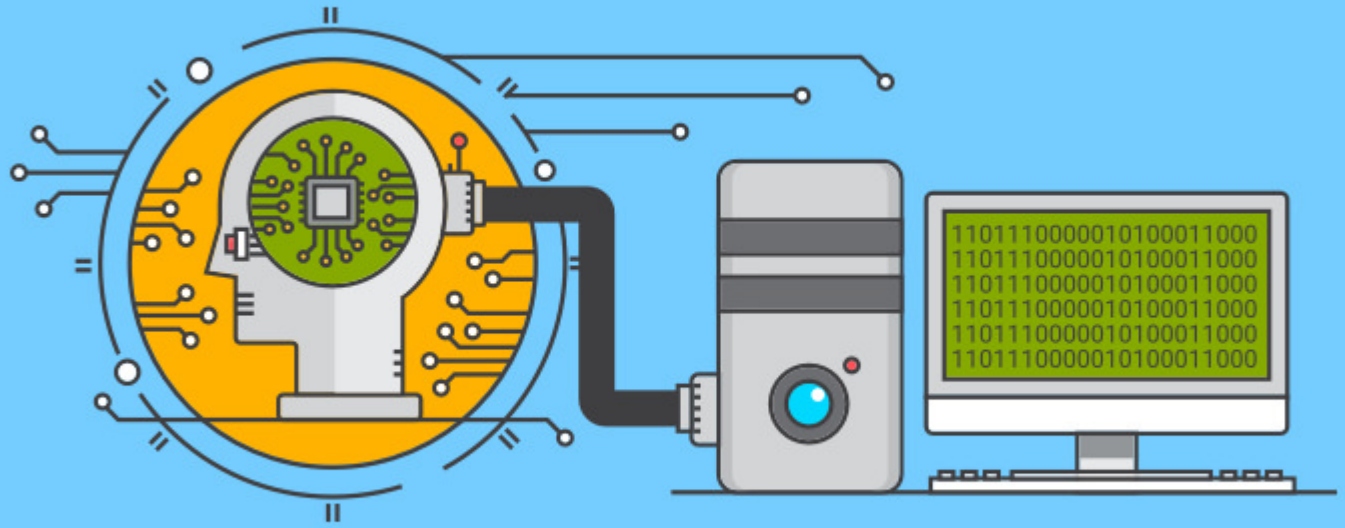
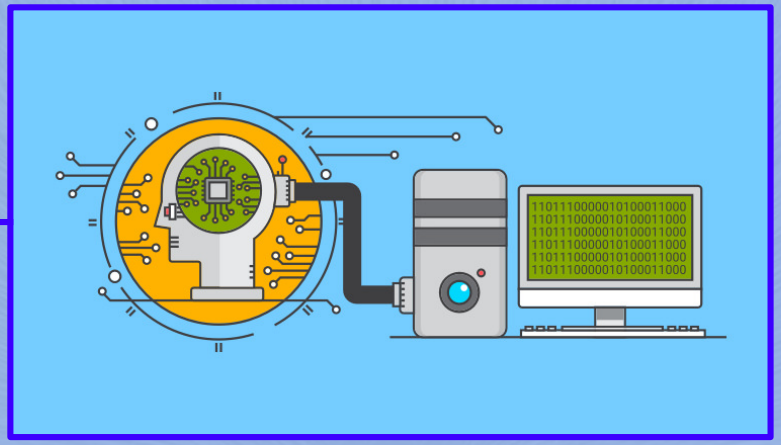
Machine Learning





Autor: Gabriel Felippe

Website: akiradev.netlify.app

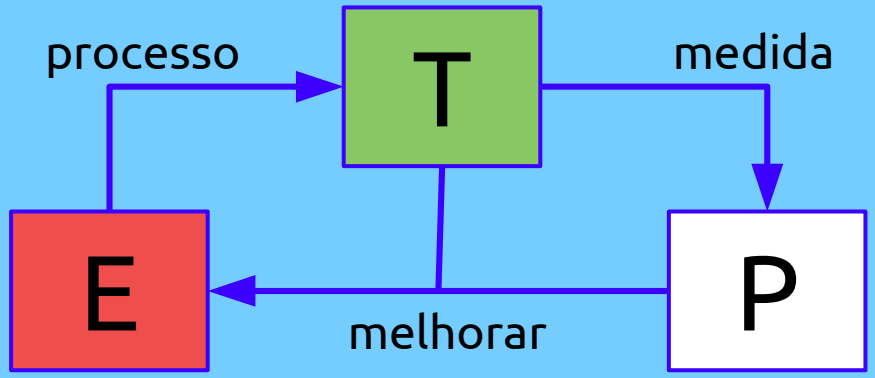
O que é Machine Learning?

- Machine Learning é a ciência (e a arte) de programar computadores para que eles possam aprender com os dados. (Aurélien Geron)

- Machine Learning é o campo de estudo que dá aos computadores a capacidade de aprender sem serem programados explicitamente. (Arthur Samuel, 1959)

O que é Machine Learning?

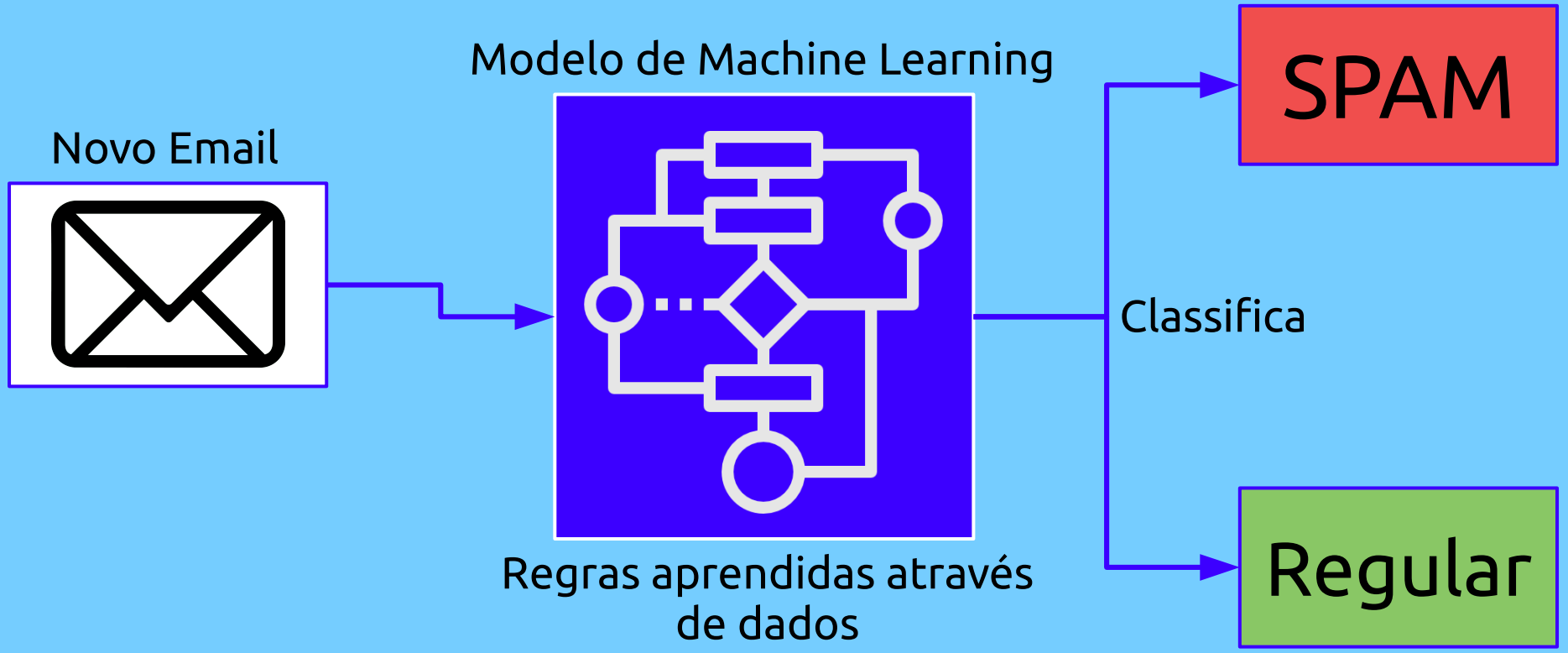
- Um computador é dito aprender através de uma experiência E em respeito a uma tarefa T e uma medida de perfomance P, se sua perfomance em T, medida por P, melhorar com a experiência E. (Tom Mitchell, 1997)



Filtro de Spam

- Seu filtro de spam é um programa de Machine Learning que, oferecidos exemplos de e-mails de spam (por exemplo, sinalizado por usuários) e exemplos de e-mails regulares (não spam), pode aprender a sinalizar spam.
- Os exemplos que o sistema usa para aprender são chamados de conjunto de treinamento. Cada exemplo de treinamento é chamado de instância de treinamento (ou amostra).
- Nesse caso, a tarefa T é sinalizar spam para novos emails, a experiência E são os dados de treinamento e a medida de perfomance P precisa ser definida; por exemplo, você pode usar a proporção de e-mails classificados corretamente. Essa medida de desempenho específica é chamada de accuracy e costuma ser usada em tarefas de classificação.

Filtro de Spam



Por que Usar Machine Learning?

