\_name))

history = model.fit(x=features, y=label, batch\_size=batch\_size,

epochs=epochs, shuffle=shuffle)

# The list of epochs is stored separately from the rest of history.

epochs = history.epoch

# Isolate the classification metric for each epoch.

hist = pd.DataFrame(history.history)

return epochs, hist

print("Defined the create\_model and train\_model functions.")

Define a plotting function

The following [matplotlib](https://developers.google.com/machine-learning/glossary/" \l "matplotlib" \t "_blank) function plots one or more curves, showing how various classification metrics change with each epoch.

Define the plotting function.

#@title Define the plotting function.

def plot\_curve(epochs, hist, list\_of\_metrics):

"""Plot a curve of one or more classification metrics vs. epoch."""

# list\_of\_metrics should be one of the names shown in:

# https://www.tensorflow.org/tutorials/structured\_data/imbalanced\_data#define\_the\_model\_and\_metrics

plt.figure()

plt.xlabel("Epoch")

plt.ylabel("Value")

for m in list\_of\_metrics:

x = hist[m]

plt.plot(epochs[1:], x[1:], label=m)

plt.legend()

print("Defined the plot\_curve function.")

Invoke the creating, training, and plotting functions

The following code cell calls specify the hyperparameters, and then invokes the functions to create and train the model, and then to plot the results.

# The following variables are the hyperparameters.

learning\_rate = 0.001

epochs = 20

batch\_size = 100

label\_name = "median\_house\_value\_is\_high"

classification\_threshold = 0.35

# Establish the metrics the model will measure.

METRICS = [

tf.keras.metrics.BinaryAccuracy(name='accuracy',

threshold=classification\_threshold),

]

# Establish the model's topography.

my\_model = create\_model(inputs, learning\_rate, METRICS)

# To view a PNG of this model's layers, uncomment the call to

# `tf.keras.utils.plot\_model` below. After running this code cell, click

# the file folder on the left, then the `my\_classification\_model.png` file.

# tf.keras.utils.plot\_model(my\_model, "my\_classification\_model.png")

# Train the model on the training set.

epochs, hist = train\_model(my\_model, train\_df\_norm, epochs,

label\_name, batch\_size)

# Plot a graph of the metric(s) vs. epochs.

list\_of\_metrics\_to\_plot = ['accuracy']

plot\_curve(epochs, hist, list\_of\_metrics\_to\_plot)

Evaluate the model against the test set

At the end of model training, you ended up with a certain accuracy against the training set. Invoke the following code cell to determine your model's accuracy against the test set.

features = {name:np.array(value) for name, value in test\_df\_norm.items()}

label = np.array(features.pop(label\_name))

my\_model.evaluate(x = features, y = label, batch\_size=batch\_size)

Task 2: How accurate is your model really?

Is your model valuable?

#@title Double-click for a possible answer to Task 2.

# A perfect model would make 100% accurate predictions.

# Our model makes 80% accurate predictions. 80% sounds

# good, but note that a model that always guesses

# "median\_house\_value\_is\_high is False" would be 75%

# accurate.

Task 3: Add precision and recall as metrics

Relying solely on accuracy, particularly for a class-imbalanced data set (like ours), can be a poor way to judge a classification model. Modify the code in the following code cell to enable the model to measure not only accuracy but also precision and recall. We have added accuracy and precision; your task is to add recall. See the [TensorFlow Reference](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fwww.tensorflow.org%2Fapi_docs%2Fpython%2Ftf%2Fkeras%2Fmetrics%2FRecall" \t "_blank) for details.

#@title Double-click to view the solution for Task 3.

# The following variables are the hyperparameters.

learning\_rate = 0.001

epochs = 20

batch\_size = 100

classification\_threshold = 0.35

label\_name = "median\_house\_value\_is\_high"

# Here is the updated definition of METRICS:

METRICS = [

tf.keras.metrics.BinaryAccuracy(name='accuracy',

threshold=classification\_threshold),

tf.keras.metrics.Precision(thresholds=classification\_threshold,

name='precision'

),

tf.keras.metrics.Recall(thresholds=classification\_threshold,

name="recall"),

]

# Establish the model's topography.

my\_model = create\_model(inputs, learning\_rate, METRICS)

# Train the model on the training set.

epochs, hist = train\_model(my\_model, train\_df\_norm, epochs,

label\_name, batch\_size)

# Plot metrics vs. epochs

list\_of\_metrics\_to\_plot = ['accuracy', "precision", "recall"]

plot\_curve(epochs, hist, list\_of\_metrics\_to\_plot)

# The new graphs suggest that precision and recall are

# somewhat in conflict. That is, improvements to one of

# those metrics may hurt the other metric.

Task 4: Experiment with the classification threshold (if time permits)

Experiment with different values for classification\_threshold in the code cell within "Invoke the creating, training, and plotting functions." What value of classification\_threshold produces the highest accuracy?

#@title Double-click to view the solution for Task 4.

# The following variables are the hyperparameters.

learning\_rate = 0.001

epochs = 20

batch\_size = 100

classification\_threshold = 0.52

label\_name = "median\_house\_value\_is\_high"

# Here is the updated definition of METRICS:

METRICS = [

tf.keras.metrics.BinaryAccuracy(name='accuracy',

threshold=classification\_threshold),

tf.keras.metrics.Precision(thresholds=classification\_threshold,

name='precision'

),

tf.keras.metrics.Recall(thresholds=classification\_threshold,

name="recall"),

]

# Establish the model's topography.

my\_model = create\_model(inputs, learning\_rate, METRICS)

# Train the model on the training set.

epochs, hist = train\_model(my\_model, train\_df\_norm, epochs,

label\_name, batch\_size)

# Plot metrics vs. epochs

list\_of\_metrics\_to\_plot = ['accuracy', "precision", "recall"]

plot\_curve(epochs, hist, list\_of\_metrics\_to\_plot)

# A `classification\_threshold` of slightly over 0.5

# appears to produce the highest accuracy (about 83%).

# Raising the `classification\_threshold` to 0.9 drops

# accuracy by about 5%. Lowering the

# `classification\_threshold` to 0.3 drops accuracy by

# about 3%.

Task 5: Summarize model performance (if time permits)

If time permits, add one more metric that attempts to summarize the model's overall performance.

#@title Double-click to view the solution for Task 5.

# The following variables are the hyperparameters.

learning\_rate = 0.001

epochs = 20

batch\_size = 100

label\_name = "median\_house\_value\_is\_high"

# AUC is a reasonable "summary" metric for

# classification models.

# Here is the updated definition of METRICS to

# measure AUC:

METRICS = [

tf.keras.metrics.AUC(num\_thresholds=100, name='auc'),

]

# Establish the model's topography.

my\_model = create\_model(inputs, learning\_rate, METRICS)

# Train the model on the training set.

epochs, hist = train\_model(my\_model, train\_df\_norm, epochs,

label\_name, batch\_size)

# Plot metrics vs. epochs

list\_of\_metrics\_to\_plot = ['auc']

plot\_curve(epochs, hist, list\_of\_metrics\_to\_plot)

Regularização de esparsidade

O foco deste módulo é os requisitos especiais para modelos aprendidos em vetores de recursos que têm muitas dimensões.

Regularização de esparsidade: regularização L1

Os vetores esparsos geralmente contêm muitas dimensões. A criação de um [cruzamento de atributos](https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture?hl=pt-br) resulta em ainda mais dimensões. Considerando esses vetores de atributos de alta dimensão, o tamanho do modelo pode se tornar enorme e exigir grandes quantidades de RAM.

Em um vetor esparso de alta dimensão, é recomendável incentivar os pesos a cairem exatamente para 0 sempre que possível. Um peso de exatamente 0 basicamente remove o recurso correspondente do modelo. zerar recursos vai economizar RAM e pode reduzir ruídos no modelo.

Por exemplo, considere um conjunto de dados de cobertura que abrange não apenas a Califórnia, mas todo o mundo. A latitude globalizada por bucket no nível de minuto (60 minutos por grau) fornece cerca de 10.000 dimensões em uma codificação esparsa. A longitude global no nível do minuto fornece cerca de 20.000 dimensões. Um cruzamento desses dois atributos resultaria em cerca de 200.000.000 dimensões. Muitas dessas 200.000.000 dimensões representam áreas de residência tão limitada (por exemplo, o meio do oceano) que seria difícil usar esses dados para generalizar de maneira eficaz. Seria tolo pagar o custo de RAM para armazenar essas dimensões desnecessárias. Portanto, é bom incentivar os pesos das dimensões sem sentido a cair exatamente para 0, o que nos permite evitar pagar pelo custo de armazenamento desses coeficientes do modelo no momento da inferência.

Talvez possamos codificar essa ideia no problema de otimização feito no momento do treinamento, adicionando um termo de regularização escolhido de maneira adequada.

A regularização de L2 realizaria essa tarefa? Infelizmente, não. A regularização de L2 incentiva os pesos a serem pequenos, mas não os força a exatamente 0,0.

Uma ideia alternativa seria tentar criar um termo de regularização que penalize a contagem de valores de coeficiente diferentes de zero em um modelo. O aumento dessa contagem só será justificado se houver um ganho suficiente na capacidade do modelo de ajustar os dados. Embora essa abordagem baseada em contagem seja intuitivamente atraente, ela transformaria nosso problema de otimização de convex em um problema de otimização não convexa. Portanto, essa ideia, conhecida como regularização L0, não é uma prática que possa ser usada na prática.

No entanto, há um termo de regularização chamado regularização L1 que serve como uma aproximação de L0, mas tem a vantagem de ser convexo e, portanto, eficiente em computação. Então, podemos usar a regularização de L1 para incentivar muitos dos coeficientes não informativos em nosso modelo a serem exatamente 0 e, assim, colher economia de RAM no inferência.

## regularização L1 x L2.

P2 e L1 penalizam os pesos de maneira diferente:

* L2 penaliza weight2.
* L1 penaliza |weight|.

Consequentemente, L2 e L1 têm derivados diferentes:

* A derivada de L2 é 2 \* weight.
* O derivado de L1 é k (uma constante com valor independente do peso).

Pense no derivado de L2 como uma força que remove x% do peso todas as vezes. Como [Zeno](https://wikipedia.org/wiki/Zeno%27s_paradoxes" \l "Dichotomy_paradox) sabe, mesmo se você remover x por cento de um número *bilhões de vezes*, o número reduzido nunca atingirá zero. Zeno não estava familiarizado com limitações de precisão de ponto flutuante, o que poderia produzir exatamente zero. De qualquer forma, L2 normalmente não direciona pesos para zero.

Pense no derivado de L1 como uma força que subtrai alguma constante do peso todas as vezes. No entanto, graças aos valores absolutos, L1 tem uma descontinuidade em 0, o que faz com que os resultados de subtração que cruzam 0 sejam zerados. Por exemplo, se a subtração forçaria um peso de +0,1 a -0,2, L1 definirá o peso como exatamente 0. Eureka, L1 zerou o peso.

A regularização de L1 (penalização do valor absoluto de todos os pesos) acaba sendo bastante eficiente para modelos grandes.

Essa descrição é verdadeira para um modelo unidimensional.

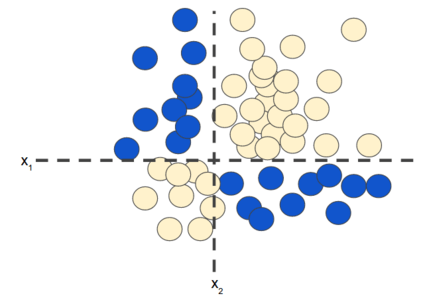
Clique no botão "Reproduzir" (play\_arrow) abaixo para comparar o efeito da regularização de L1 e L2 em uma rede de pesos.

Redes neurais

Redes neurais são uma versão mais sofisticada dos cruzamentos de atributos. Basicamente, as redes neurais aprendem os cruzamentos de atributos apropriados para você.

Redes neurais: estrutura

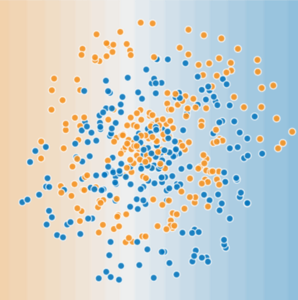
Como você se lembra da [unidade de cruzamentos de atributos](https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture?hl=pt-br), o problema de classificação a seguir não é linear:



**Figura 1. problema de classificação não linear.**

"Não linear" significa que não é possível prever # com precisão um rótulo com um modelo do formulário b + w1x1 + w2x2 Em outras palavras, a "superfície de decisão" não é uma linha. Anteriormente, observamos os [cruzamentos de atributos](https://developers.google.com/machine-learning/crash-course/feature-crosses/video-lecture?hl=pt-br) como uma possível abordagem para modelar problemas não lineares.

Agora considere o seguinte conjunto de dados:



**Figura 2. um problema de classificação não linear mais difícil.**

O conjunto de dados mostrado na Figura 2 não pode ser resolvido com um modelo linear.

Para ver como as redes neurais podem ajudar com problemas não lineares, vamos começar representando um modelo linear como um gráfico:

**Figura 3. Modelo linear como gráfico.**

Cada círculo azul representa um atributo de entrada, e o círculo verde representa a soma ponderada das entradas.

Como alterar esse modelo para melhorar a capacidade dele de lidar com problemas não lineares?

## Camadas escondidas

No modelo representado pelo gráfico a seguir, adicionamos uma camada escondida de valores intermediários. Cada nó amarelo na camada oculta é uma soma ponderada dos valores de nó de entrada azul. A saída é uma soma ponderada dos nós amarelos.

**Figura 4. Gráfico do modelo de duas camadas.**

Este modelo é linear? Sim, a saída ainda é uma combinação linear das entradas dela.

No modelo representado pelo gráfico a seguir, adicionamos uma segunda camada escondida de somas ponderadas.

**Figura 5. Gráfico do modelo de três camadas.**

Este modelo ainda é linear? Sim. Ao expressar a saída como uma função da entrada e simplificar, você recebe apenas outra soma ponderada das entradas. Essa soma não modela o problema não linear na Figura 2.

## Funções de ativação

Para modelar um problema não linear, podemos introduzir uma não linearidade diretamente. Podemos encadear cada nó de camada escondida por meio de uma função não linear.

No modelo representado pelo gráfico a seguir, o valor de cada nó na camada escondida 1 é transformado por uma função não linear antes de ser transmitido para as somas ponderadas da próxima camada. Ela é chamada de função de ativação.

**Figura 6. Gráfico do modelo de três camadas com função de ativação.**

Agora que adicionamos uma função de ativação, a adição de camadas tem mais impacto. O empilhamento de não linearidades em não linearidades nos permite modelar relações muito complicadas entre as entradas e as saídas previstas. Em resumo, cada camada está aprendendo de maneira eficaz uma função mais complexa e de alto nível sobre as entradas brutas. Para saber mais sobre como isso funciona, consulte a [excelente postagem do blog Chris Olah'](http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/) (em inglês).

