Part I. The Fundamentals of Machine Learning

27 de setembro de 2024 18:52

O que é Machine Learning?

Machine Learning é a ciência (e arte) de programar computadores para que possam aprender a partir de *dados*.
Aqui está uma definição um pouco mais geral:

[Machine Learning é o] campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados.
—Arthur Samuel, 1959

E uma definição mais voltada para a engenharia:

*Diz-se que um programa de computador aprende a partir de uma experiência **E** em relação a uma tarefa **T** e uma medida de desempenho **P**, se o seu desempenho em **T**, medido por **P**, melhora com a experiência **E**.*
—Tom Mitchell, 1997

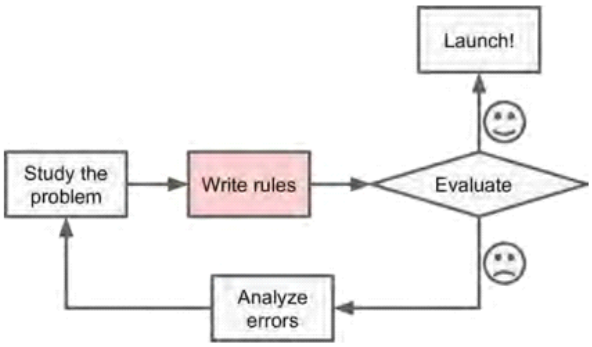
Por exemplo, o seu filtro de spam é um programa de Machine Learning que pode aprender a marcar spam com base em exemplos de e-mails de spam (por exemplo, marcados pelos usuários) e exemplos de e-mails regulares (não spam, também chamados de "ham"). Os exemplos que o sistema usa para aprender são chamados de conjunto de treinamento. Cada exemplo de treinamento é chamado de amostra de treinamento.

Nesse caso, a tarefa **T** é marcar como spam novos e-mails, a experiência **E** são os dados de treinamento, e a medida de desempenho **P** precisa ser definida; por exemplo, você pode usar a proporção de e-mails classificados corretamente. Essa medida de desempenho específica é chamada de ***acurácia***, e é frequentemente usada em tarefas de classificação.
Se você simplesmente baixar uma cópia da Wikipédia, o seu computador terá muito mais dados, mas isso não o tornará automaticamente melhor em qualquer tarefa. Portanto, isso não é Machine Learning.

Por que usar Machine Learning?

Considere que estamos criando um filtro de spam usando técnicas tradicionais de programação. Seguiríamos o diagrama abaixo:

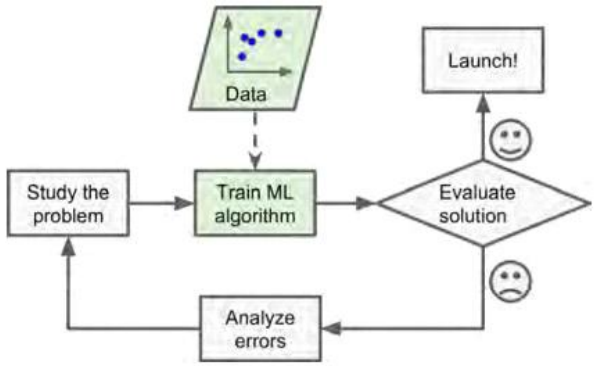
Approach tradicional:



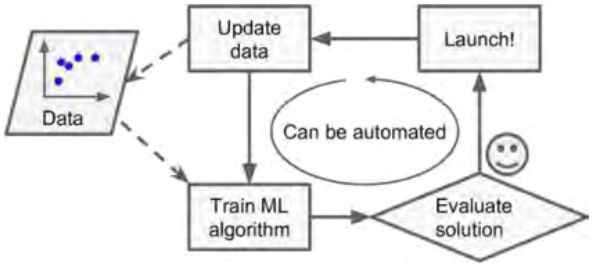
1. Study the problem:
   Primeiro, observaríamos como o spam geralmente se parece. Algumas palavras ou frases (como "4U", "credit card", "free" e "ama zing") tendem a aparecer com frequência no assunto do e-mail. Podem existir outros padrões no nome do remetente, no corpo do e-mail, e assim por diante.
2. Write rules:
   Escrever um algoritmo de detecção para cada um dos padrões que você notou, e o programa marcaria os e-mails como spam se um certo número desses padrões fosse detectado.
3. Evaluate:
   Testa-se o programa e em caso de erros, repete-se os passos 1 e 2 até que ele estivesse bom o suficiente.

Como o problema não é trivial, nosso programa provavelmente se tornará uma longa lista de regras complexas — o que é bastante difícil de manter. Em contraste, um filtro de spam baseado em técnicas de Machine Learning aprende automaticamente quais palavras e frases são bons preditores de spam, detectando padrões de palavras incomumente frequentes nos exemplos de spam em comparação com os exemplos de "ham". O programa é muito mais curto, fácil de manter e, muito provavelmente, mais preciso, como veremos a seguir.

Machine Learning Approach:



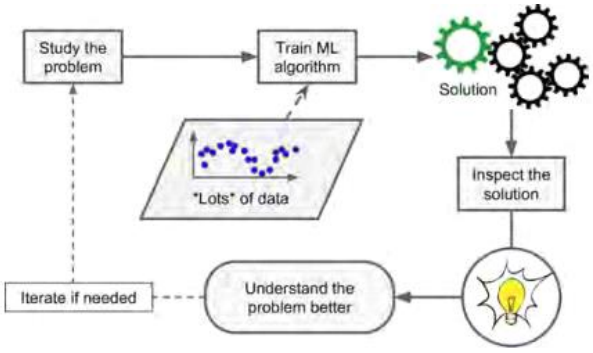
Além disso, se os spammers perceberem que todos os e-mails contendo "4U" estão sendo bloqueados, eles podem começar a escrever "For U" em vez disso. Um filtro de spam usando técnicas de programação tradicionais precisaria ser atualizado para marcar e-mails com "For U". Se os spammers continuarem a contornar seu filtro de spam, você teria que continuar escrevendo novas regras indefinidamente.
Em contraste, um filtro de spam baseado em técnicas de Machine Learning automaticamente percebe que "For U" se tornou incomumente frequente em e-mails de spam marcados pelos usuários, e começa a marcá-los sem precisar de nossa constante intervenção, como vemos na figura a seguir:



Note que o diagrama abaixo é um complemento do diagrama que apresentamos anteriormente para um exemplo de diagrama para um detector de spams usando Machine Learning, onde o conjunto de dados é atualizado constantemente, se adaptado a mudanças.

Outra área onde Machine Learning brilha é em problemas que são muito complexos para abordagens tradicionais ou que não possuem um algoritmo conhecido. Por exemplo, considere o reconhecimento de fala: digamos que você queira começar simples e escrever um programa capaz de distinguir as palavras “one” e “two.” Você poderia notar que a palavra “two” começa com um som agudo (“T”), então poderia codificar um algoritmo que mede a intensidade do som agudo e usá-lo para distinguir “ones” e “twos.” Obviamente, essa técnica não escalaria para milhares de palavras faladas por milhões de pessoas muito diferentes em ambientes barulhentos e em dezenas de idiomas. A melhor solução (pelo menos hoje) é escrever um algoritmo que aprende por conta própria, dado muitos gravações de exemplo para cada palavra.

Por fim, Machine Learning pode ajudar os humanos a aprender (como podemos ver na figura abaixo): algoritmos de ML podem ser inspecionados para ver o que aprenderam (embora para alguns algoritmos isso possa ser complicado). Por exemplo, uma vez que o filtro de spam tenha sido treinado com uma quantidade suficiente de spam, ele pode ser facilmente inspecionado para revelar a lista de palavras e combinações de palavras que acredita serem os melhores preditores de spam. Às vezes, isso revelará correlações inesperadas ou novas tendências, levando a uma melhor compreensão do problema. Aplicar técnicas de ML para explorar grandes quantidades de dados pode ajudar a descobrir padrões que não eram imediatamente aparentes. Isso é chamado de **data mining**.



Resumindo, Machine Learning é ótimo para:

- Problemas cujas soluções existentes exigem muitos ajustes manuais ou longas listas de regras: um único algoritmo de Machine L earning pode simplificar o código e apresentar melhor desempenho.
- Problemas complexos para os quais não há uma boa solução usando uma abordagem tradicional: as melhores técnicas de Machine Le arning podem encontrar uma solução.
- Ambientes em constante mudança: um sistema de Machine Learning pode se adaptar a novos dados.
- Obter insights sobre problemas complexos e grandes volumes de dados.