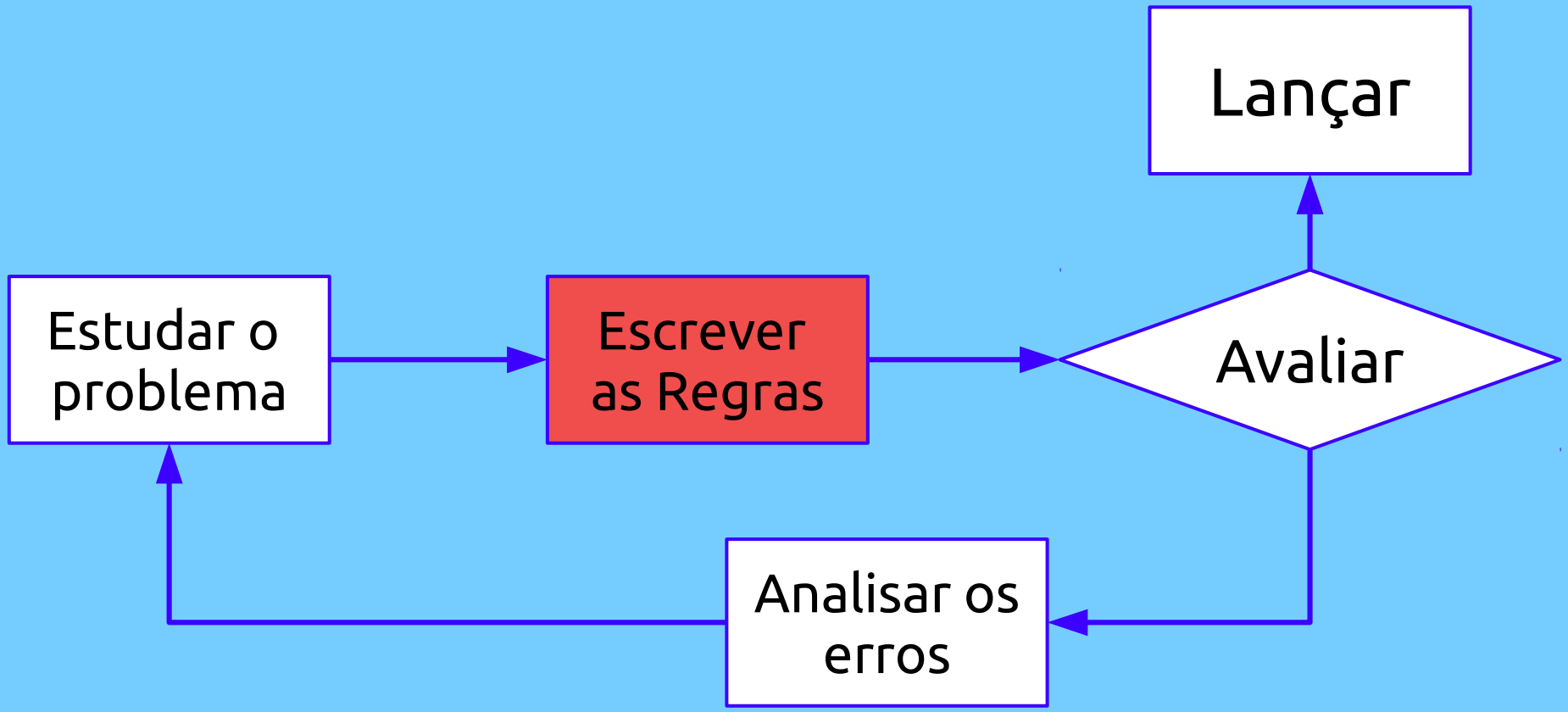
- Considere como você escreveria um filtro de spam usando técnicas de programação tradicionais:

  1. Primeiro, você deve considerar a aparência típica do spam. Você pode notar que algumas palavras ou frases (como “cartão de crédito”, “grátis” e “fácil”) tendem a aparecer muito na linha de assunto. Talvez você também perceba alguns outros padrões no nome do remetente, no corpo do e-mail e em outras partes do e-mail.

  2. Você escreveria um algoritmo de detecção para cada um dos padrões observados e seu programa sinalizaria e-mails como spam se vários desses padrões fossem detectados.

  3. Você testaria seu programa e repetiria as etapas 1 e 2 até que estivesse bom o suficiente para iniciar.

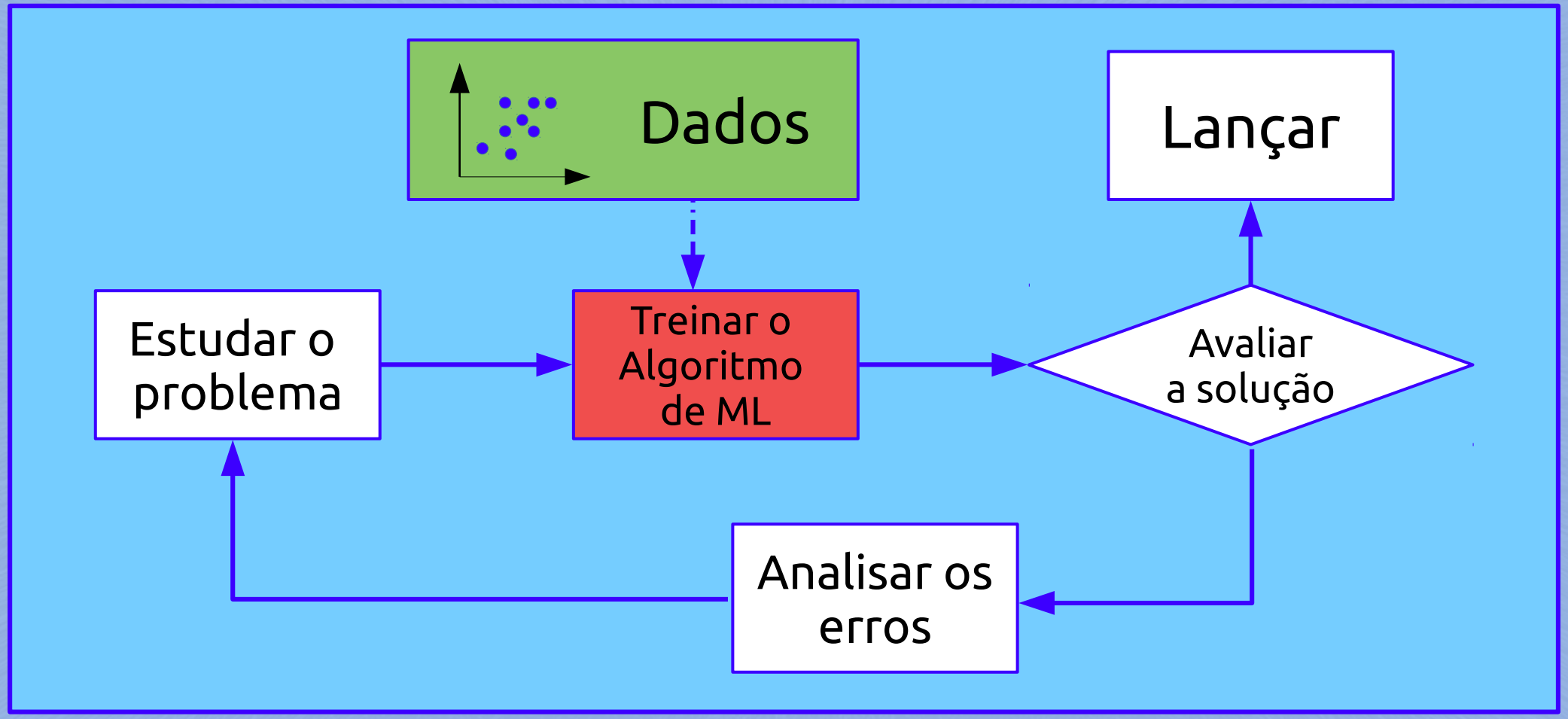
Abordagem Tradicional



Dificuldade do Problema

- Como o problema do filtro de spam é difícil, seu programa provavelmente se tornará uma longa lista de regras complexas - muito difíceis 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 regulares.
- O programa é muito mais curto, mais fácil de manter e provavelmente mais preciso.