### Funções de ativação comuns

A função de ativação **sigmoid** a seguir converte a soma ponderada em um valor entre 0 e 1.

F(x) = 1 / 1 + e ˆ−x

Veja um gráfico:

**Figura 7. Função de ativação sigmoide.**

A função de ativação de **unidade linear retificada** a seguir (ou **ReLU**, na sigla em inglês) geralmente funciona um pouco melhor do que uma função suave, como o sigmoide, além de ser significativamente mais fácil de calcular.

F(x) = max(0, x)

A superioridade da ReLU é baseada em descobertas empíricas, provavelmente impulsionada pela ReLU, tendo um intervalo de capacidade de resposta mais útil. A capacidade de resposta de um sigmoide fica relativamente rápida em ambos os lados.

**Figura 8. função de ativação ReLU.**

Na verdade, qualquer função matemática pode servir como uma função de ativação. Suponha que (o) represente nossa função de ativação (Relu, Sigmoid ou o que for). Consequentemente, o valor de um nó na rede é fornecido pela seguinte fórmula:

o(w ⋅ x + b)

O TensorFlow oferece suporte predefinido para várias funções de ativação. Você encontra essas funções de ativação na [lista de wrappers de operações primitivas de rede neural](https://www.tensorflow.org/api_docs/python/tf/nn?hl=pt-br) do TensorFlow. Mesmo assim, recomendamos começar com a ReLU.

## Resumo

Agora, nosso modelo tem todos os componentes padrão do que as pessoas costumam significar quando dizem "rede neural":

* Um conjunto de nós, análogo a neurônios, organizados em camadas.
* Um conjunto de pesos que representam as conexões entre cada camada de rede neural e a camada abaixo dela. A camada abaixo pode ser outra camada de rede neural ou algum outro tipo de camada.
* Um conjunto de vieses, um para cada nó.
* Uma função de ativação que transforma a saída de cada nó em uma camada. As camadas podem ter funções de ativação diferentes.

Advertência: as redes neurais nem sempre são melhores do que os cruzamentos de atributos, mas as redes neurais oferecem uma alternativa flexível que funciona bem em muitos casos.

Redes neurais: exercícios

## Uma primeira rede neural

Neste exercício, vamos treinar nossa primeira rede neural. As redes neurais vão nos permitir aprender modelos não lineares sem usar cruzamentos de atributos explícitos.

**Tarefa 1:** o modelo fornecido combina nossos dois recursos de entrada em um único neurônio. Esse modelo vai aprender alguma não linearidade? Execute-o para confirmar sua chute.

**Tarefa 2**: tente aumentar o número de neurônios na camada escondida de 1 para 2 e também mude de uma ativação linear para uma ativação não linear, como ReLU. Você consegue criar um modelo para aprender não linearidades? Ele consegue modelar os dados de maneira eficaz?

**Tarefa 3:** tente aumentar o número de neurônios na camada oculta de 2 para 3, usando uma ativação não linear, como a ReLU. Ele pode modelar os dados de maneira eficaz? Como a qualidade dos modelos varia de uma execução para outra?

**Tarefa 4:** continue o experimento adicionando ou removendo camadas e neurônios ocultos por camada. Também é possível alterar as taxas de aprendizado, a regularização e outras configurações de aprendizado. Qual é o *menor* número de neurônios e camadas que podem ser usados, resultando em uma perda de teste de 0,177 ou menor?

O aumento do tamanho do modelo melhora a adequação ou a convergência? Isso muda a frequência com que ele converge para um bom modelo? Por exemplo, teste a seguinte arquitetura:

* Primeira camada escondida com três neurônios.
* Segunda camada escondida com três neurônios.
* Terceira camada escondida com dois neurônios.

## Inicialização da rede neural

Esse exercício usa os dados XOR novamente, mas analisa a capacidade de repetição do treinamento de redes neurais e a importância da inicialização.

**Tarefa 1**:execute o modelo da mesma maneira quatro ou cinco vezes. Antes de cada teste, clique no botão **Redefinir a rede** para uma nova inicialização aleatória. O botão **Redefinir a rede** é a seta de redefinição circular à esquerda do botão "Reproduzir". Execute cada teste por pelo menos 500 passos para garantir a convergência. Em que formato cada saída de modelo converge? O que isso diz sobre o papel da inicialização em otimização não convexa?

**Tarefa 2:** inclua uma camada e alguns nós extras para tornar o modelo um pouco mais complexo. Repita os testes da Tarefa 1. Isso adiciona alguma estabilidade aos resultados?

## Espiral da rede neural

Este conjunto de dados é uma espiral com ruído. Obviamente, um modelo linear falhará aqui, mas até mesmo os cruzamentos de atributos definidos manualmente podem ser difíceis de construir.

**Tarefa 1**:treinar o melhor modelo que puder, usando apenas X1 e X2. Você pode adicionar ou remover camadas e neurônios, mudar configurações de aprendizado, como taxa de aprendizado, taxa de regularização e tamanho do lote. Qual é a melhor perda de teste que você pode ter? Sua superfície de saída do modelo é suave?

**Tarefa 2:** mesmo com redes neurais, é necessário algum tipo de engenharia de atributos para alcançar a melhor performance. Adicione outros recursos do produto ou outras transformações, como sin(X1) e sin(X2). Você consegue um modelo melhor? A superfície de saída do modelo é mais suave?

Redes neurais: exercício de programação

O exercício a seguir permite desenvolver e treinar uma rede neural:

 [Introdução ao Neural Networks](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/intro_to_neural_nets.ipynb?utm_source=mlcc&%3Butm_campaign=colab-external&%3Butm_medium=referral&%3Butm_content=intro_to_nn_tf2-colab&%3Bhl=+l10placeholder&hl=pt-br) do Colab.

Os exercícios de programação são executados diretamente no navegador, sem necessidade de configuração, usando a plataforma [Colaboratory](https://colab.research.google.com/?hl=pt-br) (em inglês). O Colaboratory é compatível com a maioria dos principais navegadores e é testado mais detalhadamente em versões do Chrome para computador e Chrome. Se você preferir fazer o download e executar os exercícios off-line, consulte [estas instruções](https://developers.google.com/machine-learning/crash-course/running-exercises-locally?hl=pt-br) para configurar um ambiente local.

**Introduction to Neural Nets**

This Colab builds a deep neural network to perform more sophisticated linear regression than the earlier Colabs.

**Learning Objectives:**

After doing this Colab, you'll know how to do the following:

Create a simple deep neural network.

Tune the hyperparameters for a simple deep neural network.

**The Dataset**

Like several of the previous Colabs, this Colab uses the [California Housing Dataset](https://developers.google.com/machine-learning/crash-course/california-housing-data-description).

**Import relevant modules**

The following hidden code cell imports the necessary code to run the code in the rest of this Colaboratory.

**Import relevant modules**

#@title Import relevant modules

import numpy as np

import pandas as pd

import tensorflow as tf

from matplotlib import pyplot as plt

import seaborn as sns

# The following lines adjust the granularity of reporting.

pd.options.display.max\_rows = 10

pd.options.display.float\_format = "{:.1f}".format

print("Imported modules.")

**Load the dataset**

Like most of the previous Colab exercises, this exercise uses the California Housing Dataset. The following code cell loads the separate .csv files and creates the following two pandas DataFrames:

train\_df, which contains the training set

test\_df, which contains the test set

train\_df = pd.read\_csv("https://download.mlcc.google.com/mledu-datasets/california\_housing\_train.csv")

train\_df = train\_df.reindex(np.random.permutation(train\_df.index)) # shuffle the examples

test\_df = pd.read\_csv("https://download.mlcc.google.com/mledu-datasets/california\_housing\_test.csv")

**Represent data**

The following code cell creates preprocessing layers outputting three features:

latitude X longitude (a feature cross)

median\_income

population

This code cell specifies the features that you'll ultimately train the model on and how each of those features will be represented. The transformations (collected in prepocessing\_layers) don't actually get applied until you pass a DataFrame to it, which will happen when we train the model.

We'll use preprocessing\_layers for both our linear regression model and our neural network model.

(The [keras.FeatureSpace](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fkeras.io%2Fexamples%2Fstructured_data%2Fstructured_data_classification_with_feature_space) utility offers an alternative to building individual Keras preprocessing layers -- give it a try, if you're feeling adventurous!)

# Keras Input tensors of float values.

inputs = {

'latitude':

tf.keras.layers.Input(shape=(1,), dtype=tf.float32,

name='latitude'),

'longitude':

tf.keras.layers.Input(shape=(1,), dtype=tf.float32,

name='longitude'),

'median\_income':

tf.keras.layers.Input(shape=(1,), dtype=tf.float32,

name='median\_income'),

'population':

tf.keras.layers.Input(shape=(1,), dtype=tf.float32,

name='population')

}

# Create a Normalization layer to normalize the median\_income data.

median\_income = tf.keras.layers.Normalization(

name='normalization\_median\_income',

axis=None)

median\_income.adapt(train\_df['median\_income'])

median\_income = median\_income(inputs.get('median\_income'))

# Create a Normalization layer to normalize the population data.

population = tf.keras.layers.Normalization(

name='normalization\_population',

axis=None)

population.adapt(train\_df['population'])

population = population(inputs.get('population'))

# Create a list of numbers representing the bucket boundaries for latitude.

# Because we're using a Normalization layer, values for latitude and longitude

# will be in the range of approximately -3 to 3 (representing the Z score).

# We'll create 20 buckets, which requires 21 bucket boundaries (hence, 20+1).

latitude\_boundaries = np.linspace(-3, 3, 20+1)

# Create a Normalization layer to normalize the latitude data.

latitude = tf.keras.layers.Normalization(

name='normalization\_latitude',

axis=None)

latitude.adapt(train\_df['latitude'])

latitude = latitude(inputs.get('latitude'))

# Create a Discretization layer to separate the latitude data into buckets.

latitude = tf.keras.layers.Discretization(

bin\_boundaries=latitude\_boundaries,

name='discretization\_latitude')(latitude)

# Create a list of numbers representing the bucket boundaries for longitude.

longitude\_boundaries = np.linspace(-3, 3, 20+1)

# Create a Normalization layer to normalize the longitude data.

longitude = tf.keras.layers.Normalization(

name='normalization\_longitude',

axis=None)

longitude.adapt(train\_df['longitude'])

longitude = longitude(inputs.get('longitude'))

# Create a Discretization layer to separate the longitude data into buckets.

longitude = tf.keras.layers.Discretization(

bin\_boundaries=longitude\_boundaries,

name='discretization\_longitude')(longitude)

# Cross the latitude and longitude features into a single one-hot vector.

feature\_cross = tf.keras.layers.HashedCrossing(

# num\_bins can be adjusted: Higher values improve accuracy, lower values

# improve performance.

num\_bins=len(latitude\_boundaries) \* len(longitude\_boundaries),

output\_mode='one\_hot',

name='cross\_latitude\_longitude')([latitude, longitude])

# Concatenate our inputs into a single tensor.

preprocessing\_layers = tf.keras.layers.Concatenate()(

[feature\_cross, median\_income, population])

print("Preprocessing layers defined.")

**Build a linear regression model as a baseline**

Before creating a deep neural net, find a [baseline](https://developers.google.com/machine-learning/glossary/#baseline) loss by running a simple linear regression model that uses the preprocessing layers you just created.

**Define the plotting function.**

#@title Define the plotting function.

def plot\_the\_loss\_curve(epochs, mse\_training, mse\_validation):

"""Plot a curve of loss vs. epoch."""

plt.figure()

plt.xlabel("Epoch")

plt.ylabel("Mean Squared Error")

plt.plot(epochs, mse\_training, label="Training Loss")

plt.plot(epochs, mse\_validation, label="Validation Loss")

# mse\_training is a pandas Series, so convert it to a list first.

merged\_mse\_lists = mse\_training.tolist() + mse\_validation

highest\_loss = max(merged\_mse\_lists)

lowest\_loss = min(merged\_mse\_lists)

top\_of\_y\_axis = highest\_loss \* 1.03

bottom\_of\_y\_axis = lowest\_loss \* 0.97

plt.ylim([bottom\_of\_y\_axis, top\_of\_y\_axis])

plt.legend()

plt.show()

print("Defined the plot\_the\_loss\_curve function.")

**Define functions to create and train a linear regression model**

#@title Define functions to create and train a linear regression model

def create\_model(my\_inputs, my\_outputs, my\_learning\_rate):

"""Create and compile a simple linear regression model."""

model = tf.keras.Model(inputs=my\_inputs, outputs=my\_outputs)

# Construct the layers into a model that TensorFlow can execute.

model.compile(optimizer=tf.keras.optimizers.Adam(

learning\_rate=my\_learning\_rate),

loss="mean\_squared\_error",

metrics=[tf.keras.metrics.MeanSquaredError()])

return model

def train\_model(model, dataset, epochs, batch\_size, label\_name, validation\_split=0.1):

"""Feed a dataset into the model in order to train it."""

# Split the dataset into features and label.

features = {name:np.array(value) for name, value in dataset.items()}

label = train\_median\_house\_value\_normalized(

np.array(features.pop(label\_name)))

history = model.fit(x=features, y=label, batch\_size=batch\_size,

epochs=epochs, shuffle=True, validation\_split=validation\_split)

# Get details that will be useful for plotting the loss curve.

epochs = history.epoch

hist = pd.DataFrame(history.history)

mse = hist["mean\_squared\_error"]

return epochs, mse, history.history

print("Defined the create\_model and train\_model functions.")

**Define normalized label columns**

#@title Define normalized label columns

# Create Normalization layers to normalize the median\_house\_value data.

# Because median\_house\_value is our label (i.e., the target value we're

# predicting), these layers won't be added to our model.

train\_median\_house\_value\_normalized = tf.keras.layers.Normalization(axis=None)

train\_median\_house\_value\_normalized.adapt(

np.array(train\_df['median\_house\_value']))

test\_median\_house\_value\_normalized = tf.keras.layers.Normalization(axis=None)

test\_median\_house\_value\_normalized.adapt(

np.array(test\_df['median\_house\_value']))

**Define linear regression model outputs**

#@title Define linear regression model outputs

def get\_outputs\_linear\_regression():

# Create the Dense output layer.

dense\_output = tf.keras.layers.Dense(units=1,

name='dense\_output')(preprocessing\_layers)

# Define an output dictionary we'll send to the model constructor.

outputs = {

'dense\_output': dense\_output

}

return outputs

Run the following code cell to invoke the functions defined in the preceding two code cells. (Ignore the warning messages.)

**Note:** Because we've scaled all the input data, **including the label**, the resulting loss values will be much less than models in previous Colabs (e.g., [Representation with a Feature Cross](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fcolab.sandbox.google.com%2Fgithub%2Fgoogle%2Feng-edu%2Fblob%2Fmain%2Fml%2Fcc%2Fexercises%2Frepresentation_with_a_feature_cross.ipynb)).