Tipos de Sistemas de Machine Learning

Existem tantos tipos diferentes de sistemas de Machine Learning que é útil classificá-los em categorias amplas com base em:

- Se eles são ou não treinados com supervisão humana (supervised, unsupervised, semisupervised e Reinforcement Learning)
- Se eles podem ou não aprender incrementalmente em tempo real (online versus batch learning)
- Se eles funcionam simplesmente comparando novos pontos de dados com pontos de dados conhecidos, ou se, em vez disso, detectam padrões nos dados de treinamento e constroem um modelo preditivo, como os cientistas fazem (instance-based versus model-based learning)

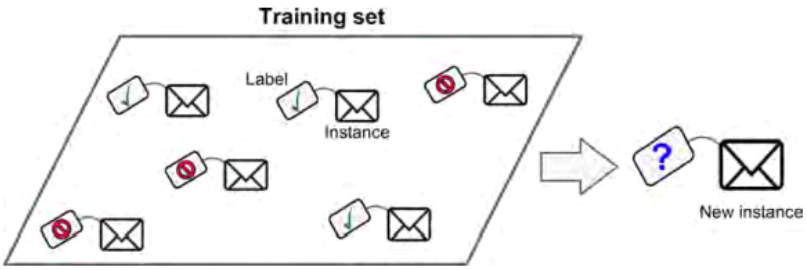
Esses critérios não são exclusivos; podemos combiná-los de várias maneiras. Por exemplo, um filtro de spam de última geração pode aprender em tempo real usando um modelo de rede neural profunda treinado com exemplos de spam e ham; isso o torna um sistema de aprendizado supervisionado, online e model-based.

Supervised/Unsupervised Learning

Os sistemas de Machine Learning podem ser classificados de acordo com a quantidade e o tipo de supervisão que recebem durante o treinamento. Existem quatro categorias principais: supervised learning, unsupervised learning, semisupervised learning e Reinforcement Learning.

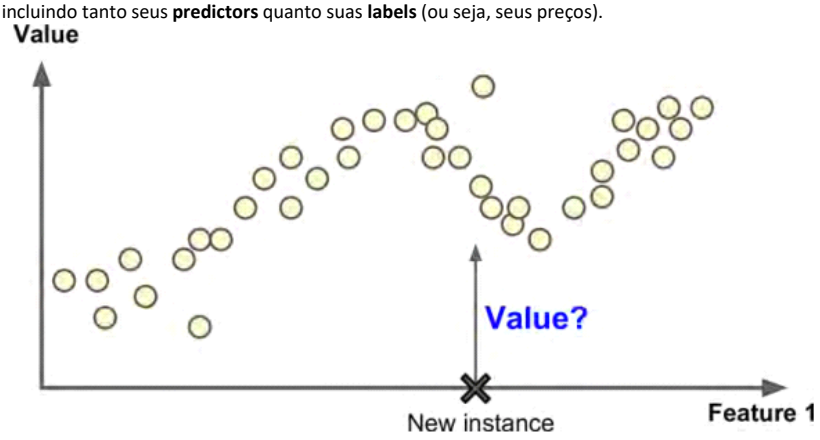
Supervised learning

Neste caso, os dados de treinamento que fornecemos ao algoritmo incluem as soluções desejadas, chamadas de **labels**, como na figura abaixo:



No caso do exemplo desta figura, as labels determinam se um e-mail é ou não um spam (ham). Uma tarefa comum de **supervised learning** é a classificação. O filtro de spam é um bom exemplo disso: ele é treinado com muitos e-mails de exemplo junto com suas classes (spam ou ham) e deve aprender a classificar novos e-mails.

Outra tarefa comum é prever um valor numérico-alvo, como o preço de um carro, dado um conjunto de características (quilometragem, idade, marca, etc.) chamadas de **predictors**. Esse tipo de tarefa é chamado de **regression** (como vemos na imagem abaixo). Para treinar o sistema, é necessário fornecer muitos exemplos de carros, incluindo tanto seus **predictors** quanto suas **labels** (ou seja, seus preços).



OBS: Em Machine Learning, um **atributo** é um tipo de dado (por exemplo, "Quilometragem"), enquanto uma **feature** tem vários significados dependendo do contexto, mas geralmente significa um atributo mais seu valor (por exemplo, "Quilometragem = 15.000"). Muitas pessoas usam as palavras atributo e feature de forma intercambiável, no entanto.

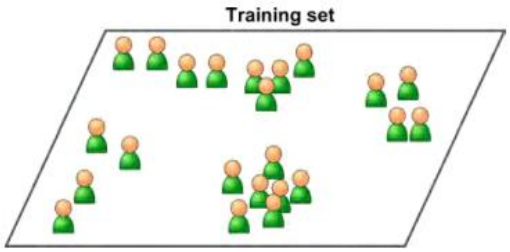
Observe que alguns algoritmos de **regression** também podem ser usados para **classification**, e vice-versa. Por exemplo, a **Logistic Regression** é comumente usada para classificação, pois pode gerar um valor que corresponde à probabilidade de pertencer a uma determinada classe (por exemplo, 20% de chance de ser spam).

Aqui estão alguns dos algoritmos de **supervised learning** mais importantes:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees e Random Forests
- Neural networks (Algumas arquiteturas de redes neurais podem ser **unsupervised**, como os **autoencoders**)

Unsupervised learning

No unsupervised learning, os dados de treinamento são não rotulados. O sistema tenta aprender sem um professor.
A figura abaixo mostra um conjunto de dados sem labels, usado em unsupervised learning.

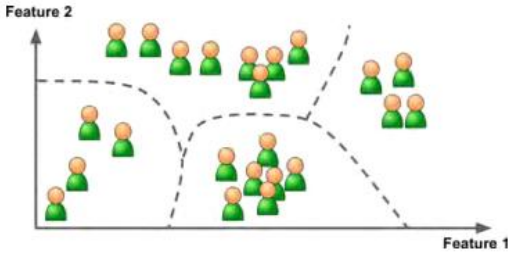


Aqui estão alguns dos algoritmos de **unsupervised learning** mais importantes:

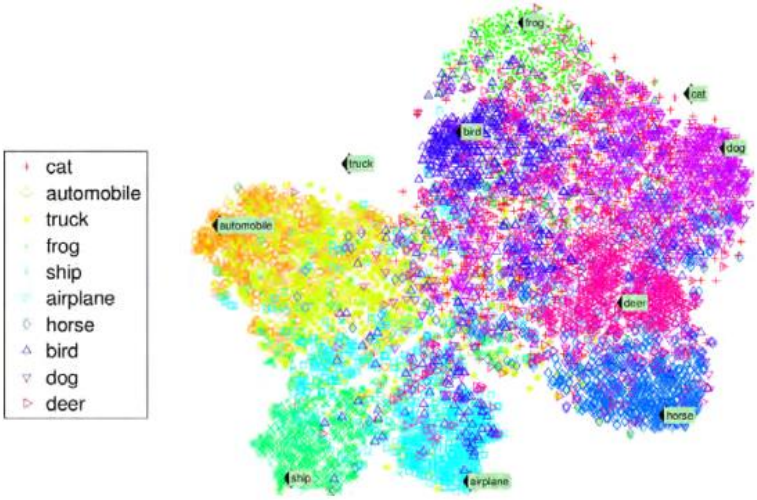
- **Clustering**
  - k-Means
  - Análise de Cluster Hierárquica (HCA)
  - Maximização da Expectativa (Expectation Maximization)
- **Visualização e redução de dimensionalidade**
  - Análise de Componentes Principais (PCA)
  - Kernel PCA
  - Embedding Linear Local (LLE)
  - Embedding Estocástico de Vizinhos Distribuídos t (t-SNE)
- **Aprendizado de regras de associação**
  - Apriori
  - Eclat

Por exemplo, digamos que você tenha muitos dados sobre os visitantes do seu blog. Você pode querer executar um algoritmo de *clustering* para tentar detectar grupos de visitantes semelhantes (como na figura abaixo). Em nenhum momento você informa ao algoritmo a qual grupo um visitante pertence: ele encontra essas

conexões sem a sua ajuda. Por exemplo, ele pode notar que 40% dos seus visitantes são homens que adoram histórias em quadrinhos e geralmente leem seu blog à noite, enquanto 20% são jovens amantes de ficção científica que visitam durante os finais de semana, e assim por diante. Se você usar um algoritmo de clustering hierárquico, ele também pode subdividir cada grupo em grupos menores. Isso pode ajudá-lo a direcionar suas postagens para cada grupo.