Abordagem de Machine Learning



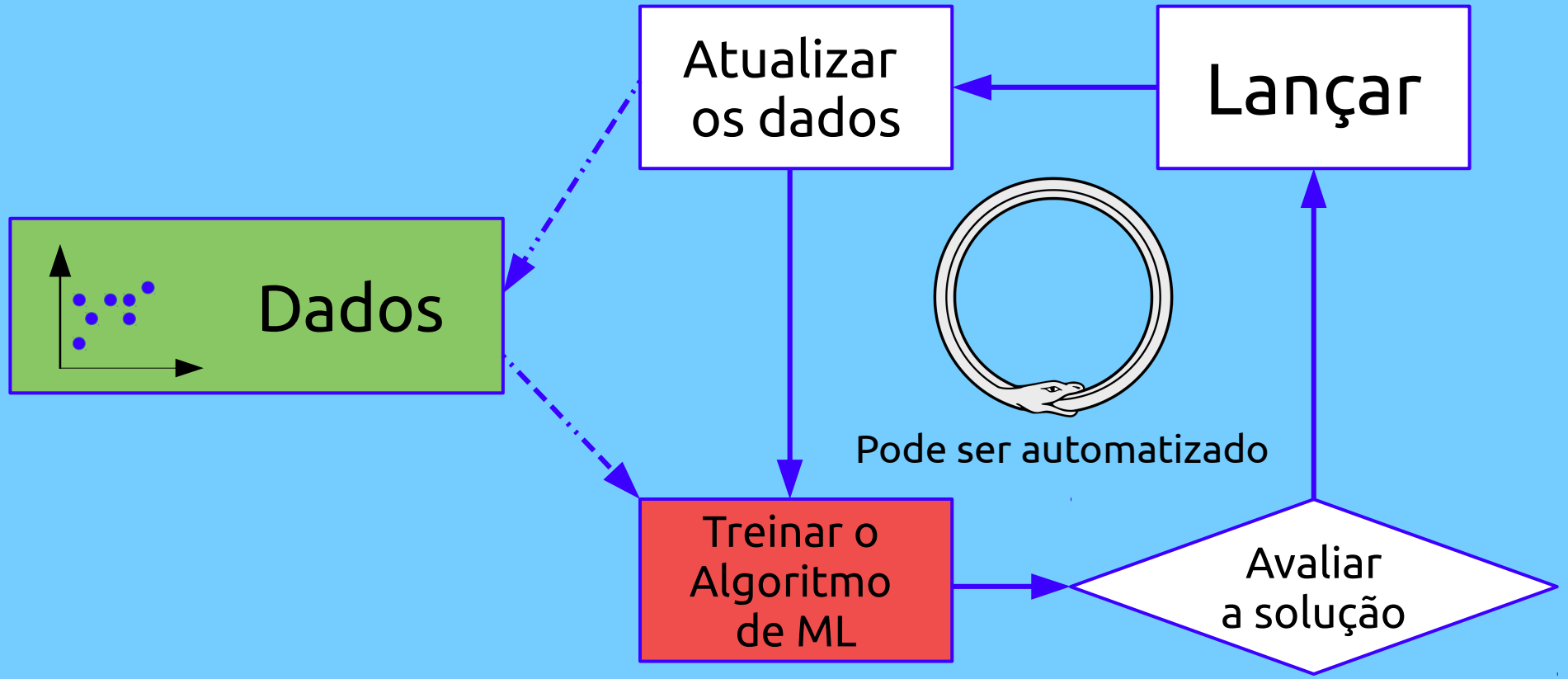
Adaptação Automática à Mudança

- E se os remetentes de spam (spammers) perceberem que todos os seus e-mails contendo determinadas palavras estão sendo bloqueados?
- Eles podem começar a escrever outras palavras em vez disso. Um filtro de spam usando técnicas de programação tradicionais precisaria ser atualizado para sinalizar e-mails com essas novas palavras.
- Se os spammers continuarem trabalhando em torno do seu filtro de spam, você precisará continuar escrevendo novas regras para sempre.

Adaptação Automática à Mudança

- Em contraste, um filtro de spam baseado em técnicas de Machine Learning automaticamente percebe que algumas palavras se tornaram incomumente frequente em spam (sinalizado por usuários) e começa a sinalizá-los sem sua intervenção.
- Essas técnicas de ML têm a capacidade de aprender e identificar e-mails de spam e mensagens de phishing, analisando cargas dessas mensagens em uma vasta coleção de computadores. Como o Machine Learning tem a capacidade de se adaptar a condições variáveis, os filtros de spam do Gmail e do Yahoo fazem mais do que apenas verificar e-mails indesejados usando regras pré-existentes. Eles geram novas regras com base no que aprenderam à medida que continuam em sua operação de filtragem de spam.

Adaptação Automática à Mudança

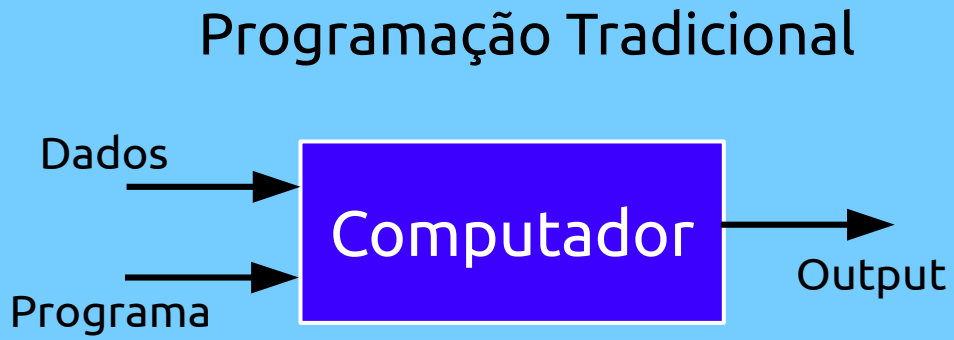


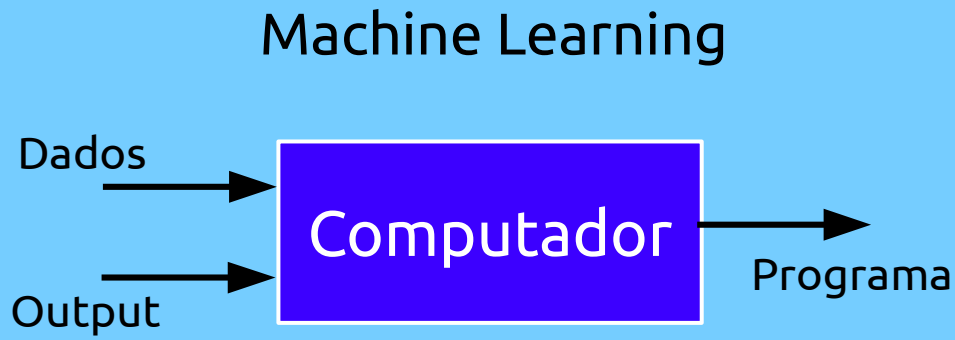
Machine Learning vs Programação Tradicional

- Machine Learning é fazer com que os computadores se programem. Se a programação é automação, o Machine Learning é automatizar o processo de automação.
- Escrever software é o gargalo, não temos desenvolvedores bons o suficiente. Deixamos que os dados façam o trabalho em vez das pessoas.
- O Machine Learning é a maneira de tornar a programação escalonável.
- O Machine Learning é como agricultura ou jardinagem. As sementes são os algoritmos, os nutrientes são os dados, o jardineiro é você e as plantas são os programas resultantes.

Machine Learning vs Programação Tradicional

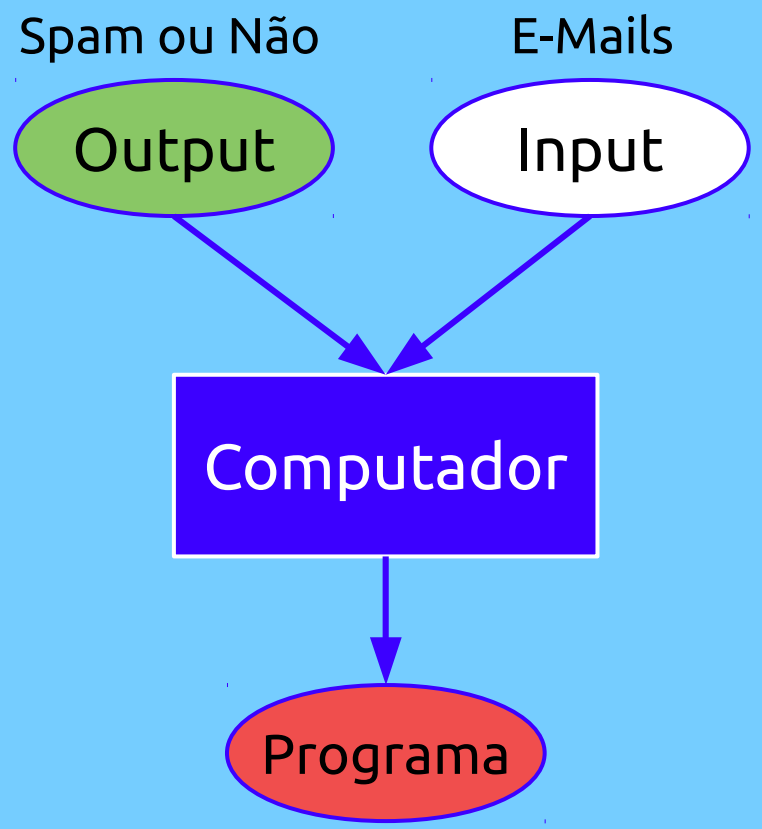
- **Programação tradicional**: os dados e o programa são executados no computador para produzir o output.
- **Machine Learning**: os dados e o output são executados no computador para criar um programa. Este programa pode ser usado na programação tradicional.





Aprendendo a Detectar Spam

- Como vimos, para fazermos um **algoritmo de Machine Learning** aprender a detectar spam, devemos usar e-mails do passado e rotular eles como spam ou não-spam.
- Então o algoritmo irá aprender um programa que recebe um e-mail futuro como input e decide se ele é spam ou não.