**Note:** Depending on the version of TensorFlow, running this cell might generate WARNING messages. Please ignore these warnings.

# The following variables are the hyperparameters.

learning\_rate = 0.01

epochs = 15

batch\_size = 1000

label\_name = "median\_house\_value"

# Split the original training set into a reduced training set and a

# validation set.

validation\_split = 0.2

outputs = get\_outputs\_linear\_regression()

# Establish the model's topography.

my\_model = create\_model(inputs, outputs, learning\_rate)

# Train the model on the normalized training set.

epochs, mse, history = train\_model(my\_model, train\_df, epochs, batch\_size,

label\_name, validation\_split)

plot\_the\_loss\_curve(epochs, mse, history["val\_mean\_squared\_error"])

test\_features = {name:np.array(value) for name, value in test\_df.items()}

test\_label = test\_median\_house\_value\_normalized(test\_features.pop(label\_name)) # isolate the label

print("\n Evaluate the linear regression model against the test set:")

my\_model.evaluate(x = test\_features, y = test\_label, batch\_size=batch\_size, return\_dict=True)

**Define a deep neural net model**

The get\_outputs\_dnn function defines the topography of the deep neural net (DNN), specifying the following:

The number of [layers](https://developers.google.com/machine-learning/glossary/#layer) in the deep neural net.

The number of [nodes](https://developers.google.com/machine-learning/glossary/#node) in each layer.

The get\_outputs\_dnn function also defines the [activation function](https://developers.google.com/machine-learning/glossary/#activation_function) of each layer.

The first Dense layer takes our previously defined preprocessing\_layers as input.

def get\_outputs\_dnn():

# Create a Dense layer with 20 nodes.

dense\_output = tf.keras.layers.Dense(units=20,

activation='relu',

name='hidden\_dense\_layer\_1')(preprocessing\_layers)

# Create a Dense layer with 12 nodes.

dense\_output = tf.keras.layers.Dense(units=12,

activation='relu',

name='hidden\_dense\_layer\_2')(dense\_output)

# Create the Dense output layer.

dense\_output = tf.keras.layers.Dense(units=1,

name='dense\_output')(dense\_output)

# Define an output dictionary we'll send to the model constructor.

outputs = {

'dense\_output': dense\_output

}

return outputs

**Call the functions to build and train a deep neural net**

Okay, it is time to actually train the deep neural net. If time permits, experiment with the three hyperparameters to see if you can reduce the loss against the test set.

# The following variables are the hyperparameters.

learning\_rate = 0.01

epochs = 20

batch\_size = 1000

# Specify the label

label\_name = "median\_house\_value"

# Split the original training set into a reduced training set and a

# validation set.

validation\_split = 0.2

dnn\_outputs = get\_outputs\_dnn()

# Establish the model's topography.

my\_model = create\_model(

inputs,

dnn\_outputs,

learning\_rate)

# Train the model on the normalized training set. We're passing the entire

# normalized training set, but the model will only use the features

# defined in our inputs.

epochs, mse, history = train\_model(my\_model, train\_df, epochs,

batch\_size, label\_name, validation\_split)

plot\_the\_loss\_curve(epochs, mse, history["val\_mean\_squared\_error"])

# After building a model against the training set, test that model

# against the test set.

test\_features = {name:np.array(value) for name, value in test\_df.items()}

test\_label = test\_median\_house\_value\_normalized(np.array(test\_features.pop(label\_name))) # isolate the label

print("\n Evaluate the new model against the test set:")

my\_model.evaluate(x = test\_features, y = test\_label, batch\_size=batch\_size, return\_dict=True)

**Task 1: Compare the two models**

How did the deep neural net perform against the baseline linear regression model?

#@title Double-click to view a possible answer

# Assuming that the linear model converged and

# the deep neural net model also converged, please

# compare the test set loss for each.

# In our experiments, the loss of the deep neural

# network model was consistently lower than

# that of the linear regression model, which

# suggests that the deep neural network model

# will make better predictions than the

# linear regression model.

**Task 2: Optimize the deep neural network's topography**

Experiment with the number of layers of the deep neural network and the number of nodes in each layer. Aim to achieve both of the following goals:

Lower the loss against the test set.

Minimize the overall number of nodes in the deep neural net.

The two goals may be in conflict.

#@title Double-click to view a possible answer

# Many answers are possible. We noticed the

# following trends:

# \* Two layers outperformed one layer, but

# three layers did not perform significantly

# better than two layers.

# In other words, two layers seemed best.

# \* Setting the topography as follows produced

# reasonably good results with relatively few

# nodes:

# \* 10 nodes in the first layer.

# \* 6 nodes in the second layer.

# As the number of nodes in each layer dropped

# below the preceding, test loss increased.

# However, depending on your application, hardware

# constraints, and the relative pain inflicted

# by a less accurate model, a smaller network

# (for example, 6 nodes in the first layer and

# 4 nodes in the second layer) might be

# acceptable.

Como treinar redes neurais

A **propagação** é o algoritmo de treinamento mais comum para redes neurais. Ele torna o gradiente descendente para redes neurais de várias camadas. O TensorFlow lida com a retropropagação automaticamente, portanto, você não precisa ter uma compreensão profunda do algoritmo. Para entender como ele funciona, consulte [esta explicação explicativa sobre o algoritmo de retropropagação](https://developers.google.com/machine-learning/crash-course/backprop-scroll?hl=pt-br). Ao rolar a explicação anterior, observe o seguinte:

* Como os dados fluem pelo grafo.
* Como a programação dinâmica nos impede de computar exponencialmente muitos caminhos pelo gráfico. Aqui, "programação dinâmica" significa apenas a gravação de resultados intermediários nas transmissões anteriores e posteriores.

Como treinar as redes neurais: práticas recomendadas

Esta seção explica casos de falha de retropropagação e a maneira mais comum de regularizar uma rede neural.

## **Casos de falha**

Há várias maneiras comuns de erro na retropropagação.

### Gradientes sumindo

Os gradientes das camadas inferiores (mais perto da entrada) podem se tornar muito pequenos. Em redes profundas, o cálculo desses gradientes pode envolver levar o produto de muitos termos pequenos.

Quando os gradientes desaparecem em direção a zero para as camadas inferiores, essas camadas são treinadas muito lentamente ou não são aplicadas.

A função de ativação ReLU pode ajudar a evitar a redução de gradientes.

### Gradientes explosivos

Se os pesos em uma rede forem muito grandes, os gradientes das camadas inferiores envolvem produtos de muitos termos grandes. Nesse caso, podem ter gradientes explosivos: gradientes muito grandes para convergir.

A normalização em lote pode ajudar a evitar gradientes explosivos, o que pode reduzir a taxa de aprendizado.

### Unidades ReLU inativas

Quando a soma ponderada de uma unidade de ReLU fica abaixo de 0, ela pode ficar travada. A saída é 0, o que não contribui para a saída da rede e os gradientes não podem mais fluir por ela durante a retropropagação. Com uma origem de gradientes cortadas, a entrada na ReLU pode nunca mudar o suficiente para trazer a soma ponderada acima de 0.

Reduzir a taxa de aprendizado pode ajudar a impedir que as unidades ReLU sejam eliminadas.

## Regularização de dropout

No entanto, outra forma de regularização chamada **Dropout** é útil para redes neurais. Ele funciona aleatoriamente &deixando as ativações de unidade em uma rede para uma única etapa de gradiente. Quanto mais você desistir, mais forte será a regularização:

* 0,0 = Sem regularização de dropout.
* 1.0 = Remover tudo. O modelo não aprende nada.
* Valores entre 0,0 e 1,0 = mais útil.

Redes neurais multiclasse

Anteriormente, você encontrou modelos de classificação binária que podem escolher entre uma *duas* opções possíveis, por exemplo:

* Um e-mail é spam ou não é spam.
* Um determinado tumor é maligno ou benigno.

Neste módulo, vamos analisar a classificação **multiclasse**, que pode escolher entre *várias* possibilidades. Exemplo:

* Este cachorro é um beagle, um baço ou um galgo?
* Esta flor é uma Íris siberiana, íris holandesa, íris de bandeira azul ou uma íris de barba anão?
* Este avião é um Boeing 747, Airbus 320, Boeing 777 ou Embraer 190?
* Esta é uma imagem de uma maçã, urso, doces, cachorros ou ovos?

Alguns problemas reais de várias classes envolvem a escolha de *milhões* de classes separadas. Por exemplo, considere um modelo de classificação multiclasse que pode identificar a imagem de quase tudo.

**Objetivos de aprendizado**

* Entender os problemas de classificação multiclasse, principalmente no Softmax.
* Desenvolva soluções Softmax no TensorFlow.

# Redes neurais multiclasse: um vs. todos

**Um x todos** fornece uma maneira de aproveitar a classificação binária. Dado um problema de classificação em N possíveis soluções, uma solução de um para todos consiste em N classificadores binários separados, um classificador binário para cada resultado possível. Durante o treinamento, o modelo executa uma sequência de classificadores binários, treinando cada um para responder a uma pergunta de classificação separada. Por exemplo, considerando a imagem de um cachorro, cinco reconhecedores diferentes podem ser treinados, quatro vendo a imagem como um exemplo negativo (não uma maçã, não um urso etc.) e um vendo a imagem como um exemplo positivo (um cachorro). Ou seja:

1. Esta imagem é uma maçã? Número
2. Esta imagem é um urso? Número
3. Esta imagem é doce? Número
4. Esta imagem é um cachorro? Sim.
5. Esta imagem é um ovo? Número

Essa abordagem é razoavelmente razoável quando o número total de classes é pequeno, mas torna-se cada vez mais ineficiente à medida que o número de classes aumenta.

É possível criar um modelo de um em relação a todos significativamente mais eficiente com uma rede neural profunda, em que cada nó de saída representa uma classe diferente. A figura a seguir sugere essa abordagem:

**Figura 1. Uma rede neural 1 em comparação a todas.**

Redes neurais multiclasse: Softmax

Não se esqueça de que a [regressão logística](https://developers.google.com/machine-learning/crash-course/logistic-regression?hl=pt-br) produz um decimal entre 0 e 1,0. Por exemplo, uma saída de regressão logística de 0,8 de um classificador de e-mail sugere uma chance de 80% de que um e-mail seja spam e 20% de chance de não ser spam. Claramente, a soma das probabilidades de um e-mail ser spam ou não é 1,0.

A **Softmax** estende essa ideia para um mundo multiclasse. Ou seja, o Softmax atribui probabilidades decimais a cada classe em um problema de várias classes. Essas probabilidades decimais precisam totalizar 1,0. Essa restrição adicional ajuda o treinamento a convergir mais rapidamente do que deveria.

Por exemplo, voltando à análise de imagens que vimos na Figura 1, o Softmax pode produzir as seguintes probabilidades de uma imagem pertencente a uma classe específica:

| **Aula** | **Probabilidade** |
| --- | --- |
| apple | 0,001 |
| urso | 0,04 |
| doces | 0,008 |
| dog | 0,95 |
| ovo | 0,001 |

O Softmax é implementado por uma camada de rede neural logo antes da camada de saída. A camada Softmax precisa ter o mesmo número de nós que a camada de saída.

**Figura 2. Camada Softmax em uma rede neural.**

Considere as seguintes variantes de Softmax:

* **Softmax completo** é o Softmax que estamos discutindo, ou seja, o Softmax calcula uma probabilidade para cada classe possível.
* **Amostragem candidata** significa que Softmax calcula uma probabilidade para todos os rótulos positivos, mas apenas para uma amostra aleatória de rótulos negativos. Por exemplo, se quisermos determinar se uma imagem de entrada é um beagle ou um galgo, não temos que fornecer probabilidades para cada exemplo que não seja "cachorro".

O Softmax completo é bastante barato quando o número de classes é pequeno, mas torna-se proibitivamente caro quando o número de classes aumenta. A amostragem de candidatos pode melhorar a eficiência em problemas com um grande número de classes.

## Um rótulo x vários rótulos

A Softmax presume que cada exemplo é um membro de exatamente uma classe. Alguns exemplos, no entanto, podem participar de várias classes simultaneamente. Para esses exemplos:

* Não é possível usar o Softmax.
* Confie em várias regressões logísticas.

Por exemplo, suponha que seus exemplos sejam imagens contendo exatamente um item, uma fruta. O Softmax pode determinar a probabilidade de esse item ser de pera, laranja, maçã e assim por diante. Se os exemplos são imagens que contêm vários tipos de coisas (arcos de diferentes tipos de fruta), você terá que usar várias regressões logísticas.

Redes neurais multiclasse: exercício de programação

No exercício a seguir, você vai explorar o Softmax no TensorFlow desenvolvendo um modelo que classificará dígitos escritos à mão:

 [Classificação multiclasse com exercício do Colab do MNIST](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/multi-class_classification_with_MNIST.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=multiclass_tf2-colab&hl=pt-br).

Multi-Class Classification

This Colab explores multi-class classification problems through the classic MNIST dataset.

Learning Objectives:

After doing this Colab, you'll know how to do the following:

Understand the classic MNIST problem.

Create a deep neural network that performs multi-class classification.

Tune the deep neural network.

This exercise introduces image classification with machine learning.

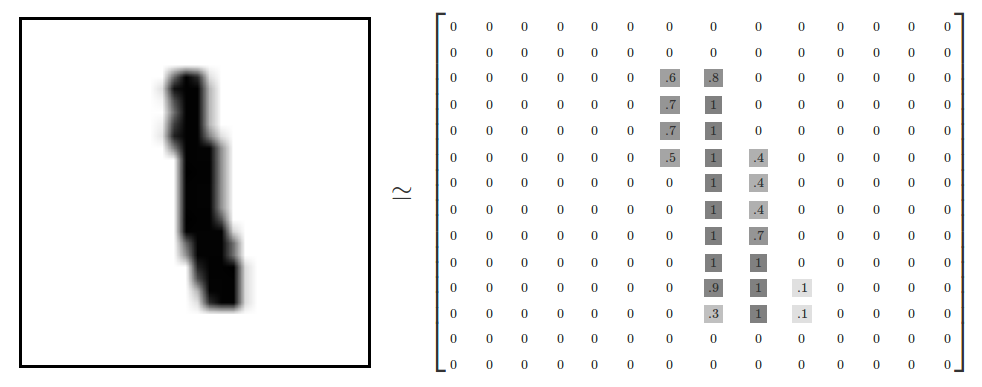
he Dataset

This MNIST dataset contains a lot of examples:

The MNIST training set contains 60,000 examples.

The MNIST test set contains 10,000 examples.

Each example contains a pixel map showing how a person wrote a digit. For example, the following images shows how a person wrote the digit 1 and how that digit might be represented in a 14x14 pixel map (after the input data is normalized).



Each example in the MNIST dataset consists of:

A label specified by a [rater](https://developers.google.com/machine-learning/glossary/#rater). Each label must be an integer from 0 to 9. For example, in the preceding image, the rater would almost certainly assign the label 1 to the example.