Os algoritmos de visualização também são bons exemplos de algoritmos de **unsupervised learning**: você os alimenta com muitos dados complexos e não rotulados, e eles produzem uma representação 2D ou 3D dos seus dados que pode ser facilmente plotada (como na figura abaixo). Esses algoritmos tentam preservar a maior parte da estrutura possível (por exemplo, tentando manter clusters separados no espaço de entrada sem sobreposição na visualização), para que você possa entender como os dados estão organizados e talvez identificar padrões inesperados.



Na figura, temos um exemplo de uma visualização t-SNE destacando clusters semânticos. Observe como os animais estão bastante separados dos veículos, como os cavalos estão próximos dos cervos, mas distantes das aves, e assim por diante. (t-SNE, que significa **t-distributed Stochastic Neighbor Embedding**, é uma técnica de redução de dimensionalidade amplamente utilizada para a visualização de dados de alta dimensão. O t-SNE transforma dados de alta dimensão em uma representação de baixa dimensão (geralmente 2D ou 3D), preservando a estrutura local dos dados.)

Uma tarefa relacionada é a redução de dimensionalidade, na qual o objetivo é simplificar os dados sem perder muita informação. Uma maneira de fazer isso é combinar várias features correlacionadas em uma só. Por exemplo, a quilometragem de um carro pode estar muito correlacionada com sua idade, então o algoritmo de redução de dimensionalidade irá fundi-las em uma feature que represente o desgaste do carro. Isso é chamado de extração de features.

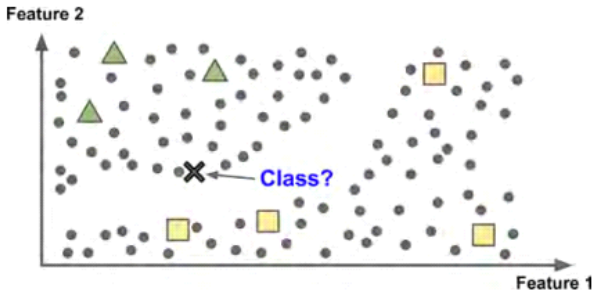
OBS: Frequentemente, é uma boa ideia tentar reduzir a dimensão dos seus dados de treinamento usando um algoritmo de redução de dimensionalidade antes de alimentá-los em outro algoritmo de Machine Learning (como um algoritmo de **supervised learning**). Isso fará com que o algoritmo execute muito mais rápido, os dados ocuparão menos espaço em disco e na memória e, em alguns casos, pode também apresentar um desempenho melhor.

Mais uma tarefa importante de **unsupervised learning** é a detecção de anomalias—por exemplo, detectar transações incomuns em cartões de crédito para prevenir fraudes, identificar defeitos na fabricação ou remover automaticamente outliers de um conjunto de dados antes de alimentá-lo em outro algoritmo de aprendizado. O sistema é treinado com instâncias normais e, ao ver uma nova instância, pode determinar se ela se parece com uma normal ou se é provavelmente uma anomalia, como vemos na figura abaixo:

Finalmente, outra tarefa comum de **unsupervised learning** é o aprendizado de regras de associação, cujo objetivo é explorar grandes quantidades de dados e descobrir relações interessantes entre atributos. Por exemplo, suponha que você possua um supermercado. Executar uma regra de associação em seus registros de vendas pode revelar que as pessoas que compram molho para churrasco e batatas fritas também tendem a comprar carne. Assim, você pode querer colocar esses itens próximos uns dos outros.

Semisupervised learning

Alguns algoritmos podem lidar com dados de treinamento parcialmente rotulados, geralmente uma grande quantidade de dados não rotulados e uma pequena quantidade de dados rotulados. Isso é chamado de **semisupervised learning**.

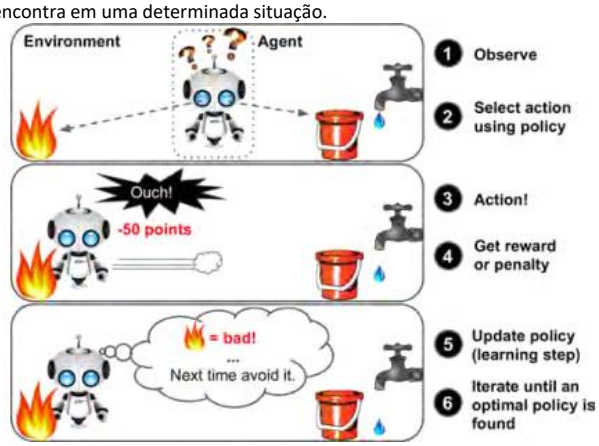


Alguns serviços de hospedagem de fotos, como o Google Fotos, são bons exemplos disso. Depois que você faz o upload de todas as suas fotos de família para o serviço, ele reconhece automaticamente que a mesma pessoa A aparece nas fotos 1, 5 e 11, enquanto outra pessoa B aparece nas fotos 2, 5 e 7. Esta é a parte não supervisionada do algoritmo (clustering). Agora, tudo o que o sistema precisa é que você diga quem são essas pessoas. Apenas uma etiqueta por pessoa, e ele consegue nomear todos em cada foto, o que é útil para buscar fotos.

A maioria dos algoritmos de **semisupervised learning** são combinações de algoritmos **unsupervised** e **supervised**. Por exemplo, as deep belief networks (DBNs) são baseadas em componentes não supervisionados chamados máquinas de Boltzmann restritas (RBMs) empilhadas uma sobre a outra. As RBMs são treinadas sequencialmente de maneira não supervisionada, e então todo o sistema é ajustado usando técnicas de **supervised learning**.

Reinforcement Learning

Reinforcement Learning é uma abordagem muito diferente. O sistema de aprendizado, chamado de agente neste contexto, pode observar o ambiente, selecionar e realizar ações e receber recompensas em troca (ou penalidades na forma de recompensas negativas, como na abaixo). Ele deve então aprender por conta própria qual é a melhor estratégia, chamada de política, para obter a maior recompensa ao longo do tempo. Uma política define qual ação o agente deve escolher quando se encontra em uma determinada situação.



Por exemplo, muitos robôs implementam algoritmos de **Reinforcement Learning** para aprender a andar. O programa AlphaGo da DeepMind também é um bom exemplo de **Reinforcement Learning**: ele fez manchetes em março de 2016 quando derrotou o campeão mundial Lee Sedol no jogo de Go. Ele aprendeu sua política vencedora analisando milhões de jogos e, em seguida, jogando muitos jogos contra si mesmo. Note que o aprendizado foi desativado durante os jogos contra o campeão; AlphaGo estava apenas aplicando a política que havia aprendido.

Batch and Online Learning

Outro critério usado para classificar sistemas de **Machine Learning** é se o sistema pode aprender incrementalmente a partir de um fluxo de dados que chegam.

Batch Learning

No **batch learning**, o sistema é incapaz de aprender de forma incremental: ele deve ser treinado usando todos os dados disponíveis. Isso geralmente leva muito tempo e recursos computacionais, por isso é tipicamente feito offline. Primeiro, o sistema é treinado e, em seguida, é colocado em produção e opera sem aprender mais; ele apenas aplica o que aprendeu. Isso é chamado de **offline learning**.

Se você quiser que um sistema de **batch learning** saiba sobre novos dados (como um novo tipo de spam), precisará treinar uma nova versão do sistema do zero com o conjunto de dados completo (não apenas os novos dados, mas também os dados antigos), depois parar o sistema antigo e substituí-lo pelo novo.

Felizmente, todo o processo de treinamento, avaliação e lançamento de um sistema de **Machine Learning** pode ser automatizado com bastante facilidade (como mostrado anteriormente), então até mesmo um sistema de **batch learning** pode se adaptar a mudanças. Basta atualizar os dados e treinar uma nova versão do sistema do zero sempre que necessário.