Aplicações de Machine Learning

- Vejamos alguns exemplos concretos de tarefas de Machine Learning, juntamente com as técnicas que podem resolvê-las:
  - Análise de imagens de produtos em uma linha de produção para classificá-los automaticamente. Esta é a classificação de imagens, normalmente realizada usando convolutional neural networks (CNN's).
  - Detecção de tumores em scans cerebrais. Esta é a segmentação semântica, onde cada pixel da imagem é classificado (pois queremos determinar a localização exata e a forma dos tumores), normalmente usando CNN's também.

Aplicações de Machine Learning

- Vejamos alguns exemplos concretos de tarefas de Machine Learning, juntamente com as técnicas que podem resolvê-las:
  - Classificando artigos de notícias automaticamente. Isso é Natural Language Processing (NLP) e, mais especificamente, classificação de texto, que pode ser resolvida usando recurrent neural networks (RNN's), CNN's ou Transformers.
  - Sinalizando automaticamente comentários ofensivos em fóruns de discussão. Esta também é uma classificação de texto, usando as mesmas ferramentas de NLP.
  - Sumarizando documentos longos automaticamente. Este é um ramo da NLP chamado de text summarization, novamente usando as mesmas ferramentas.

Aplicações de Machine Learning

- Vejamos alguns exemplos concretos de tarefas de Machine Learning, juntamente com as técnicas que podem resolvê-las:
  - Criando um chatbot ou um assistente pessoal. Isso envolve muitos componentes da NLP, incluindo natural language understanding (NLU) e módulos de resposta a perguntas.
  - Previsão da receita da sua empresa no próximo ano, com base em muitas métricas de desempenho. Esta é uma tarefa de regressão (ou seja, prever valores) que pode ser abordada usando qualquer modelo de regressão, como um modelo de regressão linear ou regressão polinomial, uma regressão SVM, uma regressão Random Forest ou uma rede neural artificial. Se você quiser levar em conta sequências de métricas de desempenho anteriores, pode usar RNN's, CNN's ou Transformers.

Aplicações de Machine Learning

- Vejamos alguns exemplos concretos de tarefas de Machine Learning, juntamente com as técnicas que podem resolvê-las:
  - Fazendo seu aplicativo reagir a comandos de voz. Este é o reconhecimento de voz, que requer o processamento de amostras de áudio: por serem sequências longas e complexas, normalmente são processadas usando RNN's, CNN's ou Transformers.
  - Detectando fraude de cartão de crédito. Esta é a detecção de anomalias.
  - Representando um conjunto de dados complexo e de alta dimensão em um diagrama claro e perspicaz. Esta é a visualização de dados, muitas vezes envolvendo técnicas de redução de dimensionalidade.

Aplicações de Machine Learning

- Vejamos alguns exemplos concretos de tarefas de Machine Learning, juntamente com as técnicas que podem resolvê-las:
  - Recomendar um produto no qual um cliente possa estar interessado, com base em compras anteriores. Este é um sistema de recomendação. Uma abordagem é alimentar compras anteriores (e outras informações sobre o cliente) para uma rede neural artificial e fazer com que ela produza a próxima compra mais provável. Essa rede neural normalmente seria treinada em sequências anteriores de compras em todos os clientes.
  - Construindo um bot inteligente para um jogo. Isso é frequentemente abordado usando Reinforcement Learning, que é um ramo do Machine Learning que treina agentes (como bots) para escolher as ações que irão maximizar suas recompensas ao longo do tempo (por exemplo, um bot pode receber uma recompensa cada vez que o jogador ganha alguns pontos de vida), dentro de um determinado ambiente (como o jogo). O famoso programa AlphaGo que venceu o campeão mundial no jogo Go foi construído usando RL.

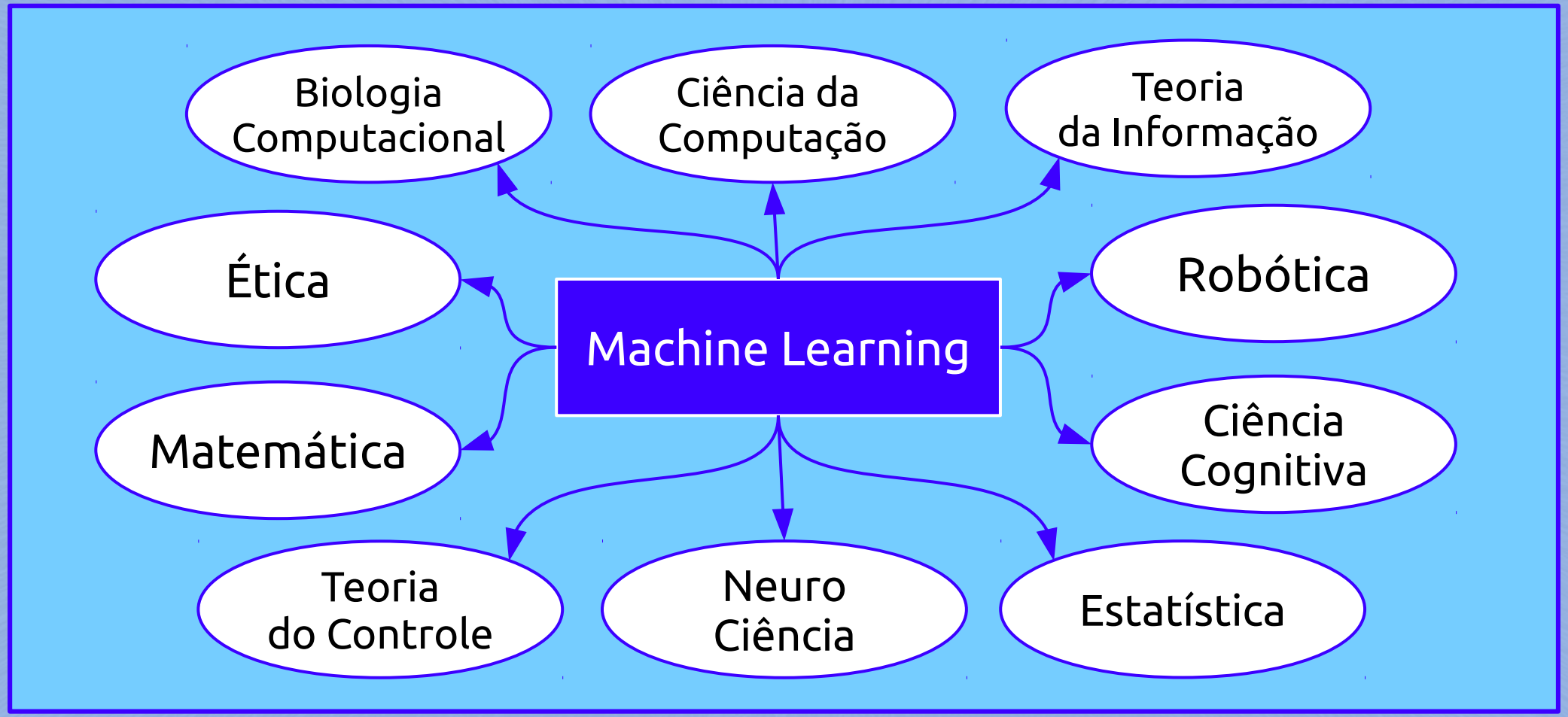
Outras Aplicações de ML

- **Web search**: ranking de páginas com base no que você tem mais probabilidade de clicar.
- **Biologia computacional**: design racional de drogas no computador com base em experimentos anteriores.
- **Finanças**: decida para quem enviar as ofertas de cartão de crédito. Avaliação de risco em ofertas de crédito. Como decidir onde investir dinheiro.
- **E-commerce**: previsão de rotatividade de clientes. Se uma transação é ou não fraudulenta.

Outras Aplicações de ML

- **Exploração espacial**: sondas espaciais e radioastronomia.
- **Robótica**: como lidar com a incerteza em novos ambientes. Autonomia. Carros autônomos.
- **Extração de informações**: fazer perguntas em bancos de dados na web.
- **Redes sociais**: dados sobre relacionamentos e preferências. Machine Learning para extrair valor dos dados.
- **Debugging**: use em problemas de ciência da computação, como debugging. Processo intensivo de mão de obra. Pode sugerir onde o bug pode estar.