A 28x28 pixel map, where each pixel is an integer between 0 and 255. The pixel values are on a gray scale in which 0 represents white, 255 represents black, and values between 0 and 255 represent various shades of gray.

This is a multi-class classification problem with 10 output classes, one for each digit.

Import relevant modules

The following hidden code cell imports the necessary code to run the code in the rest of this Colaboratory.

Import relevant modules

#@title Import relevant modules

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import layers

from matplotlib import pyplot as plt

# The following lines adjust the granularity of reporting.

pd.options.display.max\_rows = 10

pd.options.display.float\_format = "{:.1f}".format

# The following line improves formatting when ouputting NumPy arrays.

np.set\_printoptions(linewidth = 200)

Load the dataset

tf.keras provides a set of convenience functions for loading well-known datasets. Each of these convenience functions does the following:

Loads both the training set and the test set.

Separates each set into features and labels.

The relevant convenience function for MNIST is called mnist.load\_data():

(x\_train, y\_train),(x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

Notice that mnist.load\_data() returned four separate values:

x\_train contains the training set's features.

y\_train contains the training set's labels.

x\_test contains the test set's features.

y\_test contains the test set's labels.

Note: The MNIST .csv training set is already shuffled.

View the dataset

The .csv file for the California Housing Dataset contains column names (for example, latitude, longitude, population). By contrast, the .csv file for MNIST does not contain column names. Instead of column names, you use ordinal numbers to access different subsets of the MNIST dataset. In fact, it is probably best to think of x\_train and x\_test as two-dimensional NumPy arrays:

# Output example #2917 of the training set.

x\_train[2917]

Alternatively, you can call matplotlib.pyplot.imshow to interpret the preceding numeric array as an image.

# Use false colors to visualize the array.

plt.imshow(x\_train[2917])

# Output row #10 of example #2917.

x\_train[2917][10]

# Output pixel #16 of row #10 of example #2900.

x\_train[2900][10][16]

Task 1: Normalize feature values

Complete the following code cell to map each feature value from its current representation (an integer between 0 and 255) to a floating-point value between 0 and 1.0. Store the floating-point values in x\_train\_normalized and x\_test\_normalized.

#@title Double-click to see a solution to Task 1.

x\_train\_normalized = x\_train / 255.0

x\_test\_normalized = x\_test / 255.0

print(x\_train\_normalized[2900][10]) # Output a normalized row

Define a plotting function

The following function plots an accuracy curve:

Define the plotting function

#@title Define the plotting function

def plot\_curve(epochs, hist, list\_of\_metrics):

"""Plot a curve of one or more classification metrics vs. epoch."""

# list\_of\_metrics should be one of the names shown in:

# https://www.tensorflow.org/tutorials/structured\_data/imbalanced\_data#define\_the\_model\_and\_metrics

plt.figure()

plt.xlabel("Epoch")

plt.ylabel("Value")

for m in list\_of\_metrics:

x = hist[m]

plt.plot(epochs[1:], x[1:], label=m)

plt.legend()

print("Loaded the plot\_curve function.")

Create a deep neural net model

The create\_model function defines the topography of the deep neural net, specifying the following:

The number of [layers](https://developers.google.com/machine-learning/glossary/#layer) in the deep neural net.

The number of [nodes](https://developers.google.com/machine-learning/glossary/#node) in each layer.

Any [regularization](https://developers.google.com/machine-learning/glossary/#regularization) layers.

The create\_model function also defines the [activation function](https://developers.google.com/machine-learning/glossary/#activation_function) of each layer. The activation function of the output layer is [softmax](https://developers.google.com/machine-learning/glossary/#softmax), which will yield 10 different outputs for each example. Each of the 10 outputs provides the probability that the input example is a certain digit.

**Note:** Unlike several of the recent Colabs, this exercise does not define feature columns or a feature layer. Instead, the model will train on the NumPy array.

def create\_model(my\_learning\_rate):

"""Create and compile a deep neural net."""

# All models in this course are sequential.

model = tf.keras.models.Sequential()

# The features are stored in a two-dimensional 28X28 array.

# Flatten that two-dimensional array into a one-dimensional

# 784-element array.

model.add(tf.keras.layers.Flatten(input\_shape=(28, 28)))

# Define the first hidden layer.

model.add(tf.keras.layers.Dense(units=32, activation='relu'))

# Define a dropout regularization layer.

model.add(tf.keras.layers.Dropout(rate=0.2))

# Define the output layer. The units parameter is set to 10 because

# the model must choose among 10 possible output values (representing

# the digits from 0 to 9, inclusive).

#

# Don't change this layer.

model.add(tf.keras.layers.Dense(units=10, activation='softmax'))

# Construct the layers into a model that TensorFlow can execute.

# Notice that the loss function for multi-class classification

# is different than the loss function for binary classification.

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=my\_learning\_rate),

loss="sparse\_categorical\_crossentropy",

metrics=['accuracy'])

return model

def train\_model(model, train\_features, train\_label, epochs,

batch\_size=None, validation\_split=0.1):

"""Train the model by feeding it data."""

history = model.fit(x=train\_features, y=train\_label, batch\_size=batch\_size,

epochs=epochs, shuffle=True,

validation\_split=validation\_split)

# To track the progression of training, gather a snapshot

# of the model's metrics at each epoch.

epochs = history.epoch

hist = pd.DataFrame(history.history)

return epochs, hist

Invoke the previous functions

Run the following code cell to invoke the preceding functions and actually train the model on the training set.

**Note:** Due to several factors (for example, more examples and a more complex neural network) training MNIST might take longer than training the California Housing Dataset.

# The following variables are the hyperparameters.

learning\_rate = 0.003

epochs = 50

batch\_size = 4000

validation\_split = 0.2

# Establish the model's topography.

my\_model = create\_model(learning\_rate)

# Train the model on the normalized training set.

epochs, hist = train\_model(my\_model, x\_train\_normalized, y\_train,

epochs, batch\_size, validation\_split)

# Plot a graph of the metric vs. epochs.

list\_of\_metrics\_to\_plot = ['accuracy']

plot\_curve(epochs, hist, list\_of\_metrics\_to\_plot)

# Evaluate against the test set.

print("\n Evaluate the new model against the test set:")

my\_model.evaluate(x=x\_test\_normalized, y=y\_test, batch\_size=batch\_size)

Task 2: Optimize the model

Experiment with the following:

number of hidden layers

number of nodes in each layer

dropout regularization rate

What trends did you discover? Can you reach at least 98% accuracy against the test set?

#@title Double-click to view some possible answers.

# It would take much too long to experiment

# fully with topography and dropout regularization

# rate. In the real world, you would

# also experiment with learning rate, batch size,

# and number of epochs. Since you only have a

# few minutes, searching for trends can be helpful.

# Here is what we discovered:

# \* Adding more nodes (at least until 256 nodes)

# to the first hidden layer improved accuracy.

# \* Adding a second hidden layer generally

# improved accuracy.

# \* When the model contains a lot of nodes,

# the model overfits unless the dropout rate

# is at least 0.5.

# We reached 98% test accuracy with the

# following configuration:

# \* One hidden layer of 256 nodes; no second

# hidden layer.

# \* dropout regularization rate of 0.4

# We reached 98.2% test accuracy with the

# following configuration:

# \* First hidden layer of 256 nodes;

# second hidden layer of 128 nodes.

# \* dropout regularization rate of 0.2

Embeddings

Um **embedding** é um espaço relativamente baixo em que é possível transformar vetores de alta dimensão. Com os embeddings, é mais fácil fazer o machine learning em entradas grandes, como vetores esparsos que representam palavras. O ideal é que um embedding capture algumas das semânticas da entrada colocando-as semanticamente semelhantes no espaço de embedding. Um embedding pode ser aprendizado e reutilizado em vários modelos.

**Objetivos de aprendizado**

* Saiba o que é uma incorporação e para que ela serve.
* Os embeddings codificam relações semânticas.
* Saiba como usar embeddings.
* Aprenda a treinar embeddings significativos (usando o word2vec, por exemplo).

Embeddings: motivação do filtro colaborativo

A **filtragem colaborativa** é a tarefa de fazer previsões sobre os interesses de um usuário com base nos interesses de muitos outros usuários. Como exemplo, vejamos a tarefa da recomendação de filme. Imagine que temos 500.000 usuários e uma lista dos filmes que cada usuário assistiu (de um catálogo de 1.000.000 de filmes). Nosso objetivo é recomendar filmes aos usuários.

Para resolver esse problema, alguns métodos são necessários para determinar quais filmes são semelhantes entre si. Podemos atingir essa meta incorporando os filmes em um espaço de baixa dimensão criado para que filmes semelhantes estejam próximos.

Antes de descrever como podemos aprender o embedding, primeiro exploramos o tipo de qualidades que queremos que ele tenha e como representaremos os dados de treinamento para aprender o embedding.

## Organize filmes em uma linha de número unidimensional

Para ajudar a desenvolver a intuição sobre embeddings, em um pedaço de papel, tente organizar os seguintes filmes em uma linha unidimensional para que os filmes mais próximos uns dos outros estejam mais relacionados:

| **Filme** | [**Classificação**](https://wikipedia.org/wiki/Motion_Picture_Association_of_America_film_rating_system#MPAA_film_ratings) | **Descrição** |
| --- | --- | --- |
| [Bleu](http://www.imdb.com/title/tt0108394/) (em inglês) | Direita | Uma viúva francesa lamenta a perda do marido e da filha depois que elas morrem em um acidente de carro. |
| [O Cavaleiro das Trevas Ressurge](http://www.imdb.com/title/tt1345836) | PG-13 | Batman tenta lutar para salvar Gotham City da aniquilação nuclear nesta sequência de [O Cavaleiro das Trevas](http://www.imdb.com/title/tt0468569/), ambientado no universo da DC Comics. |
| [Harry Potter e a Pedra Filosofal](http://www.imdb.com/title/tt0241527/) | PG | Um menino órfão descobre que é bruxo e se matricula na Escola de Magia e Bruxaria de Hogwarts, em que luta pela primeira vez contra o Senhor Voldemort. |
| [Os Incríveis](http://www.imdb.com/title/tt0317705/) | PG | Uma família de super-heróis forçados a viver como civis nos subúrbios saem da aposentadoria para salvar a corrida de super-heróis de Síndrome e seu robô assassino. |
| [Shrek](http://www.imdb.com/title/tt0126029/) (em inglês) | PG | Um ogro adorável e seu companheiro burro partiu em uma missão para resgatar a princesa Fiona, que é privada de um castelo por dragão. |
| [Guerra nas Estrelas](http://www.imdb.com/title/tt0076759) | PG | Luke Skywalker e Han Solo se unem a dois Androids para resgatar a Princesa Leia e salvar a galáxia. |
| [The Triplets de Belleville](http://www.imdb.com/title/tt0286244) | PG-13 | Quando o campeão profissional Champion é sequestrado durante o Tour de France, a avó e o cão acima do peso em busca de ajuda, com a ajuda de um trio de cantores de jazz idosos. |
| [Memento](http://www.imdb.com/title/tt0209144/) | Direita | Uma amnésia tenta desesperadamente resolver o assassinato de sua esposa tatuando pistas sobre seu corpo. |

## Organize filmes em um espaço bidimensional

Faça o mesmo exercício de antes, mas desta vez organize os mesmos filmes em um espaço bidimensional.

Embeddings: dados de entrada categóricos

Os **dados categóricos** se referem a recursos de entrada que representam um ou mais itens discretos de um conjunto finito de opções. Por exemplo, pode ser o conjunto de filmes que um usuário assistiu, o conjunto de palavras em um documento ou a profissão de uma pessoa.

Os dados categóricos são representados de maneira mais eficiente por meio de **tensores esparsos**, que são tensores com muito poucos elementos diferentes de zero. Por exemplo, se estivermos criando um modelo de recomendação de filmes, podemos atribuir um ID exclusivo a cada filme possível e, em seguida, representar cada usuário por um tensor esparso de filmes que eles assistiram, conforme mostrado na Figura 3.



**Figura 3. Dados para nosso problema de recomendação de filmes.**

Cada linha da matriz na Figura 3 é um exemplo que captura um histórico de visualização de filmes de um usuário, e é representada como um tensor esparso porque cada usuário assiste apenas a uma pequena fração de todos os filmes possíveis. A última linha corresponde ao tensor esparso [1, 3, 999999], usando os índices de vocabulário mostrados acima dos ícones de filme.

Da mesma forma, é possível representar palavras, frases e documentos como vetores esparsos em que cada palavra no vocabulário tem um papel semelhante aos filmes do exemplo de recomendação.

Para usar essas [representações](https://developers.google.com/machine-learning/crash-course/representation/video-lecture?hl=pt-br) em um sistema de machine learning, precisamos de uma maneira de representar cada vetor esparso como um vetor de números para que itens semanticamente semelhantes (filmes ou palavras) tenham distâncias semelhantes no espaço vetorial. Mas como representar uma palavra como vetor de números?

A maneira mais simples é definir uma camada de entrada gigante com um nó para cada palavra no seu vocabulário ou pelo menos um nó para cada palavra que aparece nos seus dados. Se 500.000 palavras únicas aparecerem nos seus dados, você poderá representar uma palavra com comprimento de 500.000 vetor e atribuir cada palavra a um slot no vetor.

Se você atribuir "horse" para indexar 1247, então para inserir "horse" em sua rede, você pode copiar 1 para o 1247o nó de entrada e 0s para todo o restante. Esse tipo de representação é chamado de **codificação one-hot**, porque apenas um índice tem um valor diferente de zero.

Normalmente, seu vetor pode conter contagens de palavras em um bloco de texto maior. Isso é conhecido como uma representação de "saco de palavras". Em um vetor de saco de palavras, vários dos 500.000 nós teriam um valor diferente de zero.

No entanto, independentemente dos valores diferentes de zero, um nó por palavra oferece vetores de entrada muito *esparsos*, que são vetores muito grandes com relativamente poucos valores diferentes de zero. As representações esparsas têm alguns problemas que podem dificultar o aprendizado eficiente de um modelo.

## Tamanho da rede

Grandes vetores de entrada significam um número muito grande de pesos para uma rede neural. Se houver M palavras no vocabulário e N nós na primeira camada da rede acima da entrada, você terá pesos MxN para treinar para essa camada. Um grande número de ponderações causa mais problemas:

* **Quantidade de dados**. Quanto mais pesos no modelo, mais dados você precisa para treinar de maneira eficaz.
* **Quantidade de computação**. Quanto mais pesos, mais computação é necessária para treinar e usar o modelo. É fácil exceder os recursos do seu hardware.

## Falta de relações significativas entre os vetores

Se você alimentar os valores de pixel de canais RGB em um classificador de imagens, faz sentido falar sobre valores de "close". O azul avermelhado é próximo do azul puro, tanto semanticamente quanto em termos da distância geométrica entre os vetores. Mas um vetor com 1 no índice 1247 para "quose;horse" não está mais perto de um vetor com 1 no índice de 50.430 para "quolo;antelope"" do que um vetor com 1 no índice de 238 para "television".