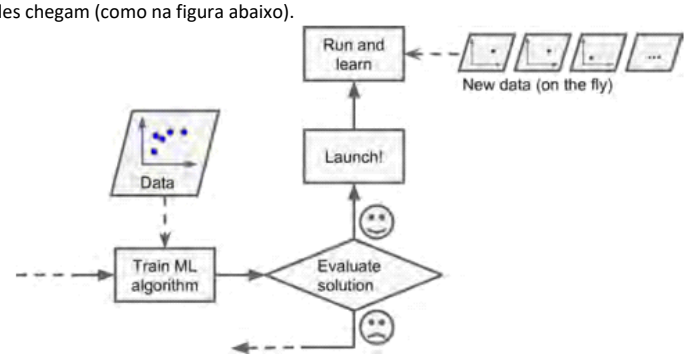
Essa solução é simples e geralmente funciona bem, mas o treinamento usando o conjunto completo de dados pode levar muitas horas, então você normalmente treinaria um novo sistema apenas a cada 24 horas ou até mesmo semanalmente. Se o seu sistema precisar se adaptar a dados que mudam rapidamente (por exemplo, para prever preços de ações), então você precisará de uma solução mais reativa.

Além disso, treinar com o conjunto completo de dados requer muitos recursos computacionais (CPU, espaço de memória, espaço em disco, I/O de disco, I/O de rede, etc.). Se você tiver muitos dados e automatizar seu sistema para treinar do zero todos os dias, isso acabará custando muito dinheiro. Se a quantidade de dados for enorme, pode até ser impossível usar um algoritmo de **batch learning**.
Finalmente, se o seu sistema precisar ser capaz de aprender de forma autônoma e tiver recursos limitados (por exemplo, um aplicativo para smartphone ou um rover em Marte), carregar grandes quantidades de dados de treinamento e consumir muitos recursos para treinar por horas todos os dias é um obstáculo.
Felizmente, uma opção melhor em todos esses casos é usar algoritmos que são capazes de aprender de forma incremental.

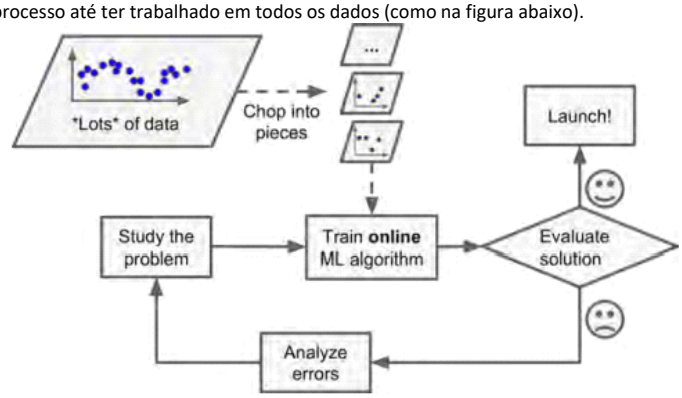
Online Learning

No **online learning**, você treina o sistema de forma incremental, alimentando-o com instâncias de dados sequencialmente, seja individualmente ou em pequenos grupos chamados de mini-batches. Cada etapa de aprendizado é rápida e barata, permitindo que o sistema aprenda sobre novos dados em tempo real, à medida que

eles chegam (como na figura abaixo).



O **online learning** é ótimo para sistemas que recebem dados como um fluxo contínuo (por exemplo, preços de ações) e precisam se adaptar rapidamente ou de forma autônoma. Também é uma boa opção se você tiver recursos computacionais limitados: uma vez que um sistema de **online learning** aprendeu sobre novas instâncias de dados, ele não precisa mais delas, permitindo que você as descarte (a menos que você queira ser capaz de reverter para um estado anterior e "reproduzir" os dados). Isso pode economizar uma quantidade enorme de espaço.
Os algoritmos de **online learning** também podem ser usados para treinar sistemas em conjuntos de dados enormes que não podem caber na memória principal de uma única máquina (isso é chamado de **out-of-core learning**). O algoritmo carrega parte dos dados, executa uma etapa de treinamento nesses dados e repete o processo até ter trabalhado em todos os dados (como na figura abaixo).



OBS: Esse processo geralmente é realizado offline (ou seja, não no sistema ativo), por isso o termo **online learning** pode ser confuso. Pense nele como aprendizado incremental.

Um parâmetro importante dos sistemas de **online learning** é a velocidade com que eles devem se adaptar a dados em mudança: isso é chamado de **learning rate**. Se você definir uma alta learning rate, seu sistema se adaptará rapidamente aos novos dados, mas também tenderá a esquecer rapidamente os dados antigos (você não quer que um filtro de spam classifique apenas os últimos tipos de spam que foi mostrado). Por outro lado, se você definir uma baixa taxa de aprendizado, o sistema terá mais inércia; ou seja, aprenderá mais lentamente, mas também será menos sensível ao ruído nos novos dados ou a sequências de pontos de dados não representativos.

Um grande desafio do **online learning** é que, se dados ruins forem alimentados ao sistema, o desempenho dele gradualmente diminuirá. Se estivermos falando de um sistema ativo, seus clientes perceberão. Por exemplo, dados ruins podem vir de um sensor com defeito em um robô ou de alguém que está fazendo spam em um motor de busca para tentar obter uma classificação alta nos resultados de pesquisa. Para reduzir esse risco, você precisa monitorar seu sistema de perto e desligar o aprendizado prontamente (e possivelmente reverter para um estado de funcionamento anterior) se detectar uma queda no desempenho. Você também pode querer monitorar os dados de entrada e reagir a dados anômalos (por exemplo, usando um algoritmo de detecção de anomalias).

O ChatGPT, como um modelo de linguagem, foi treinado usando **batch learning**. Isso significa que ele foi alimentado com grandes conjuntos de dados de texto em um processo de treinamento que ocorreu offline. Durante esse treinamento, o modelo aprendeu a partir de muitos exemplos simultaneamente, e uma vez concluído, ele não continua aprendendo com novas interações em tempo real.
Embora possa parecer que ele "aprende" a partir das conversas, isso não é o mesmo que online learning; o modelo não ajusta seus parâmetros com base em cada interação individual. Em vez disso, as interações são utilizadas para melhorar futuras versões do modelo, que são treinadas com novos dados de forma semelhante ao processo inicial.

Instance-Based Versus Model-Based Learning

Uma maneira adicional de categorizar sistemas de Machine Learning é pela forma como eles generalizam. A maioria das tarefas de Machine Learning envolve fazer previsões. Isso significa que, dado um número de exemplos de treinamento, o sistema precisa ser capaz de generalizar para exemplos que ele nunca viu antes. Ter uma boa medida de desempenho nos dados de treinamento é bom, mas insuficiente; o verdadeiro objetivo é ter um bom desempenho em novas instâncias.
Existem duas abordagens principais para a generalização: **instance-based learning** e **model-based learning**.

Instance-based learning

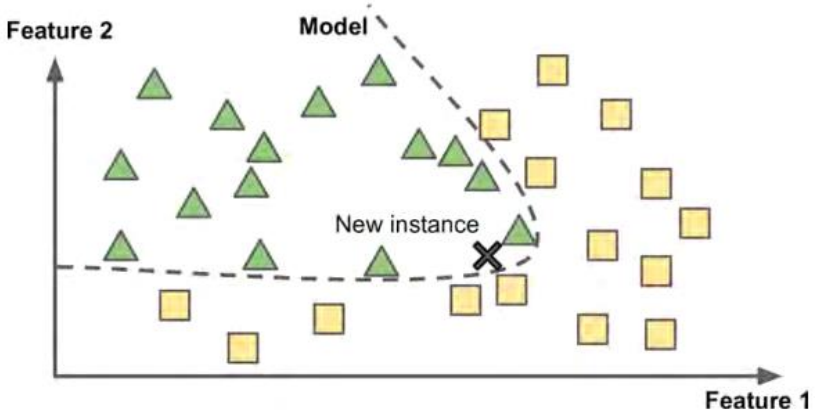
Possivelmente, a forma mais trivial de aprendizado é simplesmente decorar. Se você criasse um filtro de spam dessa maneira, ele apenas marcaria todos os e-mails que são idênticos aos e-mails que já foram marcados pelos usuários—não é a pior solução, mas certamente não é a melhor.