Machine Learning é Interdisciplinar

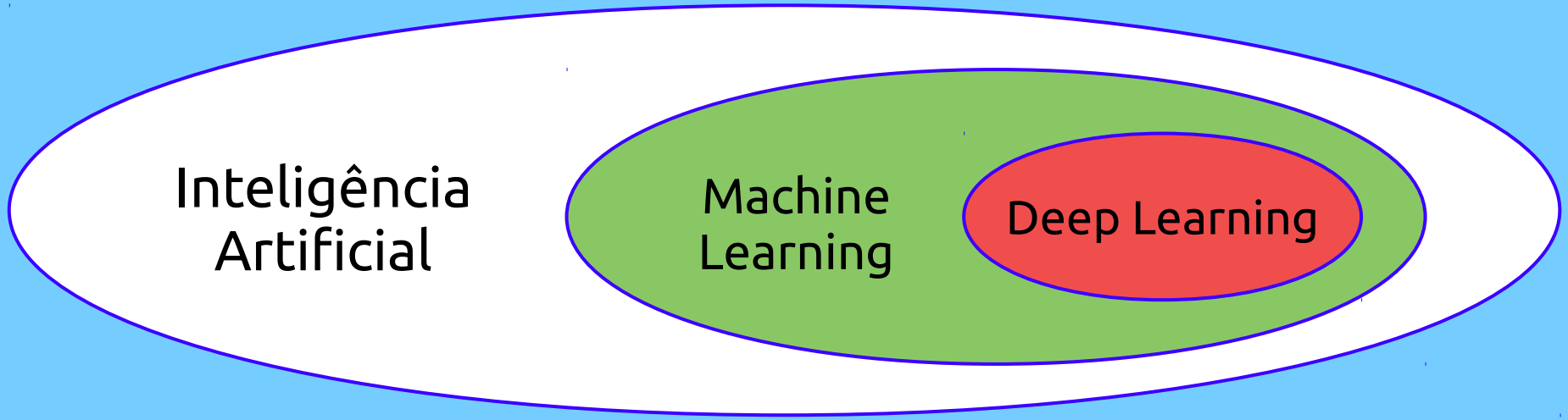


Elementos Chaves de ML

- Existem dezenas de milhares de algoritmos de Machine Learning e centenas de novos algoritmos são desenvolvidos a cada ano.
- Cada algoritmo de ML tem três componentes:
  - Representação: como representar o conhecimento. Os exemplos incluem árvores de decisão, conjuntos de regras, instâncias, modelos gráficos, redes neurais, support vector machines, model ensembles e outros.
  - Avaliação: a forma de avaliar os programas candidatos (hipóteses). Os exemplos incluem accuracy, precision e recall, erro quadrático, probabilidade, probabilidade posterior, custo, margem, divergência k-L de entropia e outros.
  - Otimização: a forma como os programas candidatos são gerados, conhecida como processo de busca. Por exemplo, otimização combinatória, otimização convexa, otimização restrita.

História do Machine Learning

- **Machine Learning** (ML) é o estudo de algoritmos de computador que se aprimoram automaticamente por meio da experiência e do uso de dados. É visto como um subcampo da inteligência artificial.



História do Machine Learning

- O termo Machine Learning foi cunhado em 1959 por Arthur Samuel, um IBMista americano e pioneiro no campo de jogos de computador e inteligência artificial.
- Um livro representativo da pesquisa de Machine Learning durante a década de 1960 foi o livro de Nilsson sobre Learning Machines, lidando principalmente com ML para classificação de padrões.
- O interesse relacionado ao reconhecimento de padrões continuou na década de 1970, conforme descrito por Duda e Hart em 1973.

História do Machine Learning

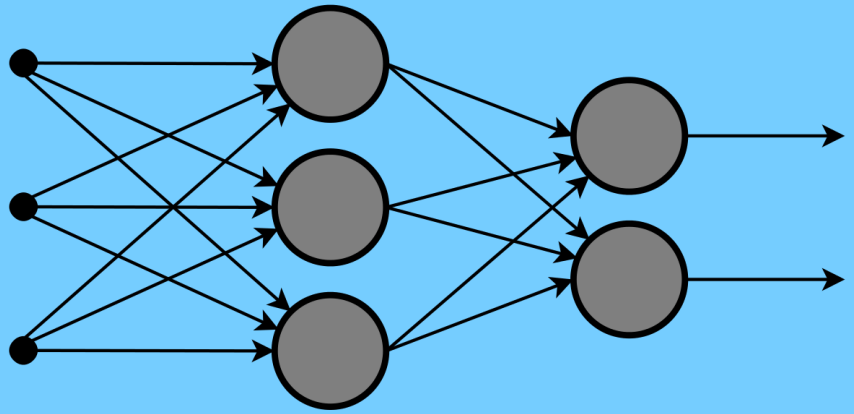
- Em 1981, foi apresentado um relatório sobre o uso de estratégias de ensino para que uma rede neural aprenda a reconhecer 40 caracteres (26 letras, 10 dígitos e 4 símbolos especiais) de um terminal de computador.
- Como um empreendimento científico, o Machine Learning surgiu da busca por inteligência artificial.
- No início da IA como disciplina acadêmica, alguns pesquisadores estavam interessados em que as máquinas aprendessem com os dados.
- Eles tentaram abordar o problema com vários métodos simbólicos, bem como com o que era então denominado "redes neurais"; estes eram principalmente perceptrons e outros modelos que mais tarde foram descobertos como reinvenções dos modelos lineares generalizados de estatística.

História do Machine Learning

- No entanto, uma ênfase crescente na abordagem lógica baseada no conhecimento causou uma cisão entre a IA e o Machine Learning.
- Os sistemas probabilísticos eram atormentados por problemas teóricos e práticos de aquisição e representação de dados.
- Em 1980, os sistemas especialistas passaram a dominar a IA, e as estatísticas eram desfavorecidas.
- O trabalho na aprendizagem simbólica / baseada no conhecimento continuou dentro da IA, levando à programação lógica indutiva, mas a linha de pesquisa mais estatística estava agora fora do campo da IA propriamente dita, no reconhecimento de padrões e recuperação de informações.

História do Machine Learning

- A pesquisa de redes neurais foi abandonada pela IA e pela ciência da computação na mesma época. Essa linha, também, foi continuada fora do campo de IA/CC, como "conexionismo", por pesquisadores de outras disciplinas, incluindo Hopfield, Rumelhart e Hinton. Seu principal sucesso veio em meados da década de 1980, com a reinvenção do backpropagation.



História do Machine Learning

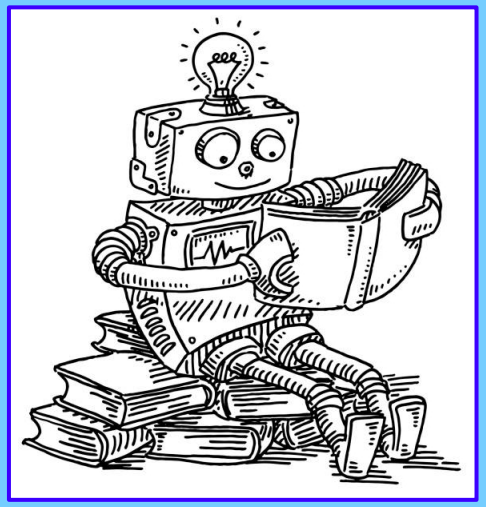
- O Machine Learning, reorganizado como um campo separado, começou a florescer na década de 1990.
- O campo mudou seu objetivo de alcançar inteligência artificial para resolver problemas solucionáveis de natureza prática.
- Ele mudou o foco das abordagens simbólicas que herdara da IA e passou a se concentrar em métodos e modelos emprestados da estatística e da teoria da probabilidade.
- Em 2020, muitas fontes continuam a afirmar que o ML continua sendo um subcampo da IA.

Data Mining

- **Machine Learning** e **Data Mining** frequentemente empregam os mesmos métodos e se sobrepõem significativamente, mas enquanto o Machine Learning se concentra na previsão, com base nas propriedades conhecidas aprendidas com os dados de treinamento, o Data Mining se concentra na descoberta de propriedades (anteriormente) desconhecidas nos dados (isto é a etapa de análise de knowledge discovery in databases).

Teoria de Machine Learning

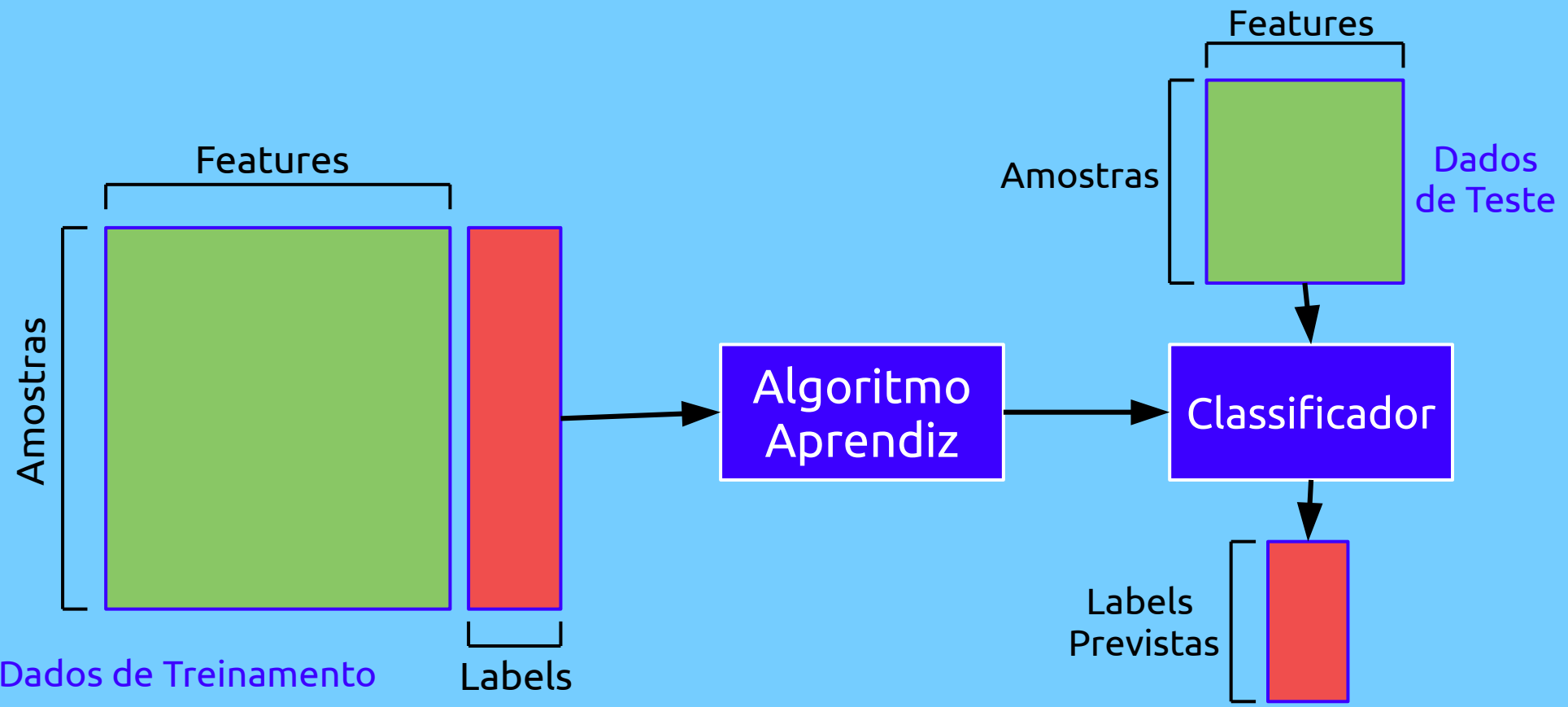
- Um objetivo central de um aprendiz é generalizar a partir de sua experiência.
- Generalização, neste contexto, é a capacidade de uma máquina aprendiz de executar com precisão em novos exemplos / tarefas não vistos, depois de ter experimentado um conjunto de dados de aprendizado.