## A solução: embeddings

A solução para esses problemas é usar **embeddings**, que convertem grandes vetores esparsos em um espaço de menor dimensão que preserva relações semânticas. Vamos analisar as incorporações de maneira intuitiva, conceitual e programática nas seções a seguir deste módulo.

Embeddings: tradução para um espaço de menor dimensão

Resolva os principais problemas de dados de entrada esparsos mapeando os dados de alta dimensão em um espaço de dimensão inferior.

Como visto nos [exercícios de cinema](https://developers.google.com/machine-learning/crash-course/embeddings/motivation-from-collaborative-filtering?hl=pt-br#arrange-movies-on-a-one-dimensional-number-line), até mesmo um pequeno espaço multidimensional oferece a liberdade de agrupar itens semanticamente semelhantes e manter os itens diferentes um do outro. A posição (distância e direção) no espaço vetorial pode codificar a semântica em uma boa incorporação. Por exemplo, as seguintes visualizações de embeddings reais mostram relações geométricas que capturam relações semânticas como a relação entre um país e sua capital:

Esse tipo de espaço significativo dá ao sistema de machine learning oportunidades de detectar padrões que podem ajudar na tarefa de aprendizado.

## Como reduzir a rede

Embora queiramos dimensões suficientes para codificar relações semânticas avançadas, também queremos um espaço de incorporação pequeno o suficiente para que treine nosso sistema mais rapidamente. Uma incorporação útil pode estar na ordem de centenas de dimensões. Isso provavelmente é várias ordens de grandeza menor que o tamanho do seu vocabulário para uma tarefa de linguagem natural.

Embeddings: como obter embeddings

Há várias maneiras de conseguir um embedding, incluindo um algoritmo de última geração criado no Google.

## Técnicas padrão de redução de dimensionalidade

Existem muitas técnicas matemáticas atuais para capturar a estrutura importante de um espaço de alta dimensão em um espaço de baixa dimensão. Em teoria, qualquer uma dessas técnicas pode ser usada para criar um embedding para um sistema de aprendizado de máquina.

Por exemplo, a [análise de componente principal](https://wikipedia.org/wiki/Principal_component_analysis) (PCA, na sigla em inglês) foi usada para criar embeddings de palavras. Considerando um conjunto de instâncias como o pacote de vetores de palavras, o PCA tenta encontrar dimensões altamente correlacionadas que podem ser recolhidas em uma única dimensão.

## Word2vec

O Word2vec é um algoritmo inventado no Google para treinar embeddings de palavra. O Word2vec depende da **hipótese de distribuição** para mapear palavras semanticamente semelhantes para vetores de embedding geométricos.

A hipótese de distribuição determina que as palavras que geralmente têm as mesmas palavras vizinhas tendem a ser semanticamente semelhantes. Tanto "dog" quanto "cat" frequentemente aparecem próximos à palavra "veterinarian" e esse fato reflete a semelhança semântica deles. Como o linguista John Firth o colocou em 1957, "você conhecerá uma palavra da empresa que mantém"

O Word2Vec explora informações contextuais como essa treinando uma rede neural para distinguir diferentes grupos de palavras que ocorrem em palavras aleatórias. A camada de entrada utiliza uma representação esparsa de uma palavra de destino com uma ou mais palavras de contexto. Essa entrada se conecta a uma única camada menor oculta.

Em uma versão do algoritmo, o sistema faz um exemplo negativo substituindo uma palavra de ruído aleatório pela palavra de destino. Considerando o exemplo "positivo" do avião voa" o sistema pode alternar em "jogging" para criar o exemplo negativo de contraste, "quot;

A outra versão do algoritmo cria exemplos negativos combinando as palavras de destino verdadeiras com as palavras de contexto escolhidas aleatoriamente. Assim, ele pode usar os exemplos positivos (o, o plano), (voar, avião) e os exemplos negativos (compilado, plano), (quem, avião) e aprender a identificar quais pares realmente apareceram juntos no texto.

No entanto, o classificador não é o objetivo real de nenhuma das versões do sistema. Depois que o modelo for treinado, você vai ter um embedding. Você pode usar os pesos que conectam a camada de entrada à camada escondida para mapear representações esparsas de palavras para vetores menores. Esse embedding pode ser reutilizado em outros classificadores.

Para mais informações sobre o Word2vec, consulte o [tutorial no tensorflow.org](https://www.tensorflow.org/tutorials/word2vec/index.html?hl=pt-br).

## Como treinar um embedding como parte de um modelo maior

Também é possível aprender a incorporação como parte da rede neural para sua tarefa de destino. Essa abordagem oferece uma incorporação bem personalizada para seu sistema específico, mas pode levar mais tempo do que treinar o embedding separadamente.

Em geral, quando há dados esparsos (ou densos que você queira incorporar), é possível criar uma unidade de embedding que seja apenas um tipo especial de unidade oculta de tamanho d. Essa camada de embedding pode ser combinada com outros recursos e camadas ocultas. Como em qualquer DNN, a camada final será a perda que está sendo otimizada. Por exemplo, imagine que vamos realizar uma colaboração colaborativa em que a meta é prever os interesses de um usuário em relação aos de outros. Podemos modelar isso como um problema de aprendizado supervisionado definindo automaticamente ou mantendo um pequeno número de filmes que o usuário assistiu como rótulos positivos. Em seguida, otimizamos uma perda softmax.

**Figura 5. Exemplo de arquitetura DNN para aprender embeddings de filmes de dados de filtragem colaborativa.**

Outro exemplo: se você quiser criar uma camada de embedding para as palavras em um anúncio imobiliário como parte de uma DNN para prever os preços de imóveis, otimize uma perda L2 usando o preço de venda conhecido das casas nos dados de treinamento como o identificador.

Ao aprender uma incorporação de d-dimensional, cada item é mapeado a um ponto em um espaço de d-dimensional para que os itens semelhantes fiquem próximos neste espaço. A Figura 6 ajuda a ilustrar a relação entre os pesos aprendidos na camada de embedding e a visualização geométrica. Os pesos de borda entre um nó de entrada e os nós na camada de embedding de dimensão d correspondem aos valores de coordenadas para cada um dos eixos d.

**Figura 6. Uma visualização geométrica dos pesos da camada de embedding.**

Production ML Systems

O machine learning é muito mais do que apenas implementar um algoritmo de ML. Um sistema de ML de produção envolve um número significativo de componentes.

## Resumo da aula sobre vídeo

Até agora, o Curso intensivo de machine learning tem se concentrado na criação de modelos de ML. No entanto, como a figura a seguir sugere, sistemas de ML de produção real são grandes ecossistemas em que o modelo é apenas uma parte.

**Figura 1. Sistema de ML de produção real.**

O código de ML está no centro de um sistema de produção de ML real, mas essa caixa geralmente representa apenas 5% ou menos do código geral do sistema de produção de ML total. Isso não é um erro. Observe que um sistema de produção de ML dedica recursos consideráveis para inserir dados: coletá-los, verificá-los e extrair atributos deles. Além disso, observe que uma infraestrutura de exibição precisa estar em vigor para colocar as previsões do modelo de ML em uso prático no mundo real.

Felizmente, muitos dos componentes na figura anterior são reutilizáveis. Além disso, você não precisa criar todos os componentes na Figura 1.

O [TensorFlow Extended (TFX)](https://www.tensorflow.org/tfx?hl=pt-br) é uma plataforma completa para implantar pipelines de produção de ML.

Os módulos a seguir vão ajudar a orientar suas decisões de design na criação de um sistema de ML de produção.

Treinamento estático vs. dinâmico

Em termos gerais, há duas maneiras de treinar um modelo:

* Um **modelo estático** é treinado off-line. Ou seja, treinamos o modelo exatamente uma vez e usamos esse modelo treinado por um tempo.
* Um **modelo dinâmico** é treinado on-line. Ou seja, os dados estão sempre entrando no sistema e estamos incorporando esses dados ao modelo usando atualizações contínuas.

**Objetivo de aprendizado**

* Identificar as vantagens e desvantagens do treinamento estático e dinâmico

## Resumo da aula sobre vídeo

Em termos gerais, os pontos a seguir dominam a decisão de treinamento estático e dinâmico:

* Modelos estáticos são mais fáceis de criar e testar.
* Os modelos dinâmicos se adaptam à mudança de dados. O mundo é um lugar altamente alternável. As previsões de vendas criadas com base nos dados do ano passado provavelmente não serão bem-sucedidas.

Se o conjunto de dados realmente não mudar ao longo do tempo, escolha o treinamento estático porque ele é mais barato de criar e manter do que o dinâmico. No entanto, muitas fontes de informações mudam com o tempo, mesmo aquelas com recursos que você considera tão constantes quanto, por exemplo, o nível do mar. Moral: mesmo com o treinamento estático, você ainda precisa monitorar seus dados de entrada para mudanças.

Por exemplo, considere um modelo treinado para prever a probabilidade de os usuários comprarem flores. Devido à pressão do tempo, o modelo é treinado apenas uma vez usando um conjunto de dados do comportamento de compra de flores durante julho e agosto. O modelo é enviado para exibir previsões em produção, mas nunca é atualizado. O modelo funciona bem por vários meses, mas, em seguida, faz previsões muito ruins por volta do [Dia dos Namorados](https://wikipedia.org/wiki/Valentine's_Day), porque o comportamento do usuário durante esse período muda significativamente.

Inferência estática x dinâmica

Você pode escolher uma das seguintes estratégias de inferência:

* **inferência off-line**, o que significa que você pode fazer todas as previsões possíveis em um lote, usando um MapReduce ou algo semelhante. Em seguida, você grava as previsões em um SSTable ou no Bigtable e os alimenta em uma tabela de cache/consulta.
* **inferência on-line**, ou seja, a previsão sob demanda usando um servidor

**Objetivos de aprendizado**

* Entender os prós e contras da inferência estática e dinâmica
* Estime as necessidades de treinamento e disponibilização em cenários reais.

## Resumo da aula sobre vídeo

Veja as vantagens e desvantagens da inferência off-line:

* P: Não é preciso se preocupar muito com o custo de inferências.
* Pro: provavelmente pode usar cota em lote ou algum MapReduce gigante.
* P: Pode realizar verificações pós-verificação de previsões antes de enviar.
* Desvantagem: só é possível prever o que sabemos. Isso é ruim para uma cauda longa.
* Contra: a latência da atualização provavelmente é medida em horas ou dias.

Veja as vantagens e desvantagens da inferência on-line:

* Pro: é possível fazer uma previsão em qualquer novo item conforme ele chega. Isso é ótimo para cauda longa.
* Desvantagem: computação intensiva e sensível à latência pode limitar a complexidade do modelo.
* Desvantagem: as necessidades de monitoramento são mais intensas.

Dependências de dados

Os dados são tão importantes para os desenvolvedores de ML quanto o código para os programadores tradicionais. Esta lição se concentra nos tipos de perguntas que você precisa fazer sobre seus dados.

**Objetivos de aprendizado**

* Entender as dependências de dados nos sistemas de ML de produção.

## Resumo da aula sobre vídeo

O comportamento de um sistema de ML depende do comportamento e das qualidades dos [recursos de entrada](https://developers.google.com/machine-learning/crash-course/representation?hl=pt-br). Os dados de entrada desses atributos também mudam, assim como o modelo. Às vezes, a mudança é desejável, mas não é.

No desenvolvimento de software tradicional, você se concentra mais no código do que nos dados. No desenvolvimento de machine learning, embora a programação ainda faça parte do job, seu foco precisa ser ampliado para incluir dados. Por exemplo, em projetos de desenvolvimento de software tradicionais, é recomendável programar testes de unidade para validar o código. Em projetos de ML, também é preciso testar, verificar e monitorar continuamente os dados de entrada.

Por exemplo, é preciso monitorar continuamente seu modelo para remover os recursos não utilizados (ou pouco usados). Imagine um determinado atributo que contribui pouco ou nada para o modelo. Se os dados de entrada desse recurso mudarem abruptamente, o comportamento do seu modelo também pode mudar abruptamente de maneira indesejável.

### Confiabilidade

Algumas perguntas a serem feitas sobre a confiabilidade dos dados de entrada:

* O sinal vai estar sempre disponível ou é de uma fonte não confiável? Exemplo:
  + O sinal está vindo de um servidor que falha sob carga pesada?
  + O sinal vem de humanos que vão de férias todos os anos?

### Controle de versões

Algumas perguntas a serem feitas sobre o controle de versões:

* O sistema que calcula esses dados muda? Nesse caso, faça o seguinte:
  + Com que frequência?
  + Como você sabe quando esse sistema muda?

Às vezes, os dados vêm de um processo upstream. Se esse processo mudar abruptamente, seu modelo pode sofrer.

Considere criar sua própria cópia dos dados recebidos do processo upstream. Em seguida, avance para a próxima versão dos dados upstream quando tiver certeza de que é seguro fazer isso.

### Necessidade

A seguinte pergunta pode lembrar você de [regularização](https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/video-lecture?hl=pt-br):

* A utilidade do recurso justifica o custo de inclusão?

É sempre tentador adicionar mais atributos ao modelo. Por exemplo, suponha que você encontre um novo atributo com uma adição que torne seu modelo um pouco mais preciso. Uma precisão certa é melhor do que com menos precisão. No entanto, agora você acabou de aumentar a carga da manutenção. Esse recurso pode ser degradado inesperadamente, portanto, é preciso monitorá-lo. Pense com cuidado antes de adicionar recursos que levem a pequenas vitórias de curto prazo.

### Correlações

Alguns atributos estão correlacionados (positiva ou negativamente) com outros. Faça esta pergunta:

* Existe algum recurso tão complexo que é preciso chamar a atenção para outras estratégias?

### Ciclos de feedbacks

Às vezes, um modelo pode afetar os próprios dados de treinamento. Por exemplo, os resultados de alguns modelos, por sua vez, são atributos de entrada direta ou indiretas para esse mesmo modelo.

Às vezes, um modelo pode afetar outro. Por exemplo, considere dois modelos para prever preços de ações:

* Modelo A, que é um modelo preditivo inadequado.
* Modelo B.

Como o modelo A tem bugs, ele decide comprar um estoque estocado por engano. Essas compras impulsionam o preço da ação X. O modelo B usa o preço do estoque X como atributo de entrada, portanto, o modelo B pode facilmente chegar a algumas conclusões falsas sobre o valor do estoque X. O modelo B, portanto, poderia comprar ou vender ações da bolsa X com base no comportamento com bugs do modelo A. O comportamento do modelo B, por sua vez, pode afetar o modelo A, possivelmente acionando uma [mania de tulipas](https://wikipedia.org/wiki/Tulip_mania) ou um slide no inventário da empresa X

Justiça

**Objetivos de aprendizado**

* Saber quais são os vieses humanos comuns que podem ser reproduzidos acidentalmente por algoritmos de ML.
* Explore de maneira proativa os dados para identificar fontes de viés antes de treinar um modelo
* Avaliar previsões do modelo para viés

A avaliação responsável de um modelo de machine learning exige mais do que apenas calcular as métricas de perda. Antes de colocar um modelo em produção, é essencial auditar os dados de treinamento e avaliar as previsões por viés.