Em vez de apenas marcar e-mails que são idênticos a e-mails de spam conhecidos, seu filtro de spam poderia ser programado para também marcar e-mails que são muito semelhantes a e-mails de spam conhecidos. Isso requer uma medida de similaridade entre dois e-mails. Uma medida de similaridade (muito básica) entre dois e-mails poderia ser contar o número de palavras que eles têm em comum. O sistema marcaria um e-mail como spam se ele tiver muitas palavras em comum com um e-mail de spam conhecido.

Isso é chamado de **instance-based learning**: o sistema aprende os exemplos de cor e, em seguida, generaliza para novos casos usando uma medida de similaridade, como vemos na imagem abaixo:

Model-based learning

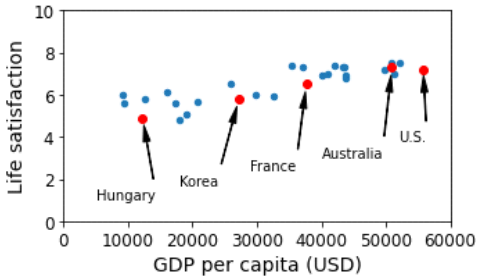
Outra maneira de generalizar a partir de um conjunto de exemplos é construir um modelo desses exemplos e, em seguida, usar esse modelo para fazer previsões. Isso é chamado de **model-based learning** (como na figura abaixo)



Por exemplo, suponha que você queira saber se o dinheiro faz as pessoas felizes, então você baixa os dados do **Better Life Index** do site da **OECD**, bem como estatísticas sobre o **PIB per capita** do site do **IMF**. Em seguida, você junta as tabelas e ordena pelo **PIB per capita**. Abaixo, temos uma parte destas tabelas com alguns poucos países aleatórios:

Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

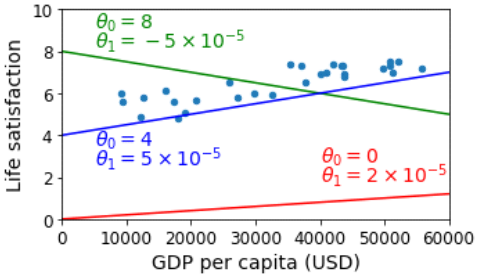
Agora, vamos plotar este gráfico destacando os países da tabela anterior:



Parece haver uma tendência. Embora os dados sejam ruidosos (ou seja, parcialmente aleatórios), parece que a satisfação com a vida aumenta de forma mais ou menos linear conforme o PIB per capita do país aumenta. Então, você decide modelar a satisfação com a vida como uma função linear do PIB per capita. Este passo é chamado de **seleção de modelo**: selecionamos um modelo linear de satisfação com a vida com apenas um atributo, o PIB per capita, que pode ser representado pela fórmula a seguir:

$$life\_satisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$$

Esse modelo tem dois parâmetros, θ_0 e θ_1. Ajustando esses parâmetros, você pode fazer com que seu modelo represente qualquer função linear (ax +b), como mostrado na figura abaixo:

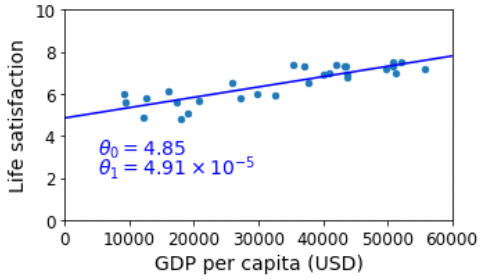


Dessa forma, é necessário definir os valores dos parâmetros θ_0 e θ_1. Como saber quais valores farão o modelo ter o melhor desempenho? Para responder a essa pergunta, você precisa especificar uma medida de desempenho. Pode-se definir uma *função utilidade* (ou *fitness function*) que mede o quão bom o modelo é, ou uma

função de custo que mede o quão ruim ele é. Para problemas de regressão linear, normalmente usa-se uma função de custo que mede a distância entre as previsões do modelo linear e os exemplos de treinamento; o objetivo é minimizar essa distância.

Aqui entra o algoritmo de **Regressão Linear**: você fornece os exemplos de treinamento e ele encontra os parâmetros que fazem o modelo linear se ajustar melhor aos seus dados. Esse processo é chamado de treinamento do modelo. No nosso caso, o algoritmo encontra os valores ótimos dos parâmetros como sendo θ_0 = 4,85 e θ_1 = $4{,}91 \times 10^{-5}$.

Agora, o modelo se ajusta aos dados de treinamento o mais próximo possível (para um modelo linear), como você pode ver na figura abaixo:



Exemplo 1-1:
Este exemplo consiste em criar um modelo capaz de fazer previsões. Queremos saber qual é a life satisfation dos cripriotas, e o dataset OECD que estamos usando não possui este dado, então vamos prevê-lo usando a GDP per capita do Chipre. Criaremos dois modelos em Python usando a biblioteca scikit-learn, um será de model-based e outro de instance-based. (Aqui faremos apenas as explicações teóricas, para visualizar o código deve acessar o arquivo 1-1.ipynb)

- **Model-based learning with linear regression**: Este foi o primeiro approach. Para construir um model-based learning, relembramos que este tipo de modelo detecta padrões no dataset de treino e, com base nesses padrões, constrói um **modelo preditivo**. Esse modelo preditivo é depois utilizado para fazer previsões sobre novos dados.

  Regressão linear é uma técnica estatística usada para **modelar a relação entre uma variável independente (ou mais) e uma variável dependente**. Ela assume que essa relação pode ser descrita por uma linha reta, no formato da equação: y=aX+b, onde:
  - y é a variável dependente (target, no caso a **life satisfaction**),
  - X é a variável independente (feature, no caso **GDP per capita**),
  - a é o coeficiente angular (a inclinação da linha),
  - b é o intercepto (o ponto onde a linha cruza o eixo y).

  Durante o treinamento, o modelo de regressão linear examina o **conjunto de dados de treino** (valores de GDP per capita e satisfação com a vida) e **encontra os padrões** ao calcular a melhor linha reta que se ajusta a esses dados. A "melhor linha" é aquela que minimiza o erro entre as previsões do modelo e os valores reais do target (essa técnica de ajuste é chamada de **mínimos quadrados**). A equação dessa linha se torna o **modelo preditivo**.
  No nosso caso, a regressão linear é a técnica utilizada para criar o **modelo preditivo**, e é isso que faz o processo ser **model-based**. Como vimos anteriormente, **Model-based** significa que o algoritmo **aprende uma fórmula ou um modelo** (no caso, a equação da linha reta) a partir dos dados de treino. Uma vez que o modelo é treinado, ele **não precisa armazenar todos os dados de treino**, apenas a equação do modelo (os coeficientes da linha) são necessários para fazer previsões em novos dados. Isso difere de **instance-based learning** (como KNN), que mantém todos os dados de treino e faz comparações diretas entre novas instâncias e os dados existentes, como veremos a seguir.

- **Instance-based learning with k-nearest neighbors regression**: Para construir um instance-based learning, relembramos que este tipo de modelo compara novos pontos com pontos já existentes no dataset.
  Neste caso, o novo ponto que iremos adicionar para efeitos de comparação será um ponto que representa o GDP per capita do Chi pre, e iremos comparar este ponto com pontos já existentes no dataset.
  Mas que pontos serão estes? Todos os pontos do dataset?
  Não, no nosso caso usaremos uma métrica para escolher os pontos do dataset que usaremos para esta comparação (ou quantos pont os serão usados nessa comparação). A métrica usada é chamada k-nearest neighbors (KNN). Para usar esta métrica, primeiro vamos calcular a **distância** entre o **GDP per capita do Chipre** e o **GDP per capita de outros países** no dataset. Em seguida, selecionar os **k vizinhos mais próximos** com base nessas distâncias (no exemplo, k=3, ou seja, os 3 países mais próximos em termos de GDP per capita). A previsão para o Chipre será a **média dos valores de satisfação com a vida** desses 3 vizinhos.

Se tudo correu bem, o modelo fará boas previsões. Caso contrário, será preciso usar mais atributos (taxa de emprego, saúde, poluição do ar, etc.), obter mais ou melhores dados de treinamento, ou talvez selecionar um modelo mais poderoso (como um modelo de Regressão Polinomial).
Em resumo:
- Estudamos os dados.
- Selecionamos um modelo.
- Treinamos com os dados de treinamento (ou seja, o algoritmo de aprendizado buscou os valores dos parâmetros do modelo que min imizam a função *loss*[1] ou função custo).
- Finalmente, aplicamos o modelo para fazer previsões em novos casos (isso é chamado de inferência), esperando que o modelo se generalize bem.