Teoria de Machine Learning

- Os exemplos de treinamento vêm de alguma distribuição de probabilidade geralmente desconhecida (considerada representativa do espaço de ocorrências) e o aprendiz deve construir um modelo geral sobre esse espaço que lhe permita produzir previsões suficientemente precisas em novos casos.
- A análise computacional de algoritmos de Machine Learning e seu desempenho é um ramo da ciência da computação teórica conhecida como teoria de aprendizado computacional.
- Como os conjuntos de treinamento são finitos e o futuro é incerto, a teoria de aprendizagem geralmente não oferece garantias do desempenho dos algoritmos. Em vez disso, os limites probabilísticos do desempenho são bastante comuns.
- A bias–variance decomposition é uma forma de quantificar o erro de generalização.

Teoria de Machine Learning



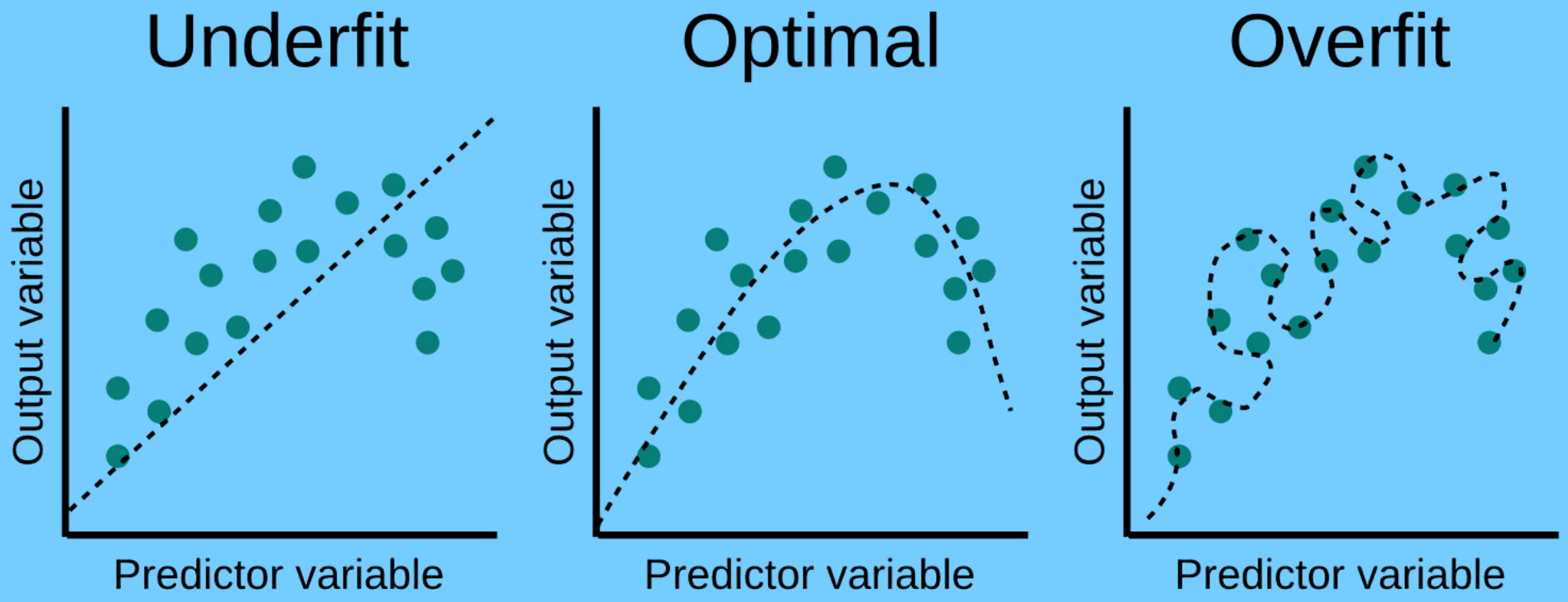
Teoria de Machine Learning

- **Dados de treinamento**: para construir / treinar seu classificador, você precisará fornecer uma biblioteca de exemplos de cada classe de destino. Para supervised learning, este conjunto de dados deve ser pré-rotulado e é normalmente referido como seus dados de treinamento.
- **Dados de teste**: o objetivo final de construir um classificador é ser capaz de usá-lo em dados nunca antes vistos e recuperar as classificações corretas para cada amostra de dados. Esse conjunto de dados invisível normalmente é conhecido como seus dados de teste.
- **Algoritmo de aprendizado**: o algoritmo de aprendizado é qualquer forma de Machine Learning que você escolheu para seu conjunto de dados. Freqüentemente, você terá que especificar não apenas a forma do algoritmo em si, mas também a forma da função de custo que o algoritmo emprega. Cada modelo de ML particiona o feature space de uma maneira diferente, impulsionada pelo objetivo de otimizar a função de custo.
- **Modelo de Machine Learning**: O modelo de ML é o output do algoritmo de aprendizado. Aplica-se os seus dados de teste para derivar a classe prevista de cada amostra de teste.

Teoria de Machine Learning

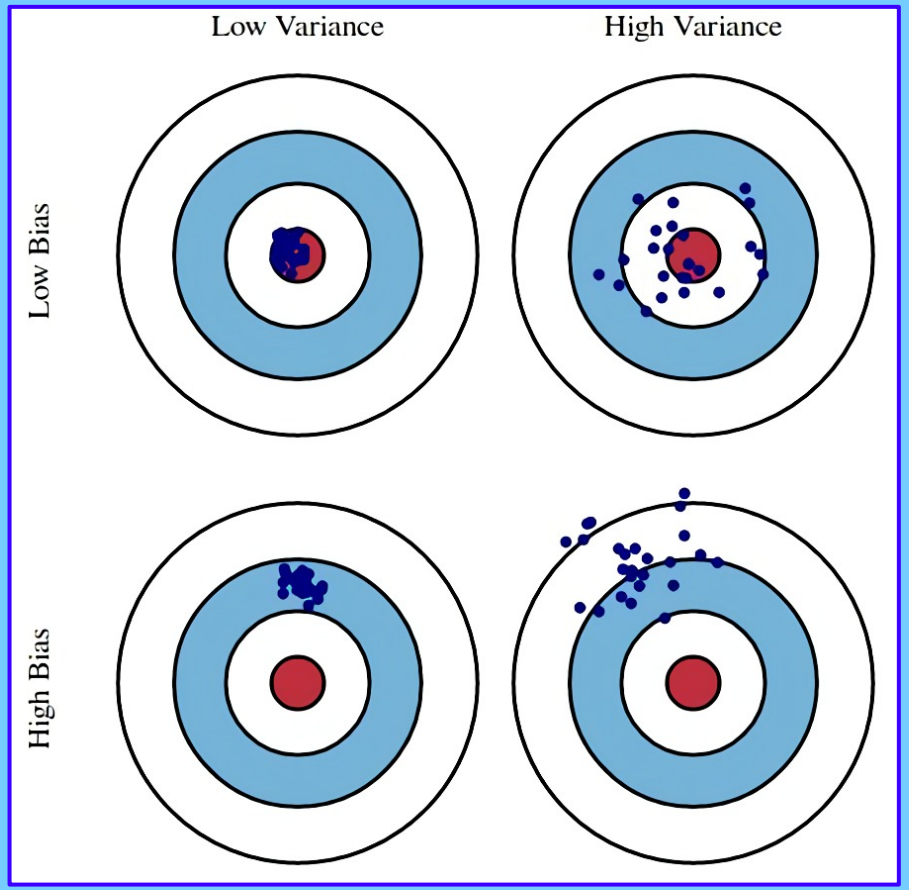
- Para o melhor desempenho no contexto de generalização, a complexidade da hipótese deve corresponder à complexidade da função subjacente aos dados. Se a hipótese for menos complexa do que a função, o modelo ajustou os dados de forma insuficiente (underfitted).
- Se a complexidade do modelo for aumentada em resposta, o erro de treinamento diminui. Mas se a hipótese for muito complexa, o modelo está sujeito a sobreajuste (overfitting) e a generalização será mais pobre.
- Além dos limites de desempenho, os teóricos da aprendizagem estudam a complexidade do tempo e a viabilidade da aprendizagem. Na teoria de aprendizagem computacional, um cálculo é considerado viável se puder ser feito em tempo polinomial. Existem dois tipos de resultados de complexidade de tempo. Resultados positivos mostram que uma determinada classe de funções pode ser aprendida em tempo polinomial. Resultados negativos mostram que certas classes não podem ser aprendidas em tempo polinomial.