Neste módulo, analisamos diferentes tipos de vieses humanos que podem se manifestar nos dados de treinamento. Ele fornece estratégias para identificá-los e avaliar os efeitos.

Justiça: tipos de viés

Modelos de machine learning não são inerentemente objetivos. Os engenheiros treinam modelos ao alimentar um conjunto de dados de exemplos de treinamento, e o envolvimento humano no provisionamento e na seleção desses dados pode tornar as previsões de um modelo suscetíveis a vieses.

Ao criar modelos, é importante conhecer vieses humanos comuns que possam se manifestar nos seus dados. Assim, você pode tomar medidas proativas para reduzir os efeitos deles.

**AVISO**: o inventário de vieses a seguir fornece apenas uma pequena seleção de vieses que são frequentemente descobertos em conjuntos de dados de machine learning. Esta lista não é considerada completa. O [catálogo de vieses cognitivos](https://wikipedia.org/wiki/List_of_cognitive_biases) da Wikipédia enumera mais de cem tipos diferentes de vieses humanos que podem afetar o julgamento. Ao auditar seus dados, fique atento a todas as fontes de viés em potencial que podem distorcer as previsões do modelo.

## viés de relatório

O **viés de relatório** ocorre quando a frequência de eventos, propriedades e/ou resultados capturados em um conjunto de dados não reflete com precisão a frequência real deles. Esse viés pode surgir porque as pessoas tendem a se concentrar em documentar circunstâncias incomuns ou especialmente memoráveis, supondo que o comum possa "ir sem dizer."

**EXEMPLO**: um modelo de análise de sentimento é treinado para prever se as avaliações do livro são positivas ou negativas com base em um corpus de envios de usuários para um site conhecido. A maioria das avaliações no conjunto de dados de treinamento reflete opiniões extremas (revisores que adoraram ou odiaram um livro), porque as pessoas têm menos probabilidade de enviar uma revisão de um livro se não responderem totalmente a ele. Como resultado, o modelo é menos capaz de prever corretamente a sentimento de avaliações que usam linguagem mais sutil para descrever um livro.

## Viés de automação

O **viés de automação** é uma tendência a favorecer os resultados gerados por sistemas automatizados em relação aos gerados por sistemas não automatizados, independentemente das taxas de erro de cada um.

**EXEMPLO**: engenheiros de software que trabalhavam para um fabricante de pináculos que estavam ansiosos para implantar o novo modelo "groundbreaking" que treinaram para identificar desvios de dentes, até que o supervisor de fábrica indicou que as taxas de precisão e recall do modelo foram 15% mais baixas do que as de inspetores humanos.

## Viés de seleção

O **viés de seleção** ocorre quando os exemplos de um conjunto de dados são escolhidos de maneira que não reflete a distribuição real deles. O viés de seleção pode ter muitas formas diferentes:

* **Viés de cobertura**: os dados não são selecionados de maneira representativa.

**EXEMPLO**: um modelo é treinado para prever as vendas futuras de um novo produto com base em pesquisas por smartphone conduzidas com uma amostra de consumidores que compraram o produto. Os consumidores que optaram por comprar um produto concorrente não foram entrevistados e, como resultado, esse grupo de pessoas não foi representado nos dados de treinamento.

* **Viés de não resposta** (ou **viés de participação**): os dados acabam sendo não representativos devido a lacunas de participação no processo de coleta de dados.

**EXEMPLO**: um modelo é treinado para prever as vendas futuras de um novo produto com base em pesquisas por smartphone conduzidas com uma amostra de consumidores que compraram o produto e com uma amostra de consumidores que compraram um produto concorrente. Os consumidores que compraram o produto concorrente tinham 80% mais chances de se recusar a responder à pesquisa, e os dados estavam sub-representados na amostra.

* **Viés de amostragem**: a escolha aleatória não é usada durante a coleta de dados.

**EXEMPLO**: um modelo é treinado para prever as vendas futuras de um novo produto com base em pesquisas por smartphone conduzidas com uma amostra de consumidores que compraram o produto e com uma amostra de consumidores que compraram um produto concorrente. Em vez de segmentar aleatoriamente os consumidores, o pesquisador escolheu os primeiros 200 consumidores que responderam a um e-mail, que podem estar mais entusiasmados com o produto do que os compradores médios.

## Viés de atribuição do grupo

O **viés de atribuição de grupos** é uma tendência a generalizar o que é verdadeiro de indivíduos para um grupo inteiro ao qual eles pertencem. Duas manifestações importantes desse viés são:

* **Viés no grupo**: uma preferência para membros de um grupo a que você também pertence ou para características que você também compartilha.

**EXEMPLO**: dois engenheiros que treinam um modelo de triagem de currículos para desenvolvedores de software estão dispostos a acreditar que os candidatos que participaram da mesma academia de ciência da computação que ambos foram mais qualificados para a função.

* **Viés de homogeneidade fora do grupo**: uma tendência a estereotipar membros individuais de um grupo ao qual *você não pertence* ou ver as características deles como mais uniformes.

**EXEMPLO**: dois engenheiros que treinam um modelo de triagem de currículo para desenvolvedores de software estão dispostos a acreditar que todos os candidatos que não frequentam uma academia de ciência da computação não têm conhecimento suficiente para o papel.

## Viés implícito

O **viés implícito** ocorre quando suposições são feitas com base em modelos mentais e experiências pessoais próprios que não se aplicam de modo mais geral.

**EXEMPLO**: um engenheiro que treina um modelo de reconhecimento de gestos usa um [shake de cabeça](https://wikipedia.org/wiki/Head_shake) como atributo para indicar que uma pessoa está comunicando a palavra "quot;no." No entanto, em algumas regiões do mundo, uma balanço na cabeça significa "yes."

Uma forma comum de viés implícito é o **viés de confirmação**, em que criadores de modelos processam inconscientemente dados de maneira que afirmem crenças e hipóteses preexistentes. Em alguns casos, um criador de modelos pode continuar treinando um modelo até produzir um resultado que se alinhe à hipótese original. Isso é chamado de **viés do experimento**.

**EXEMPLO**: um engenheiro está criando um modelo que prevê agressividade em cães com base em diversos atributos (altura, peso, raça, ambiente). A engenheira teve um encontro desagradável com um poodle de brinquedo hiperativo como criança e, desde então, associou a raça a agressão. Quando o modelo treinado previu que a maioria dos poodles de brinquedo era relativamente dócil, o engenheiro o treinou várias vezes até produzir um resultado que mostrasse poodles menores para serem mais violentos.

Justiça: identificação de tendências

À medida que você explora seus dados para determinar a melhor maneira de [representar](https://developers.google.com/machine-learning/crash-course/representation?hl=pt-br) esses modelos no modelo, é importante considerar os problemas de imparcialidade e auditar proativamente fontes em potencial de viés.

Onde o viés pode parecer distorcido? Veja três sinais de alerta no seu conjunto de dados.

## Valores de atributo ausente

Se o conjunto de dados tiver um ou mais atributos sem muitos valores, isso poderá indicar que determinadas características importantes dele são sub-representadas.

Por exemplo, a tabela abaixo mostra um resumo das principais estatísticas de um subconjunto de recursos no [conjunto de dados de habitação da Califórnia](https://developers.google.com/machine-learning/crash-course/california-housing-data-description?hl=pt-br), armazenado em um DataFrame do pandas e gerado via [DataFrame.describe](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.describe.html). Todos os recursos têm um count de 17000, indicando que não há valores ausentes:

|  | **longitude** | **latitude** | **total\_rooms** | **population** | **households** | **median\_income** | **median\_house\_value** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **contagem** | 17.000,0 | 17.000,0 | 17.000,0 | 17.000,0 | 17.000,0 | 17.000,0 | 17.000,0 |
| **média** | -119,6 | 35,6 | 2.643,7 | 1.429,6 | 501,2 | 3,9 | 207,3 |
| **padrão** | 2 | 2.1 | 2179,9 | 1147,9 | 384,5 | 1.9 | 116,0 |
| **min** | -124,3 | 32,5 | 2 | 3 | 1.0 | 0,5 | 15 |
| **25%** | -121,8 | 33,9 | 1462,0 | 790,0 | 282,0 | 2.6 | 119,4 |
| **50%** | -118,5 | 34,2 | 2127,0 | 1167,0 | 409,0 | 3.5 | 180,4 |
| **75%** | -118,0 | 37,7 | 3151,2 | 1721,0 | 605,2 | 4,8 | 265,0 |
| **max** | -114,3 | 42,0 | 37937,0 | 35682,0 | 6082,0 | 15 | 500,0 |

Vamos supor que três recursos (population, households e median\_income) tenham apenas uma contagem de 3000. Em outras palavras, que havia 14.000 valores ausentes para cada recurso:

|  | **longitude** | **latitude** | **total\_rooms** | **population** | **households** | **median\_income** | **median\_house\_value** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **contagem** | 17.000,0 | 17.000,0 | 17.000,0 | 3000,0 | 3000,0 | 3000,0 | 17.000,0 |
| **média** | -119,6 | 35,6 | 2.643,7 | 1.429,6 | 501,2 | 3,9 | 207,3 |
| **padrão** | 2 | 2.1 | 2179,9 | 1147,9 | 384,5 | 1.9 | 116,0 |
| **min** | -124,3 | 32,5 | 2 | 3 | 1.0 | 0,5 | 15 |
| **25%** | -121,8 | 33,9 | 1462,0 | 790,0 | 282,0 | 2.6 | 119,4 |
| **50%** | -118,5 | 34,2 | 2127,0 | 1167,0 | 409,0 | 3.5 | 180,4 |
| **75%** | -118,0 | 37,7 | 3151,2 | 1721,0 | 605,2 | 4,8 | 265,0 |
| **max** | -114,3 | 42,0 | 37937,0 | 35682,0 | 6082,0 | 15 | 500,0 |

Esses 14.000 valores ausentes dificultariam a correlação precisa de renda familiar com preços médios de casas. Antes de treinar um modelo com esses dados, é prudente investigar a causa desses valores ausentes para garantir que não haja vieses latentes responsáveis pela ausência de dados de renda e população.

## Valores de atributo inesperados

Ao analisar os dados, procure também exemplos que contenham valores de recursos que se destacam como especialmente não característicos ou incomuns. Esses valores de recurso inesperados podem indicar problemas ocorridos durante a coleta de dados ou outras imprecisões que podem introduzir viés.

Por exemplo, veja os seguintes exemplos do conjunto de dados de habitação da Califórnia:

|  | **longitude** | **latitude** | **total\_rooms** | **population** | **households** | **median\_income** | **median\_house\_value** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | -121,7 | 38.0 | 7105,0 | 3523,0 | 1088,0 | 5 | 0,2 |
| **2** | -122,4 | 37,8 | 2479,0 | 1816,0 | 496,0 | 3.1 | 0,3 |
| **3** | -122,0 | 37,0 | 2813,0 | 1337,0 | 477,0 | 3.7 | 0,3 |
| **4** | -103,5 | 43,8 | 2212,0 | 803,0 | 144,0 | 5.3 | 0,2 |
| **5** | -117,1 | 32,8 | 2.963,0 | 1162,0 | 556,0 | 3.6 | 0,2 |
| **6** | -118,0 | 33,7 | 3396,0 | 1542,0 | 472,0 | 7.4 | 0,4 |

Você consegue identificar valores de atributo inesperados?

## Desvios nos dados

Qualquer tipo de distorção nos dados, em que determinados grupos ou características são sub-representados ou super-representados em relação à precedência real deles, pode introduzir vieses no modelo.

Se você concluiu o [exercício de programação de validação](https://developers.google.com/machine-learning/crash-course/validation/programming-exercise?hl=pt-br), pode se lembrar de descobrir como uma falha na ordem aleatória do conjunto de dados de habitação da Califórnia antes de dividi-lo em conjuntos de treinamento e validação resultou em um desvio de dados pronunciado. A Figura 1 mostra um subconjunto de dados extraídos do conjunto completo que representa exclusivamente a região noroeste da Califórnia.

**Figura 1. Mapa de estado da Califórnia sobreposto por dados do conjunto de dados de habitação da Califórnia. Cada ponto representa um bloco habitacional, com cores que vão de azul a vermelho, correspondente ao preço médio da casa, de baixo para alto, respectivamente.**

Se essa amostra não representativa fosse usada para treinar um modelo para prever preços de casas na Califórnia, a falta de dados imobiliários de partes sul da Califórnia seria problemática. O viés geográfico codificado no modelo pode afetar negativamente os compradores de imóveis em comunidades não representadas.

Imparcialidade: avaliação de viés

Ao avaliar um modelo, as métricas calculadas com base em um conjunto inteiro de teste ou validação nem sempre dão uma visão precisa de como o modelo é justo.

Considere um novo modelo desenvolvido para prever a presença de tumors que são avaliados em comparação com um conjunto de validação de 1.000 pacientes. 500 registros são de pacientes do sexo feminino, e 500 registros são de pacientes do sexo masculino. A [matriz de confusão](https://developers.google.com/machine-learning/glossary?hl=pt-br#confusion_matrix) a seguir resume os resultados de todos os 1.000 exemplos:

|  |  |
| --- | --- |
| Verdadeiros positivos (VPs): 16 | Falsos positivos (FPs): 4 |
| Falsos negativos (FNs): 6 | Verdadeiros negativos (VNs): 974 |
| Precision = TP / TP + FP = 16 / 16 + 4 = 0.800 | |
| Recall = TP / TP + FN = 16 / 16 + 6 = 0.727 | |

Estes resultados parecem promissores: precisão de 80% e recall de 72,7%. Mas o que acontece se calcularmos o resultado separadamente para cada conjunto de pacientes? Vamos dividir os resultados em duas matrizes de confusão separadas: uma para pacientes femininos e outra para pacientes masculinos.

#### Resultados de pacientes do sexo feminino

|  |  |
| --- | --- |
| Verdadeiros positivos (VPs): 10 | Falsos positivos (FPs): 1 |
| Falsos negativos (FNs): 1 | Verdadeiros negativos (VNs): 488 |
| Precision = TP / TP + FP = 10 / 10 + 1 = 0.909 | |
| Recall = TP / TP + FN = 10 / 10 + 1 = 0.909 | |

#### Resultados de pacientes do sexo masculino

|  |  |
| --- | --- |
| Verdadeiros positivos (VPs): 6 | Falsos positivos (FPs): 3 |
| Falsos negativos (FNs): 5 | Verdadeiros negativos (VNs): 486 |
| Precision = TP / TP + FP = 6 / 6 + 3 = 0.667 | |
| Recall = TP / TP + FN = 6 / 6 + 5 = 0.545 | |

Quando calculamos métricas separadamente para pacientes femininos e masculinos, vemos grandes diferenças no desempenho do modelo para cada grupo.