[1] Nos exemplos que fizemos até agora, qual é a função loss?

No Modelo Model-Based (Regressão Linear):

1. **Minimização da Função de Custo**:
   - No caso da **regressão linear**, estamos lidando com um modelo **model-based**, o que significa que o algoritmo ajusta os parâmetros (os coeficientes da reta) para **minimizar uma função de custo**.
   - A função de custo comum usada na regressão linear é a **Mean Squared Error (MSE)**, que mede a diferença entre os valores previstos pelo modelo e os valores reais.
   - O algoritmo de treinamento da regressão linear (como o método de **mínimos quadrados** ou o **gradiente descendente**) ajusta os **coeficientes** da linha reta (slope e intercepto) para minimizar o erro (O scikit-learn faz isso internamente e automaticamente "por baixo dos panos").
     Resumindo o que acontece:
   - O modelo tenta encontrar a melhor linha reta que **minimiza o erro total** (diferença ao quadrado entre as previsões e os valores reais) no conjunto de dados de treino.
   - Esse processo resulta em um modelo generalizado (a equação da reta) que pode ser usado para fazer previsões em novos dados.

No Modelo Instance-Based (K-Nearest Neighbors):

1. **Não Há Minimização Explícita de Função de Custo**:
   - No caso do **K-Nearest Neighbors (KNN)**, que é um exemplo de **instance-based learning**, **não há uma função de custo que está sendo minimizada**

diretamente. O KNN não "aprende" no sentido tradicional de ajustar parâmetros.

- Em vez disso, o KNN **armazena o conjunto de dados de treino** e faz previsões para novos pontos comparando-os com os exemplos do treino. O cálculo é feito em tempo real, usando uma métrica de distância (geralmente, a **distância euclidiana**) para encontrar os k vizinhos mais próximos.
- A previsão para um novo ponto (como no caso do GDP per capita do Chipre) é simplesmente a **média** dos valores target (neste caso, a satisfação com a vida) dos k vizinhos mais próximos.

Resumindo o que acontece:

- O KNN não ajusta nenhum parâmetro nem tenta criar um modelo generalizado. Ele **apenas armazena os dados** e faz previsões com base na comparação entre o novo ponto e os pontos de treino.
- Não há uma função de custo sendo minimizada; em vez disso, o foco é na **proximidade** entre os novos dados e os exemplos do conjunto de treino.

Principais Desafios do Machine Learning

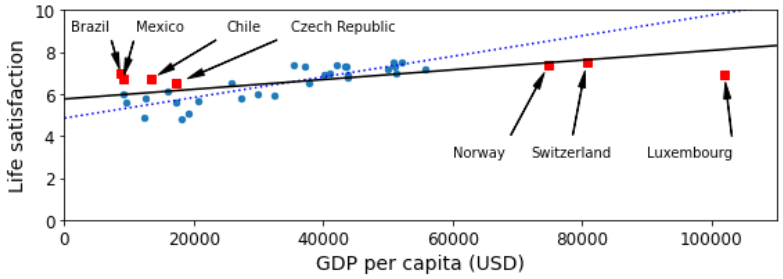
Em resumo, como nossa principal tarefa é selecionar um algoritmo de aprendizado e treiná-lo com dados, as duas coisas que podem dar errado são "algoritmo ruim" e "dados ruins". Vamos começar com exemplos de dados ruins.

Quantidade Insuficiente de Dados de Treinamento

Para uma criança aprender o que é uma maçã, basta você apontar para uma maçã e dizer "maçã" (possivelmente repetindo isso algumas vezes). Agora a criança consegue reconhecer maçãs de várias cores e formas. Genial.
O **Machine Learning** ainda não chegou a esse ponto; a maioria dos algoritmos de **Machine Learning** precisa de muitos dados para funcionar corretamente. Mesmo para problemas muito simples, geralmente é necessário ter milhares de exemplos, e para problemas complexos, como reconhecimento de imagens ou de fala, pode ser necessário ter milhões de exemplos (a menos que você consiga reutilizar partes de um modelo já existente).

Dados de Treinamento Não Representativos

Para generalizar bem, é crucial que os seus dados de treinamento sejam representativos dos novos casos aos quais você deseja generalizar. Isso é verdade tanto para **instance-based learning** quanto para **model-based learning**.
Por exemplo, o conjunto de países que usamos anteriormente para treinar o modelo linear não era perfeitamente representativo; alguns países estavam faltando. A figura abaixo mostra como os dados ficam quando adicionamos os países que estavam faltando:



Se treinarmos um modelo linear com esses novos dados, obteremos a linha sólida, enquanto o modelo antigo é representado pela linha pontilhada. Como podemos ver, adicionar apenas alguns países faltantes altera significativamente o modelo, além de deixar claro que um modelo linear tão simples provavelmente nunca funcionará bem. Parece que países muito ricos não são mais felizes do que países moderadamente ricos (na verdade, alguns parecem menos felizes), e, por outro lado, alguns países pobres parecem mais felizes do que muitos países ricos.
Ao usar um conjunto de treinamento não representativo, treinamos um modelo que provavelmente não fará previsões precisas, especialmente para países muito pobres ou muito ricos.

É crucial usar um conjunto de treinamento que seja representativo dos casos que você deseja generalizar. Isso muitas vezes é mais difícil do que parece: se a amostra for muito pequena, você terá **ruído amostral** (dados não representativos devido ao acaso); mas mesmo amostras muito grandes podem ser não representativas se o método de amostragem for falho. Isso é chamado de **viés de amostragem**.

Dados de Má Qualidade

Obviamente, se seus dados de treinamento estiverem cheios de erros, outliers e ruído (por exemplo, devido a medições de má qualidade), será mais difícil para o sistema detectar os padrões subjacentes, tornando o desempenho do sistema menos provável. Muitas vezes, vale a pena investir tempo na limpeza dos dados de treinamento. A verdade é que a maioria dos cientistas de dados passa uma parte significativa do seu tempo fazendo exatamente isso. Por exemplo:

- Se algumas instâncias forem claramente outliers, pode ser útil simplesmente descartá-las ou tentar corrigir os erros manualmente.
- Se algumas instâncias estiverem faltando alguns recursos (por exemplo, 5% de seus clientes não especificaram sua idade), você deve decidir se deseja ignorar completamente esse atributo, ignorar essas instâncias, preencher os valores ausentes (por exemplo, com a idade mediana) ou tr einar um modelo com esta feature e outro modelo sem ela, e assim por diante.

Features Irrelevantes

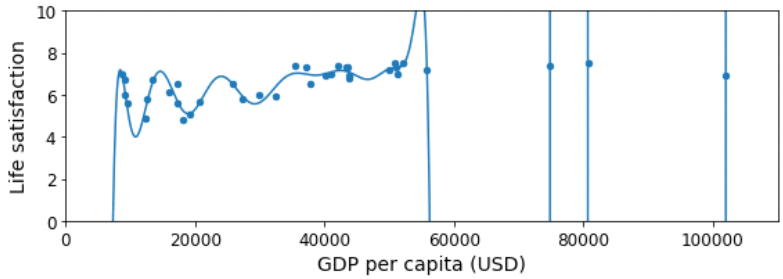
Como diz o ditado: lixo entra, lixo sai. Seu sistema só será capaz de aprender se os dados de treinamento contiverem features relevantes suficientes e não muitos irrelevantes. Uma parte crítica do sucesso de um projeto de Machine Learning é elaborar um bom conjunto de features para treinar. Esse processo, chamado de engenharia de features, envolve:

- **Seleção de Features**: selecionar as features mais úteis para treinar entre as features existentes.
- **Extração de Features**: combinar features existentes para produzir um feature mais útil (como vimos anteriormente, algoritmos de redução de dimensi onalidade podem ajudar).
- **Criação de Novos Features**: reunir novos dados para gerar novas features.