Teoria de Machine Learning



Bias–variance Tradeoff

- Em estatística e Machine Learning, a compensação de viés-variância (bias–variance tradeoff) é a propriedade de um modelo de que a variância das estimativas de parâmetro entre as amostras pode ser reduzida aumentando o viés nos parâmetros estimados.



Bias–variance Tradeoff

- O dilema de bias–variance ou problema de bias–variance é o conflito em tentar minimizar simultaneamente essas duas fontes de erro que impedem que algoritmos de supervised learning generalizem além de seu conjunto de treinamento:
  - O bias error é um erro de suposições errôneas no algoritmo de aprendizado. A alto bias pode fazer com que um algoritmo perca as relações relevantes entre os features e os resultados desejados (underfitting).
  - A variance é um erro de sensibilidade a pequenas flutuações no conjunto de treinamento. A alta variance pode resultar de um algoritmo que modela o ruído aleatório nos dados de treinamento (overfitting).

Bias–variance Tradeoff

- A bias-variance tradeoff é um problema central no supervised learning.
- O ideal é escolher um modelo que capture com precisão as regularidades em seus dados de treinamento, mas também generalize bem para dados invisíveis.
- Infelizmente, normalmente é impossível fazer as duas coisas simultaneamente.
- A bias–variance decomposition é uma forma de analisar o erro de generalização esperado de um algoritmo de aprendizagem com relação a um problema específico como uma soma de três termos, a bias, a variance e uma quantidade chamada erro irredutível, resultante do ruído no próprio problema.

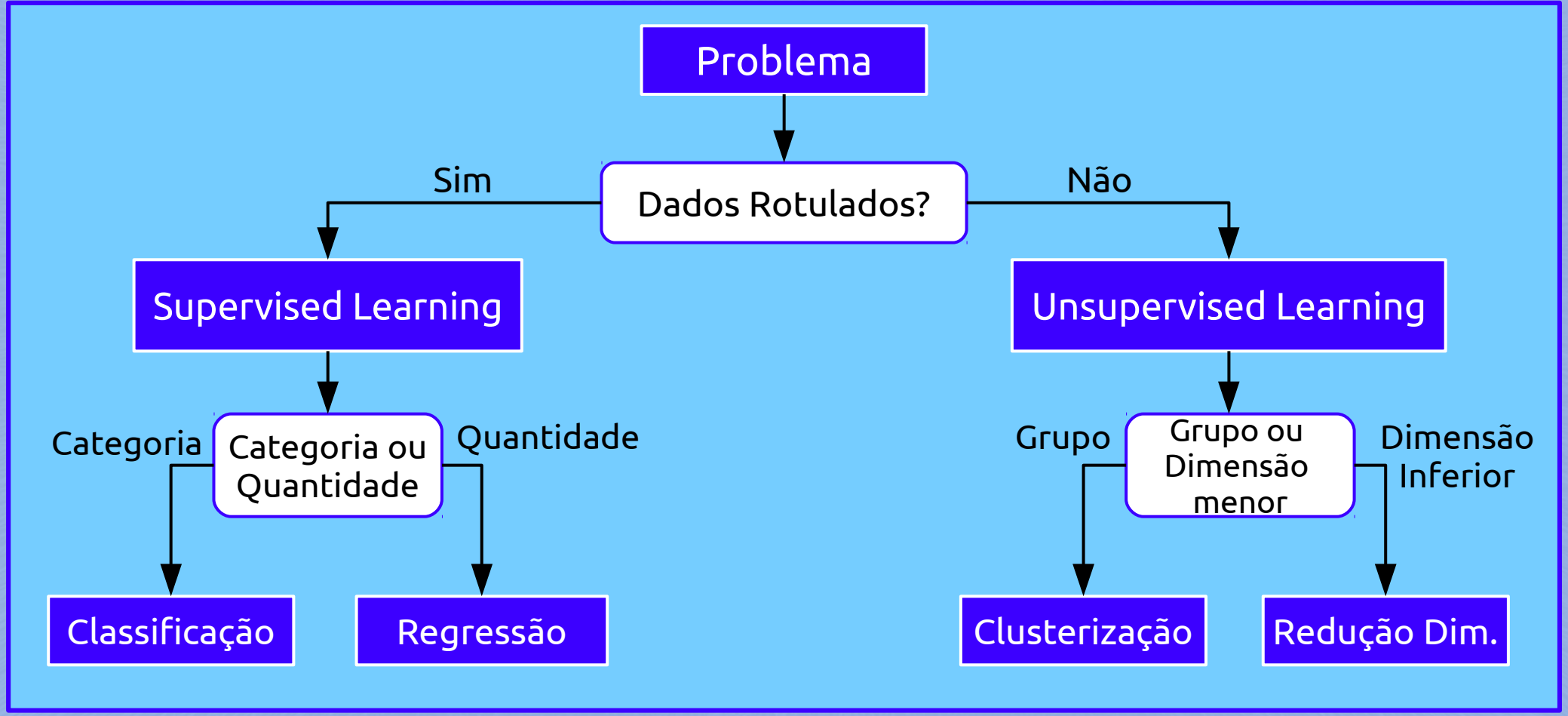
Tipos de Sistemas de Machine Learning

- Existem tantos tipos diferentes de sistemas de Machine Learning que é útil classificá-los em categorias amplas, com base nos seguintes critérios:
  - Sejam ou não treinados com supervisão humana (supervised, unsupervised, semisupervised e reinforcement learning).
  - Se eles podem ou não aprender de forma incremental na hora (online vs batch learning).
  - Se eles trabalham simplesmente comparando novos pontos de dados com pontos de dados conhecidos ou, em vez disso, detectando padrões nos dados de treinamento e construindo um modelo preditivo, assim como os cientistas fazem (instance-based vs model-based learning).
- Esses critérios não são exclusivos; é possível combiná-los da maneira que desejarmos.

Supervised & Unsupervised Learning

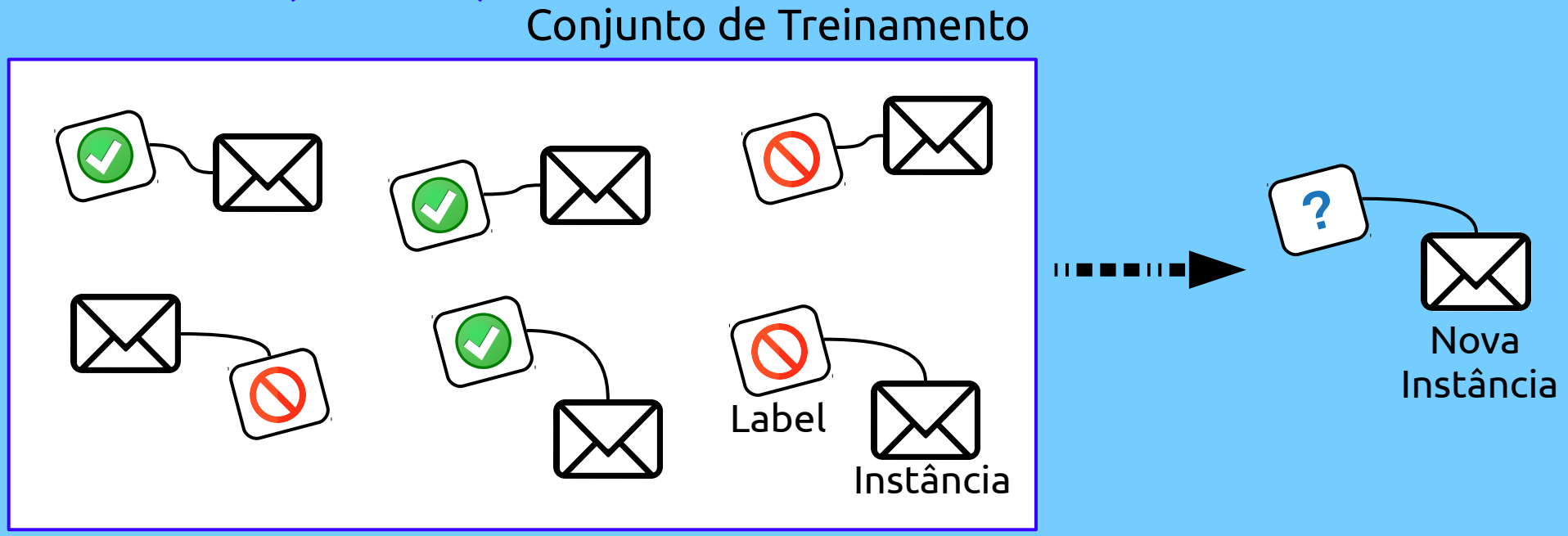
- Os sistemas de Machine Learning podem ser classificados de acordo com a quantidade e o tipo de supervisão que recebem durante o treinamento.
- Existem quatro categorias principais:
  - Supervised Learning;
  - Unsupervised Learning;
  - Semisupervised Learning;
  - Reinforcement Learning;

Supervised & Unsupervised Learning



Supervised Learning

- Em Supervised Learning, o conjunto de treinamento que você alimenta ao algoritmo inclui as soluções desejadas, chamadas de labels (rótulos).