Pacientes:

* Das 11 pacientes pacientes com tumors, o modelo prevê corretamente positivamente para 10 pacientes (taxa de recall: 90,9%). Em outras palavras, **o modelo perde um diagnóstico de tumor em 9,1% dos casos femininos**.
* Da mesma forma, quando o modelo retorna positivo para tumor em pacientes do sexo feminino, ele está correto em 10 de 11 casos (taxa de precisão: 90,9%). Em outras palavras, **o modelo prevê incorretamente o tumor em 9,1% dos casos femininos**.

Pacientes homens:

* No entanto, entre os 11 pacientes do sexo masculino que realmente têm tumors, o modelo prevê corretamente positivamente apenas seis pacientes (taxa de recall: 54,5%). Isso significa que **o modelo deixa de diagnosticar tumor em 45,5% dos casos masculinos**.
* Quando o modelo retorna positivo para tumor em pacientes do sexo masculino, ele está correto em apenas 6 de 9 casos (taxa de precisão: 66,7%). Em outras palavras, **o modelo prevê incorretamente o tumor em 33,3% dos casos de homens**.

Agora temos uma compreensão muito melhor dos vieses inerentes às previsões do modelo, bem como os riscos para cada subgrupo se o modelo for lançado para uso médico na população geral.

## Recursos adicionais de imparcialidade

A imparcialidade é um subcampo relativamente novo na disciplina de machine learning. Para saber mais sobre pesquisas e iniciativas dedicadas ao desenvolvimento de novas ferramentas e técnicas para identificar e mitigar vieses em modelos de machine learning, acesse [Página de recursos de imparcialidade de machine learning do Google](https://developers.google.com/machine-learning/fairness-overview?hl=pt-br).

Justiça: exercício de programação

O exercício a seguir demonstra como auditar conjuntos de dados com imparcialidade e uma estratégia para avaliar um modelo para imparcialidade:

 Introdução ao exercício de [imparcialidade](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/intro_to_ml_fairness.ipynb?utm_source=mlcc&%3Butm_campaign=colab-external&%3Butm_medium=referral&%3Butm_content=fairness_tf2-colab&%3Bhl=+l10n-placeholder1&hl=pt-br) no Colab (em inglês).

Introduction to Fairness in ML

Disclaimer

This exercise explores just a small subset of ideas and techniques relevant to fairness in machine learning; it is not the whole story!

Learning Objectives

Increase awareness of different types of biases that can manifest in model data.

Explore feature data to proactively identify potential sources of bias before training a model.

Evaluate model performace by subgroup rather than in aggregate.

Overview

In this exercise, you'll explore datasets and evaluate classifiers with fairness in mind, noting the ways undesirable biases can creep into machine learning (ML).

Throughout, you will see **FairAware** tasks, which provide opportunities to contextualize ML processes with respect to fairness. In performing these tasks, you'll identify biases and consider the long-term impact of model predictions if these biases are not addressed.

About the Dataset and Prediction Task

In this exercise, you'll work with the [Adult Census Income dataset](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Farchive.ics.uci.edu%2Fml%2Fdatasets%2FCensus%2BIncome), which is commonly used in machine learning literature. This data was extracted from the [1994 Census bureau database](https://colab.research.google.com/corgiredirector?site=http%3A%2F%2Fwww.census.gov%2Fen.html) by Ronny Kohavi and Barry Becker.

Each example in the dataset contains the following demographic data for a set of individuals who took part in the 1994 Census:

Numeric Features

age: The age of the individual in years.

fnlwgt: The number of individuals the Census Organizations believes that set of observations represents.

education\_num: An enumeration of the categorical representation of education. The higher the number, the higher the education that individual achieved. For example, an education\_num of 11 represents Assoc\_voc (associate degree at a vocational school), an education\_num of 13 represents Bachelors, and an education\_num of 9 represents HS-grad (high school graduate).

capital\_gain: Capital gain made by the individual, represented in US Dollars.

capital\_loss: Capital loss made by the individual, represented in US Dollars.

hours\_per\_week: Hours worked per week.

Categorical Features

workclass: The individual's type of employer. Examples include: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, and Never-worked.

education: The highest level of education achieved for that individual.

marital\_status: Marital status of the individual. Examples include: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, and Married-AF-spouse.

occupation: The occupation of the individual. Example include: tech-support, Craft-repair, Other-service, Sales, Exec-managerial and more.

relationship: The relationship of each individual in a household. Examples include: Wife, Own-child, Husband, Not-in-family, Other-relative, and Unmarried.

gender: Gender of the individual available only in binary choices: Female or Male.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Black, and Other.

native\_country: Country of origin of the individual. Examples include: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, and more.

Prediction Task

The prediction task is to **determine whether a person makes over $50,000 US Dollar a year.**

Label

income\_bracket: Whether the person makes more than $50,000 US Dollars annually.

Notes on Data Collection

All the examples extracted for this dataset meet the following conditions:

age is 16 years or older.

The adjusted gross income (used to calculate income\_bracket) is greater than $100 USD annually.

fnlwgt is greater than 0.

hours\_per\_week is greater than 0.

Setup

We'll import the necessary modules to run the code in the rest of this Colaboratory notebook.

In addition to importing the usual libraries, this setup code cell also installs [Facets](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fpair-code.github.io%2Ffacets%2F), an open-source tool created by [PAIR](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fresearch.google%2Fteams%2Fbrain%2Fpair%2F) that contains two robust visualizations we'll be using to aid in understanding and analyzing ML datasets.

Import revelant modules and install Facets

#@title Import revelant modules and install Facets

import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import layers

from matplotlib import pyplot as plt

from matplotlib import rcParams

import seaborn as sns

# The following lines adjust the granularity of reporting.

pd.options.display.max\_rows = 10

pd.options.display.float\_format = "{:.1f}".format

from google.colab import widgets

# For facets

from IPython.core.display import display, HTML

import base64

!pip install facets-overview==1.1.1

from facets\_overview.feature\_statistics\_generator import FeatureStatisticsGenerator

Load the Adult Dataset

With the modules now imported, we can load the Adult dataset into a pandas DataFrame data structure.

COLUMNS = ["age", "workclass", "fnlwgt", "education", "education\_num",

"marital\_status", "occupation", "relationship", "race", "gender",

"capital\_gain", "capital\_loss", "hours\_per\_week", "native\_country",

"income\_bracket"]

train\_csv = tf.keras.utils.get\_file('adult.data',

'https://download.mlcc.google.com/mledu-datasets/adult\_census\_train.csv')

test\_csv = tf.keras.utils.get\_file('adult.test' ,

'https://download.mlcc.google.com/mledu-datasets/adult\_census\_test.csv')

train\_df = pd.read\_csv(train\_csv, names=COLUMNS, sep=r'\s\*,\s\*',

engine='python', na\_values="?")

test\_df = pd.read\_csv(test\_csv, names=COLUMNS, sep=r'\s\*,\s\*', skiprows=[0],

engine='python', na\_values="?")

# Strip trailing periods mistakenly included only in UCI test dataset.

test\_df['income\_bracket'] = test\_df.income\_bracket.str.rstrip('.')

Analyzing the Adult Dataset with Facets

As mentioned in MLCC, it is important to understand your dataset before diving straight into the prediction task.

Some important questions to investigate when auditing a dataset for fairness:

**Are there missing feature values for a large number of observations?**

**Are there features that are missing that might affect other features?**

**Are there any unexpected feature values?**

**What signs of data skew do you see?**

To start, we can use [Facets Overview](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fpair-code.github.io%2Ffacets%2F), an interactive visualization tool that can help us explore the dataset. With Facets Overview, we can quickly analyze the distribution of values across the Adult dataset.

Visualize the Data in Facets

#@title Visualize the Data in Facets

fsg = FeatureStatisticsGenerator()

dataframes = [

{'table': train\_df, 'name': 'trainData'}]

censusProto = fsg.ProtoFromDataFrames(dataframes)

protostr = base64.b64encode(censusProto.SerializeToString()).decode("utf-8")

HTML\_TEMPLATE = """<script src="https://cdnjs.cloudflare.com/ajax/libs/webcomponentsjs/1.3.3/webcomponents-lite.js"></script>

<link rel="import" href="https://raw.githubusercontent.com/PAIR-code/facets/1.0.0/facets-dist/facets-jupyter.html">

<facets-overview id="elem"></facets-overview>

<script>

document.querySelector("#elem").protoInput = "{protostr}";

</script>"""

html = HTML\_TEMPLATE.format(protostr=protostr)

display(HTML(html))

FairAware Task #1

Review the descriptive statistics and histograms for each numerical and continuous feature. Click the **Show Raw Data** button above the histograms for categorical features to see the distribution of values per category.

Then, try to answer the following questions from earlier:

Are there missing feature values for a large number of observations?

Are there features that are missing that might affect other features?

Are there any unexpected feature values?

What signs of data skew do you see?

Solution

Click below for some insights we uncovered.

We can see from reviewing the missing column that the following categorical features contain missing values:

workclass

occupation

Now, because it's only a small percentage of samples that contain either a missing workclass value or occupation value, we can safely drop those rows from the data set. If that percentage was much higher, then we would have to consider using a different data set that is more complete.

Luckily, in Pandas, there is a convenient way to drop any row containing a missing value in the data set:

# pandas.DataFrame.dropna(how="any", axis=0, inplace=True)

We will use this method prior to training the model when we convert a Pandas DataFrame to a Numpy array.

As for the remaining data that does not contain any missing values: if we look at the min/max values and histograms for each numeric feature, then we can pinpoint any extreme outliers in our data set.

For hours\_per\_week, we can see that the minimum is 1, which might be a bit surprising, given that most jobs typically require multiple hours of work per week. For capital\_gain and capital\_loss, we can see that over 90% of values are 0. Given that capital gains/losses are only registered by individuals who make investments, it's certainly plausible that less than 10% of examples would have nonzero values for these feature, but we may want to take a closer look to verify the values for these features are valid.

In looking at the histogram for gender, we see that over two-thirds (approximately 67%) of examples represent males. This strongly suggests data skew, as we would expect the breakdown between genders to be closer to 50/50.

A Deeper Dive

To futher explore the dataset, we can use [Facets Dive](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fpair-code.github.io%2Ffacets%2F), a tool that provides an interactive interface where each individual item in the visualization represents a data point. But to use Facets Dive, we need to convert the data to a JSON array. Thankfully the DataFrame method to\_json() takes care of this for us.

Run the cell below to perform the data transform to JSON and also load Facets Dive.

FairAware Task #2

Use the menus on the left panel of the visualization to change how the data is organized:

In the **Binning | X-Axis** menu, select **education**, and in the **Color By** and **Label By** menus, select **income\_bracket**. How would you describe the relationship between education level and income bracket?

Next, in the **Binning | X-Axis** menu, select **marital\_status**, and in the **Color By** and **Label By** menus, select **gender**. What noteworthy observations can you make about the gender distributions for each marital-status category?

As you perform the above tasks, keep the following fairness-related questions in mind:

**What's missing?**

**What's being overgeneralized?**

**What's being underrepresented?**

**How do the variables, and their values, reflect the real world?**

**What might we be leaving out?**

Solution

Click below for some insights we uncovered.

In the data set, higher education levels generally tend to correlate with a higher income bracket. An income level of greater than $50,000 is more heavily represented in examples where education level is Bachelor's degree or higher.

In most marital-status categories, the distribution of male vs. female values is close to 1:1. The one notable exception is "married-civ-spouse", where male outnumbers female by more than 5:1. Given that we already discovered in Task #1 that there is a disproportionately high representation of men in the data set, we can now infer that it's married women specifically that are underrepresented in the data.

Summary

Plotting histograms, ranking most-to-least common examples, identifying duplicate or missing examples, making sure the training and test sets are similar, computing feature quantiles—**these are all critical analyses to perform on your data.**

**The better you know what's going on in your data, the more insight you'll have as to where unfairness might creep in!**

FairAware Task #3

Now that you've explored the dataset using Facets, see if you can identify some of the problems that may arise with regard to fairness based on what you've learned about its features.

Which of the following features might pose a problem with regard to fairness?

Choose a feature from the drop-down options in the cell below, and then run the cell to check your answer. Then explore the rest of the options to get more insight about how each influences the model's predictions.

Predicting income using the Keras API

Now that we have a better sense of the Adult dataset, we can now begin with creating a neural network to predict income. In this section, as with previous exercises, we will be using TensorFlow's Keras API (specifically, tf.keras.Sequential) to construct our neural network model.

Convert Adult Dataset into Tensors

We first have to define our input fuction, which will take the Adult dataset that is in a pandas DataFrame and convert it a Numpy array.

While a pandas DataFrame is great — especially when working with Facets and other Python modules that visualize data — tf.keras.Sequential doesn't accept a pandas DataFrame as a data type. Luckily for us, it's quite trivial to convert a pandas DataFrame into a Numpy array, which is an accepted data type.

def pandas\_to\_numpy(data):

'''Convert a pandas DataFrame into a Numpy array'''

# Drop empty rows.

data = data.dropna(how="any", axis=0)

# Separate DataFrame into two Numpy arrays.

labels = np.array(data['income\_bracket'] == ">50K")

features = data.drop('income\_bracket', axis=1)

features = {name:np.array(value) for name, value in features.items()}

return features, labels

Represent Features in TensorFlow

TensorFlow requires that data maps to a model. To accomplish this, you have to use tf.feature\_columns to ingest and represent features in TensorFlow.