Agora, veremos exemplos de algoritmos ruins:

Overfitting dos Dados de Treinamento

Digamos que você esteja visitando um país estrangeiro e o taxista te engane. Você pode ser tentado a afirmar que todos os taxistas daquele país são ladrões. A supergeneralização é algo que nós, humanos, fazemos com muita frequência, e, infelizmente, as máquinas também podem cair nessa armadilha se não tivermos cuidado. Em Machine Learning, isso é chamado de **overfitting**: significa que o modelo apresenta um bom desempenho nos dados de treinamento, mas não generaliza bem.
A figura abaixo mostra um exemplo de um modelo polinomial de alta ordem para satisfação com a vida que superajusta fortemente os dados de treinamento. Embora ele tenha um desempenho muito melhor nos dados de treinamento do que o modelo linear simples, você realmente confiaria em suas previsões?



Modelos complexos, como deep neural networks, podem detectar padrões sutis nos dados, mas se o conjunto de treinamento estiver ruidoso ou se for muito pequeno (o que introduz ruído de amostragem), então é provável que o modelo detecte padrões no próprio ruído. Obviamente, esses padrões não se generalizarão para novas instâncias.
Por exemplo, se você alimentar seu modelo de satisfação com a vida com muitos mais atributos, incluindo aqueles não informativos, como o nome do país, um modelo complexo pode detectar padrões como o fato de que todos os países no conjunto de treinamento que têm uma letra "w" em seus nomes apresentam uma satisfação com a vida maior que 7: New Zealand (7,3), Norway (7,4), Sweden (7,2) e Switzerland (7,5). Quão confiante você está de que a regra da satisfação "W" se generaliza para Rwanda ou Zimbabwe? Obviamente, esse padrão ocorreu nos dados de treinamento por pura coincidência, mas o modelo não tem como saber se um padrão é real ou se é simplesmente o resultado do ruído nos dados.

O overfitting acontece quando o modelo é muito complexo em relação à quantidade e ao ruído dos dados de treinamento. As possíveis soluções são:

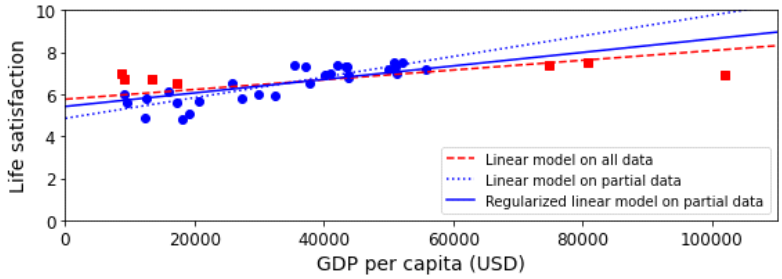
- **Simplificar o modelo**: Selecionando um modelo com menos parâmetros (por exemplo, um modelo linear em vez de um modelo polinomial de alto grau), reduzindo o número de atributos nos dados de treinamento ou restringindo o modelo.
- **Coletar mais dados de treinamento**.
- **Reduzir o ruído nos dados de treinamento**: Por exemplo, corrigindo erros nos dados e removendo outliers.

Restringir um modelo para torná-lo mais simples e reduzir o risco de overfitting é chamado de **regularização**. Por exemplo, o modelo linear que definimos anteriormente tem dois parâmetros, θ_0 e θ_1. Isso dá ao algoritmo de aprendizado dois graus de liberdade para adaptar o modelo aos dados de treinamento: ele pode ajustar tanto a altura (θ_0) quanto a inclinação (θ_1) da linha.

Se forçarmos θ_0 = 0, o algoritmo terá apenas um grau de liberdade e terá muito mais dificuldade para ajustar os dados corretamente: tudo o que ele poderia fazer seria mover a linha para cima ou para baixo para se aproximar o máximo possível das instâncias de treinamento, resultando em uma linha que fica em torno da média. Um modelo muito simples, de fato!

Se permitirmos que o algoritmo modifique θ_1, mas o obrigarmos a mantê-lo pequeno, o algoritmo de aprendizado terá efetivamente algo entre um e dois graus de liberdade. Isso produzirá um modelo mais simples do que com dois graus de liberdade, mas mais complexo do que apenas com um. O objetivo é encontrar o equilíbrio certo entre ajustar os dados perfeitamente e manter o modelo simples o suficiente para garantir que ele generalize bem.

A figura abaixo mostra três modelos: a linha pontilhada representa o modelo original que foi treinado com alguns países ausentes, a linha tracejada é nosso segundo modelo treinado com todos os países, e a linha sólida é um modelo linear treinado com os mesmos dados do primeiro modelo, mas com uma restrição de regularização. Você pode ver que a regularização forçou o modelo a ter uma inclinação menor, o que se ajusta um pouco menos aos dados de treinamento, mas, na verdade, permite que ele generalize melhor para novos exemplos.



A quantidade de regularização a ser aplicada durante o aprendizado pode ser controlada por um **hiperparâmetro**. Um hiperparâmetro é um parâmetro de um algoritmo de aprendizado (não do modelo). Como tal, ele não é afetado pelo próprio algoritmo de aprendizado; deve ser definido antes do treinamento e permanece constante durante o treinamento.
Se você definir o hiperparâmetro de regularização para um valor muito grande, obterá um modelo quase plano (uma inclinação próxima de zero); o algoritmo de aprendizado quase certamente não fará overfitting nos dados de treinamento, mas terá menos probabilidade de encontrar uma boa solução. Ajustar hiperparâmetros é uma parte importante da construção de um sistema de Machine Learning.

Underfitting the Training Data

Underfitting é o oposto de overfitting, e ocorre quando o modelo é simples demais para aprender a estrutura subjacente dos dados. Por exemplo, um modelo linear de satisfação de vida tende a sofrer de underfitting; a realidade é mais complexa do que o modelo, portanto, suas previsões provavelmente serão imprecisas, até mesmo nos exemplos de treinamento.
As principais opções para corrigir esse problema são:

- Selecionar um modelo mais poderoso, com mais parâmetros
- Fornecer melhores características para o algoritmo de aprendizado (**feature engineering**)
- Reduzir as restrições no modelo (por exemplo, diminuir o hiperparâmetro de regularização)

Resumindo...

Até agora, já vimos que:

- **Machine Learning** é sobre fazer máquinas melhorarem em uma tarefa a partir de dados, sem a necessidade de codificar regras explicitamente.
- Existem muitos tipos de sistemas de **Machine Learning**: supervisionados ou não, **batch** ou **online**, **instance-based** ou **model-based**, e assim por diante.
- Em um projeto de **Machine Learning**, você coleta dados em um conjunto de treinamento e os alimenta em um algoritmo de aprendizado. Se o algoritmo for **model-based**, ele ajusta alguns parâmetros para adequar o modelo ao conjunto de treinamento (ou seja, para fazer boas previsões sobre o p róprio conjunto de treinamento) e, com sorte, ele também será capaz de fazer boas previsões em novos casos. Se o algoritmo for **instance-based**, ele memoriza os exemplos e utiliza uma medida de similaridade para generalizar para novos casos.
- O sistema não terá um bom desempenho se o conjunto de treinamento for muito pequeno, se os dados não forem representativos, s e forem ruidosos ou poluídos com características irrelevantes (garbage in, garbage out). Por fim, seu modelo não deve ser nem muito simples (caso contrário, ele sofrerá

underfitting), nem muito complexo (caso contrário, ele sofrerá **overfitting**).

Há apenas mais um tópico importante a cobrir: uma vez que você tenha treinado um modelo, você não deve simplesmente "esperar" que ele generalize para novos casos. É preciso avaliá-lo e ajustá-lo, se necessário.

Testando e Validando (Cross-Validation)

A única maneira de saber o quão bem um modelo irá generalizar para novos casos é, de fato, testá-lo em novos casos. Uma forma de fazer isso é colocá-lo em produção e monitorar seu desempenho. Isso pode funcionar bem, mas se o modelo for ruim, os seus usuários irão reclamar—não é a melhor ideia.
Uma opção melhor é dividir seus dados em dois conjuntos: **conjunto de treinamento** e **conjunto de teste**. Como os nomes indicam, você treina o modelo usando o conjunto de treinamento e o testa usando o conjunto de teste. A taxa de erro em novos casos é chamada de **erro de generalização** (ou **erro fora da amostra**), e avaliando o modelo no conjunto de teste, obtemos uma estimativa desse erro. Esse valor indica o quão bem o modelo funcionará em instâncias que ele nunca viu antes.
Se o erro de treinamento for baixo (ou seja, seu modelo comete poucos erros no conjunto de treinamento), mas o **erro de generalização** for alto, isso significa que seu modelo está sofrendo de **overfitting** no conjunto de treinamento.

Dessa forma, avaliar um modelo é simples: basta usar um **conjunto de teste**.
Agora, suponha que você esteja em dúvida entre dois modelos (por exemplo, um modelo linear e um modelo polinomial): como decidir? Uma opção é treinar ambos e comparar como eles se generalizam usando o conjunto de teste.

Agora, suponha que o modelo linear se generalize melhor, mas você queira aplicar alguma **regularização** para evitar o **overfitting**. A questão é: como escolher o valor do **hiperparâmetro** de regularização? Uma opção seria treinar 100 modelos diferentes usando 100 valores diferentes para esse hiperparâmetro.

Suponha que você encontre o melhor valor de hiperparâmetro, que produz um modelo com o menor erro de generalização, digamos 5% de erro. Então, você lança esse modelo em produção, mas, infelizmente, ele não performa tão bem quanto o esperado e gera 15% de erros. O que aconteceu?
O problema é que você mediu o erro de generalização várias vezes no **conjunto de teste** e ajustou o modelo e os hiperparâmetros para produzir o melhor resultado para esse conjunto. Isso significa que o modelo provavelmente não irá se sair tão bem com novos dados.

Uma solução comum para esse problema é ter um segundo conjunto reservado chamado **validation set**. Você treina vários modelos com diferentes hiperparâmetros usando o conjunto de treinamento, seleciona o modelo e os hiperparâmetros que têm o melhor desempenho no conjunto de validação, e quando estiver satisfeito com o modelo, faz um único teste final contra o **conjunto de teste** para obter uma estimativa do erro de generalização.
Para evitar "desperdiçar" muitos dados de treinamento em conjuntos de validação, uma técnica comum é usar a **Cross-validation**. O conjunto de treinamento é dividido em subconjuntos complementares, e cada modelo é treinado contra uma combinação diferente desses subconjuntos e validado contra as partes restantes. Depois que o tipo de modelo e os hiperparâmetros forem selecionados, um modelo final é treinado usando esses hiperparâmetros no conjunto completo de treinamento, e o erro de generalização é medido no conjunto de teste.

É comum usar 80% dos dados para **treinamento** e reservar 20% para **teste**. Essa divisão garante que a maior parte dos dados seja utilizada para treinar o modelo, enquanto uma parte menor é usada para avaliar sua capacidade de generalização em novos dados.

Machine Learning Q&A

1. **Como você definiria Machine Learning?**

Machine Learning é uma área da inteligência artificial que foca em permitir aos computadores aprenderem com dados, melhorando sua performance em uma tarefa específica sem serem explicitamente programados para executar aquela tarefa. Ao invés de seguir uma lista de tarefas predeterminada, um modelo de Machine Learning identifica padrões em dados e usa estes padrões para tomar decisões ou fazer previsões em novos (e não vistos) dados.

2. **Você pode nomear quatro tipos de problemas nos quais ele se destaca?**

Basicamente, modelos de machine learning são úteis em problemas complexos onde não é possível escrever um algoritmo de que o solucione, ou para substituir algoritmos onde seriam necessárias muitas instruções, ou também para lidar com ambientes que estão em constante mudança. Como por exemplo, reconhecimento de imagens, tradução, previsões do tempo, recomendação de música no Spotify.

3. **O que é um conjunto de treinamento rotulado?**

Um conjunto de treinamento rotulado é usado em supervised learning. Estas labels estão acompanhadas de uma instância, e servem para caracterizar esta instância. Dessa forma, o conjunto de treinamento rotulado possui a "resposta" para cada instância.

4. **Quais são as tarefas de supervised learning mais comuns?**

Caracterização e regressão.

5. **Nomeie cinco tarefas unsupervised.**

Clustering, visualização, redução de dimensionalidade e regra da associação.

6. **Que tipo de algoritmo em ML você usaria para permitir a um robô a andar em vários terrenos desconhecidos?**

Aprendizado por repetição (reinforcement learning).

7. **Qual algoritmo de ML você usaria para separar seus clientes em múltiplos grupos?**

Se não fazemos ideia de quem são os clientes, usaríamos um Algoritmo de clustering, que é um algoritmo de unsupervised learning. Este algoritmo os dividiria baseado em semelhanças. No caso de sabermos quem são estes clientes, estes poderiam ter labels e poderíamos usar um algoritmo de classificação em um supervised learning.

8. **O problema de detecção de spams é um problema unsupervised ou supervised?**

É um problema de supervised learning. Pois os e-mails estão caracterizados em spams e não spams (ham).

9. **O que é um sistema de aprendizado online (online learning system)?**

Este sistema vai adicionando novos dados ao dataset de forma online e aprendendo com eles incrementalmente.

10. **O que é out-of-core learning?**

Algoritmos out-of-core conseguem lidar com uma vasta quantidade de dados, dados estes que não caberiam na memória de um computador. Estes algoritmos dividem a data em mini-batches e usa online learning para aprender com estes mini-batches.

11. **Qual algoritmo de ML usa uma medida de similaridade para fazer previsões?**

Um sistema de instance-based learning. Lembramos que estes sistemas decoram o dataset, e usam alguma métrica de similaridade para fazer previsões.

12. Qual é a diferença entre um parâmetro do modelo e de um hiperparâmetro do algoritmo de aprendizado?
Um modelo tem um ou mais parâmetros de modelo que determinam o que ele irá prever dada uma nova instância (por exemplo, a inclinação de um modelo linear). Como um hiperparâmetro é um parâmetro de um algoritmo de aprendizado, e não do modelo em si, ele não é afetado pelo próprio algoritmo de aprendizado. Este deve ser definido antes do treinamento e permanece constante durante o treinamento.

13. O que algoritmos model-based learning buscam? Qual a sua estratégia principal? Como eles fazem previsões?
Buscam um valor ótimo para os parâmetros do modelo afim de que o modelo generalize bem em novas instâncias. Normalmente, treinamos esses sistemas minimizando uma função de custo que mede quão ruim o sistema está em fazer previsões com os dados de treinamento, além de uma penalização pela complexidade do modelo, se o modelo estiver regularizado. Para fazer previsões, alimentamos as características da nova instância na função de previsão do modelo, usando os valores de parâmetros encontrados pelo algoritmo de aprendizado.

14. Quais são os desafios principais em Machine Learning?
Falta de dados, e quando há dados, a baixa qualidade destes dados e quando não são representativos. A dificuldade também pode estar no ajuste do modelo, quando usamos modelos muito simples e que não se ajustam ao conjunto de dados, ou quando usamos modelos complexos demais e que se ajustam aos dados até demais.

15. **Se um modelo está performando bem no dataset de treino, mas generaliza mal em novas instâncias, o que pode estar acontecendo? Nomeie três soluções possíveis para este problema**.

Isto pode estar acontecendo pois o modelo está sofrendo de overfitting. Para solucionar isso, podemos aumentar o conjunto de dados, simplificar o modelo ou reduzir o número de ruído nos dados de treino.

16. O que é um conjunto de treino e por que usá-lo?
Dividir o dataset em dataset de treino e dataset de teste nos ajuda a perceber como o modelo performará em novas instâncias, pois conseguimos estimar um erro de generalização. Isto permite verificar a eficácia do modelo em dados nunca antes vistos por ele.

17. Qual a finalidade do conjunto de validação?
Um conjunto de validação é utilizado para comparar modelos. Ele permite selecionar o melhor modelo e ajustar os hiperparâmetros. Ao usar um conjunto de validação, você pode avaliar o desempenho de diferentes modelos e configurações, garantindo que a escolha final seja a mais adequada antes de testar o modelo em um conjunto de teste.

18. O que pode dar errado se escolhermos os hiperparâmetros baseado em testes no conjunto de teste?
Neste caso, os hiperparâmetros escolhidos são os que performaram melhor no conjunto de teste (i.e., causaremos um overfitting ao conjunto de teste), e não significa que performarão bem em novas instâncias.

19. O que é cross-validation e por que preferimos usar este método a usar um conjunto de validação?
A técnica de cross-validation consiste em dividir o dataset de treino em subsets, e treinamos o modelo várias vezes usando combinações destes subsets, e validando contra as partes restantes.
Esta técnica acaba por ser melhor do que criar um conjunto de validação pois evitamos o desperdícios de dados preciosos em um conjunto de validação.