Create categorical feature columns

#@title Create categorical feature columns

# Since we don't know the full range of possible values with occupation and

# native\_country, we'll use categorical\_column\_with\_hash\_bucket() to help map

# each feature string into an integer ID.

occupation = tf.feature\_column.categorical\_column\_with\_hash\_bucket(

"occupation", hash\_bucket\_size=1000)

native\_country = tf.feature\_column.categorical\_column\_with\_hash\_bucket(

"native\_country", hash\_bucket\_size=1000)

# For the remaining categorical features, since we know what the possible values

# are, we can be more explicit and use categorical\_column\_with\_vocabulary\_list()

gender = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"gender", ["Female", "Male"])

race = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"race", [

"White", "Asian-Pac-Islander", "Amer-Indian-Eskimo", "Other", "Black"

])

education = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"education", [

"Bachelors", "HS-grad", "11th", "Masters", "9th",

"Some-college", "Assoc-acdm", "Assoc-voc", "7th-8th",

"Doctorate", "Prof-school", "5th-6th", "10th", "1st-4th",

"Preschool", "12th"

])

marital\_status = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"marital\_status", [

"Married-civ-spouse", "Divorced", "Married-spouse-absent",

"Never-married", "Separated", "Married-AF-spouse", "Widowed"

])

relationship = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"relationship", [

"Husband", "Not-in-family", "Wife", "Own-child", "Unmarried",

"Other-relative"

])

workclass = tf.feature\_column.categorical\_column\_with\_vocabulary\_list(

"workclass", [

"Self-emp-not-inc", "Private", "State-gov", "Federal-gov",

"Local-gov", "?", "Self-emp-inc", "Without-pay", "Never-worked"

])

Create numeric feature columns

#@title Create numeric feature columns

# For Numeric features, we can just call on feature\_column.numeric\_column()

# to use its raw value instead of having to create a map between value and ID.

age = tf.feature\_column.numeric\_column("age")

fnlwgt = tf.feature\_column.numeric\_column("fnlwgt")

education\_num = tf.feature\_column.numeric\_column("education\_num")

capital\_gain = tf.feature\_column.numeric\_column("capital\_gain")

capital\_loss = tf.feature\_column.numeric\_column("capital\_loss")

hours\_per\_week = tf.feature\_column.numeric\_column("hours\_per\_week")

Make Age a Categorical Feature

If you chose age when completing **FairAware Task #3**, you will have noticed that we suggested bucketing (also known as binning) this feature, grouping together similar ages into different groups. This might help the model generalize better across age. As such, we will convert age from a numeric feature (technically, an [ordinal feature](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FOrdinal_data)) to a categorical feature.

age\_buckets = tf.feature\_column.bucketized\_column(

age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])

Consider Key Subgroups

When performing feature engineering, it's important to keep in mind that you may be working with data drawn from individuals belonging to subgroups, for which you'll want to evaluate model performance separately.

**NOTE:** In this context, a subgroup is defined as a group of individuals who share a given characteristic—such as race, gender, or sexual orientation—that merits special consideration when evaluating a model with fairness in mind.

When we want our models to mitigate, or leverage, the learned signal of a characteristic pertaining to a subgroup, we will want to use different kinds of tools and techniques—**most of which are still actively being researched and developed**. You can find a list of related research work and techniques at our [Responsible AI Practices](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fai.google%2Fresponsibilities%2Fresponsible-ai-practices%2F%3Fcategory%3Dfairness) page.

As you work with different variables and define tasks for them, it can be useful to think about what comes next. For example, where are the places where the interaction of the variable and the task could be a concern?

Define the Model Features

Now we can explicitly define which feature we will include in our model.

We'll consider gender a subgroup and save it in a separate subgroup\_variables list, so we can add special handling for it as needed.

# List of variables, with special handling for gender subgroup.

variables = [native\_country, education, occupation, workclass,

relationship, age\_buckets]

subgroup\_variables = [gender]

feature\_columns = variables + subgroup\_variables

Train a Deep Neural Net Model on Adult Dataset

With the features now ready to go, we can try predicting income using deep learning.

For the sake of simplicity, we are going to keep the neural network architecture light by simply **defining a feed-forward neural network with two hidden layers**.

But first, we have to convert our high-dimensional categorical features into a low-dimensional and dense real-valued vector, which we call an embedding vector. Luckily, indicator\_column (think of it as one-hot encoding) and embedding\_column (that converts sparse features into dense features) helps us streamline the process.

Based on our analysis of the data set from previous FairAware Tasks, we are going to move forward with the following features:

workclass

education

age\_buckets

relationship

native\_country

occupation

All other features will be omitted from training — but you are welcome to experiment. gender is the only feature that will be used to filter the test set for subgroup evaluation purposes.

The following cell creates the deep columns required to define the input layer of the model:

CodeText

deep\_columns = [

tf.feature\_column.indicator\_column(workclass),

tf.feature\_column.indicator\_column(education),

tf.feature\_column.indicator\_column(age\_buckets),

tf.feature\_column.indicator\_column(relationship),

tf.feature\_column.embedding\_column(native\_country, dimension=8),

tf.feature\_column.embedding\_column(occupation, dimension=8),

]

With all the data preprocessing taken care of, we can now define and compile the deep neural net model. Start by using the parameters defined below. (Later on, after you've defined evaluation metrics and evaluated the model, you can come back and tweak these parameters to compare results.)

Define Deep Neural Net Model

#@title Define Deep Neural Net Model

# Parameters from form fill-ins

HIDDEN\_UNITS\_LAYER\_01 = 128 #@param

HIDDEN\_UNITS\_LAYER\_02 = 64 #@param

LEARNING\_RATE = 0.1 #@param

L1\_REGULARIZATION\_STRENGTH = 0.001 #@param

L2\_REGULARIZATION\_STRENGTH = 0.001 #@param

RANDOM\_SEED = 512

tf.random.set\_seed(RANDOM\_SEED)

# List of built-in metrics that we'll need to evaluate performance.

METRICS = [

tf.keras.metrics.TruePositives(name='tp'),

tf.keras.metrics.FalsePositives(name='fp'),

tf.keras.metrics.TrueNegatives(name='tn'),

tf.keras.metrics.FalseNegatives(name='fn'),

tf.keras.metrics.BinaryAccuracy(name='accuracy'),

tf.keras.metrics.Precision(name='precision'),

tf.keras.metrics.Recall(name='recall'),

tf.keras.metrics.AUC(name='auc'),

]

regularizer = tf.keras.regularizers.l1\_l2(

l1=L1\_REGULARIZATION\_STRENGTH, l2=L2\_REGULARIZATION\_STRENGTH)

model = tf.keras.Sequential([

layers.DenseFeatures(deep\_columns),

layers.Dense(

HIDDEN\_UNITS\_LAYER\_01, activation='relu', kernel\_regularizer=regularizer),

layers.Dense(

HIDDEN\_UNITS\_LAYER\_02, activation='relu', kernel\_regularizer=regularizer),

layers.Dense(

1, activation='sigmoid', kernel\_regularizer=regularizer)

])

model.compile(optimizer=tf.keras.optimizers.Adagrad(LEARNING\_RATE),

loss=tf.keras.losses.BinaryCrossentropy(),

metrics=METRICS)

To keep things simple, we'll pass through the full training data 10 times.

CodeText

Fit Deep Neural Net Model to the Adult Training Dataset

#@title Fit Deep Neural Net Model to the Adult Training Dataset

EPOCHS = 10 #@param

BATCH\_SIZE = 500 #@param

features, labels = pandas\_to\_numpy(train\_df)

model.fit(x=features, y=labels, epochs=EPOCHS, batch\_size=BATCH\_SIZE)

Evaluate Deep Neural Net Performance

#@title Evaluate Deep Neural Net Performance

features, labels = pandas\_to\_numpy(test\_df)

model.evaluate(x=features, y=labels);

You can try retraining the model using different parameters. If you leave the parameters as is, then you see that this relatively simple deep neural net does a decent job in predicting income with an **overall accuracy of 0.8317** and an **AUC of 0.8817**.

**But evaluation metrics with respect to subgroups are missing.** We will cover some of the ways you can evaluate at the subgroup level in the next section.

Evaluating for Fairness Using a Confusion Matrix

While evaluating the overall performance of the model gives us some insight into its quality, it doesn't give us much insight into how well our model performs for different subgroups.

When evaluating a model for fairness, it's important to determine whether prediction errors are uniform across subgroups or whether certain subgroups are more susceptible to certain prediction errors than others.

A key tool for comparing the prevalence of different types of model errors is a confusion matrix. Recall from the [Classification module of Machine Learning Crash Course](https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative) that a confusion matrix is a grid that plots predictions vs. ground truth for your model, and tabulates statistics summarizing how often your model made the correct prediction and how often it made the wrong prediction.

Let's start by creating a binary confusion matrix for our income-prediction model—binary because our label (income\_bracket) has only two possible values (<50K or >50K). We'll define an income of >50K as our **positive label**, and an income of <50k as our **negative label**.

**NOTE:** Positive and negative in this context should not be interpreted as value judgments (we are not suggesting that someone who earns more than 50k a year is a better person than someone who earns less than 50k). They are just standard terms used to distinguish between the two possible predictions the model can make.

Cases where the model makes the correct prediction (the prediction matches the ground truth) are classified as **true**, and cases where the model makes the wrong prediction are classified as **false**.

Our confusion matrix thus represents four possible states:

**true positive**: Model predicts >50K, and that is the ground truth.

**true negative**: Model predicts <50K, and that is the ground truth.

**false positive**: Model predicts >50K, and that contradicts reality.

**false negative**: Model predicts <50K, and that contradicts reality.

**NOTE:** If desired, we can use the number of outcomes in each of these states to calculate secondary evaluation metrics, such as [precision and recall](https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall).

Plot the Confusion Matrix

Since we've already defined which metrics we're interested in back when we defined and compiled our model, all we have to do now is:

Define a function that will help us visualize the heatmap.

Select which subgroup we're interested in, then pass that subgroup selection into tf.keras.Model.predict() for evaluation.

Define Function to Visualize Binary Confusion Matrix

#@title Define Function to Visualize Binary Confusion Matrix

def plot\_confusion\_matrix(

confusion\_matrix, class\_names, subgroup, figsize = (8,6)):

# We're taking our calculated binary confusion matrix that's already in the

# form of an array and turning it into a pandas DataFrame because it's a lot

# easier to work with a pandas DataFrame when visualizing a heat map in

# Seaborn.

df\_cm = pd.DataFrame(

confusion\_matrix, index=class\_names, columns=class\_names,

)

rcParams.update({

'font.family':'sans-serif',

'font.sans-serif':['Liberation Sans'],

})

sns.set\_context("notebook", font\_scale=1.25)

fig = plt.figure(figsize=figsize)

plt.title('Confusion Matrix for Performance Across ' + subgroup)

# Combine the instance (numercial value) with its description

strings = np.asarray([['True Positives', 'False Negatives'],

['False Positives', 'True Negatives']])

labels = (np.asarray(

["{0:g}\n{1}".format(value, string) for string, value in zip(

strings.flatten(), confusion\_matrix.flatten())])).reshape(2, 2)

heatmap = sns.heatmap(df\_cm, annot=labels, fmt="",

linewidths=2.0, cmap=sns.color\_palette("GnBu\_d"));

heatmap.yaxis.set\_ticklabels(

heatmap.yaxis.get\_ticklabels(), rotation=0, ha='right')

heatmap.xaxis.set\_ticklabels(

heatmap.xaxis.get\_ticklabels(), rotation=45, ha='right')

plt.ylabel('References')

plt.xlabel('Predictions')

return fig

Now that we have all the necessary functions defined, we can now compute the binary confusion matrix and evaluation metrics using the outcomes from [our deep neural net model](https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/intro_to_ml_fairness.ipynb?utm_source=mlcc&%3Butm_campaign=colab-external&%3Butm_medium=referral&%3Butm_content=fairness_tf2-colab&%3Bhl=+l10n-placeholder1&hl=pt-br#scrollTo=3nYSMg67jWaA). The output of this cell is a tabbed view, which allows us to toggle between the confusion matrix and evaluation metrics table.

**FairAware Task #4**

Use the form below to generate confusion matrices for the two gender subgroups: Female and Male. Compare the number of False Positives and False Negatives for each subgroup. Are there any significant disparities in error rates that suggest the model performs better for one subgroup than another?

**Visualize Binary Confusion Matrix and Compute Evaluation Metrics Per Subgroup**

#@title Visualize Binary Confusion Matrix and Compute Evaluation Metrics Per Subgroup

CATEGORY = "gender" #@param {type:"string"}

SUBGROUP = "Male" #@param {type:"string"}

# Labels for annotating axes in plot.

classes = ['Over $50K', 'Less than $50K']

# Given define subgroup, generate predictions and obtain its corresponding

# ground truth.

subgroup\_filter = test\_df.loc[test\_df[CATEGORY] == SUBGROUP]

features, labels = pandas\_to\_numpy(subgroup\_filter)

subgroup\_results = model.evaluate(x=features, y=labels, verbose=0)

confusion\_matrix = np.array([[subgroup\_results[1], subgroup\_results[4]],

[subgroup\_results[2], subgroup\_results[3]]])

subgroup\_performance\_metrics = {

'ACCURACY': subgroup\_results[5],

'PRECISION': subgroup\_results[6],

'RECALL': subgroup\_results[7],

'AUC': subgroup\_results[8]

}

performance\_df = pd.DataFrame(subgroup\_performance\_metrics, index=[SUBGROUP])

pd.options.display.float\_format = '{:,.4f}'.format

plot\_confusion\_matrix(confusion\_matrix, classes, SUBGROUP);

performance\_df

**Solution**

Click below for some insights we uncovered

Using default parameters, you may find that the model performs better for female than male. Specifically, in our run, we found that both accuracy and AUC for female (0.9137 and 0.9089, respectively) outperformed male (0.7923 and 0.8549, respectively). What is going on here?

Notice the number of true positives (top-left corner) for female is way lower compared to male (479 to 3822). Recall that in Task #1 we noticed a disproportionately high representation of male in the data set (almost 2-to-1). If you further explore the data set using Facets Dive in Task #2 by setting the color to income\_bracket and one of the axes to gender, then you will also find a disproportionately small number of female examples in the higher income bracket, our positive label.

What this is all suggesting is that the model is **overfitting, particuarly with respect to female and lower income bracket**. In other words, this model will not generalize well, particularly with female data, as it does not have enough positive examples for the model to learn from. It is **not doing that much better with male, either, as there is a disproportionately small number of high income bracket compared to low income bracket** — though not nearly as poorly represented as with female.

Hopefully going through this confusion matrix demonstration you find that the results varies slightly from the overall performance metrics, highlighting the importance of evaluating model performance across subgroup rather than in aggregate.

In your work, make sure that you make a good decision about the tradeoffs between false positives, false negatives, true positives, and true negatives. For example, you may want a very low false positive rate, but a high true positive rate. Or you may want a high precision, but a low recall is okay.

**Choose your evaluation metrics in light of these desired tradeoffs.**

Próximas etapas

## Continue seu curso de ML

Você já pode fazer o próximo curso de machine learning:

* [Preparação de dados e engenharia de atributos em ML](https://developers.google.com/machine-learning/data-prep/?utm_source=mlcc&utm_campaign=mlcc-next-steps&utm_medium=referral&utm_content=data-prep-ss&hl=pt-br)

Este curso mostra como preparar seus dados e criar atributos úteis, duas etapas essenciais para criar qualquer modelo de ML.

### Prática em machine learning

Para mais recursos, confira estes estudos de caso reais de como o Google usa machine learning nos produtos com exercícios práticos de codificação de vídeo e vídeo:

* [**Classificação de imagens**](https://developers.google.com/machine-learning/practica/image-classification?hl=pt-br): veja como o Google desenvolveu o modelo de classificação de imagens baseado em pesquisa no Google Fotos e depois crie seu próprio classificador de imagens.
* Mais aprendizado de máquina em breve

### TensorFlow

Para começar a usar o TensorFlow, primeiro [instale o TensorFlow](https://www.tensorflow.org/install/?hl=pt-br) e depois siga as etapas em [Primeiros passos com o TensorFlow](https://www.tensorflow.org/tutorials/?hl=pt-br).