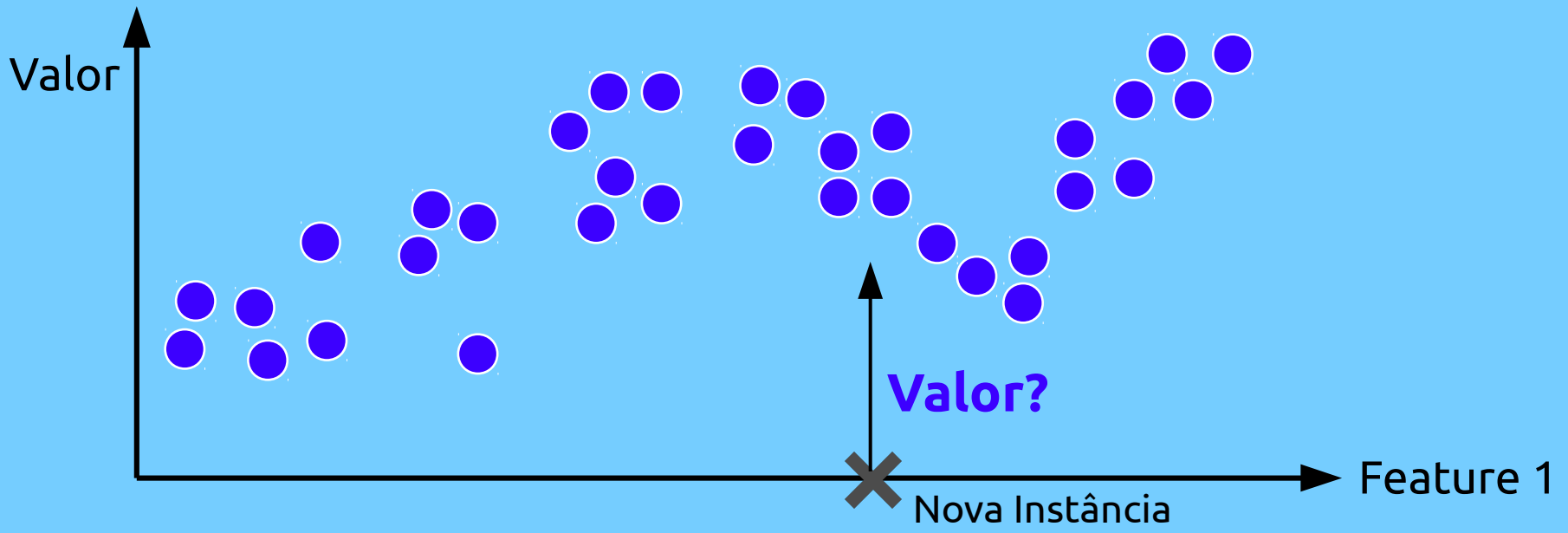


Supervised Learning

- Uma tarefa típica 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 sua classe (spam ou regular) e deve aprender a classificar novos e-mails.
- Outra tarefa típica é prever um valor numérico alvo, como o preço de um carro, dado um conjunto de features (quilometragem, idade, marca, etc.) chamados preditores. Esse tipo de tarefa é chamado de regressão.
- Para treinar o sistema, precisamos dar a ele muitos exemplos de carros, incluindo seus preditores e suas labels (ou seja, seus preços).

Problema de Regressão

- Um problema de regressão: prever um valor, dado um input feature (geralmente há vários input features e, às vezes, vários valores de output).



Algoritmos de Supervised Learning

- 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

Unsupervised Learning

- Em unsupervised learning, como é possível imaginar, os dados de treinamento não são rotulados (não possuem labels).
- O sistema tenta aprender sem um professor.
- A seguir temos um conjunto de treinamento não rotulado para unsupervised learning.

Conjunto de Treinamento

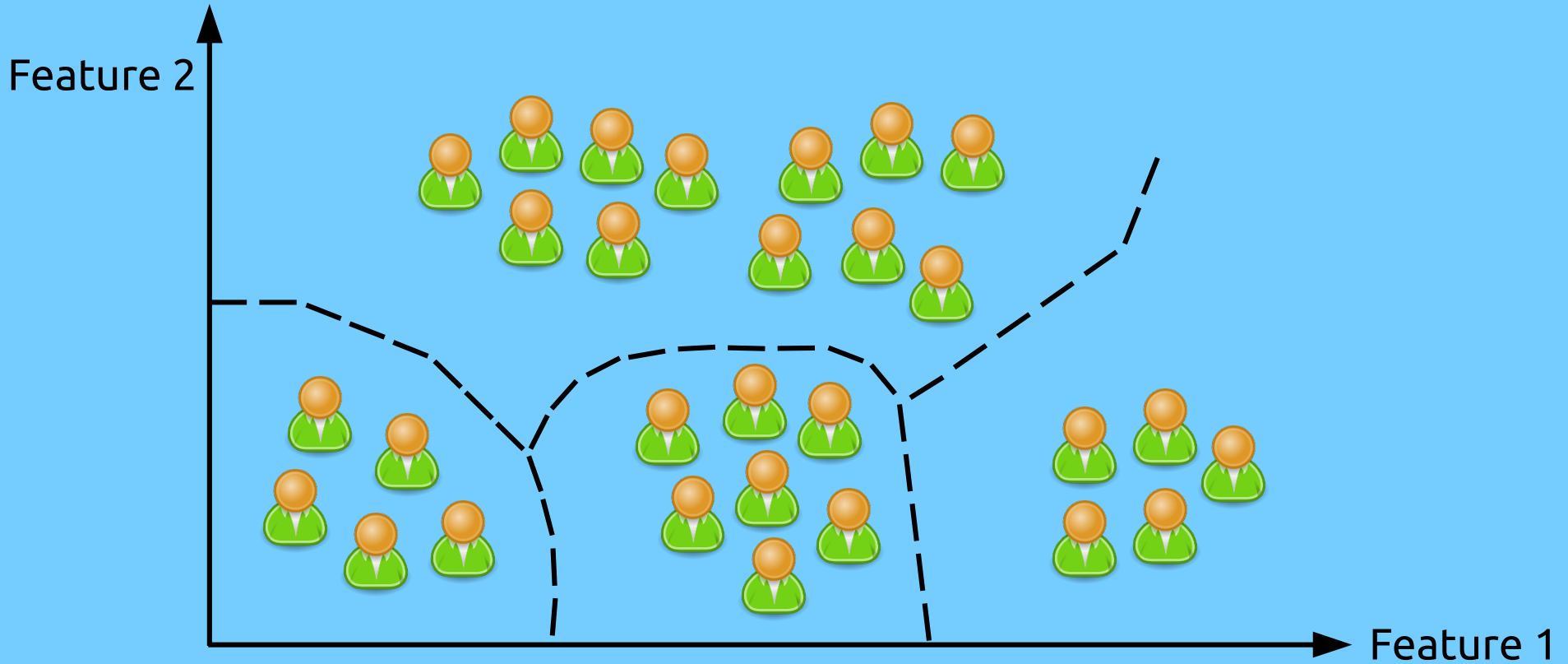
Algoritmos de Unsupervised Learning

- Aqui estão alguns dos algoritmos de unsupervised learning mais importantes:
  - **Clusterização**: K-Means, DBSCAN, Hierarchical Cluster Analysis (HCA);
  - **Detecção de anomalias e detecção de novidades**: One-class SVM, Isolation Forest;
  - **Visualização e redução de dimensionalidade**: Principal Component Analysis (PCA), Kernel PCA, Locally Linear Embedding (LLE), t-Distributed Stochastic Neighbor Embedding (t-SNE);
  - **Aprendizagem de regras de associação**: Apriori, Eclat;

Unsupervised Learning

- Por exemplo, digamos que você tenha muitos dados sobre os visitantes do seu blog. Você pode querer executar um algoritmo de clusterização para tentar detectar grupos de visitantes semelhantes. Em nenhum momento você diz ao algoritmo a qual grupo um visitante pertence: ele encontra essas conexões sem a sua ajuda. Por exemplo, você pode notar que 40% de seus visitantes são homens que amam histórias em quadrinhos e geralmente leem seu blog à noite, enquanto 20% são jovens amantes de ficção científica que o visitam durante os fins de semana. Se você usar um algoritmo de agrupamento hierárquico, ele também pode subdividir cada grupo em grupos menores. Isso pode ajudá-lo a direcionar suas postagens para cada grupo.

Unsupervised Learning



Unsupervised Learning

- 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 geram uma representação 2D ou 3D de seus dados que podem ser facilmente plotados. Esses algoritmos tentam preservar o máximo de estrutura possível (por exemplo, tentando manter clusters separados no input space de sobreposição na visualização) para que você possa entender como os dados são organizados e talvez identificar padrões insuspeitos.

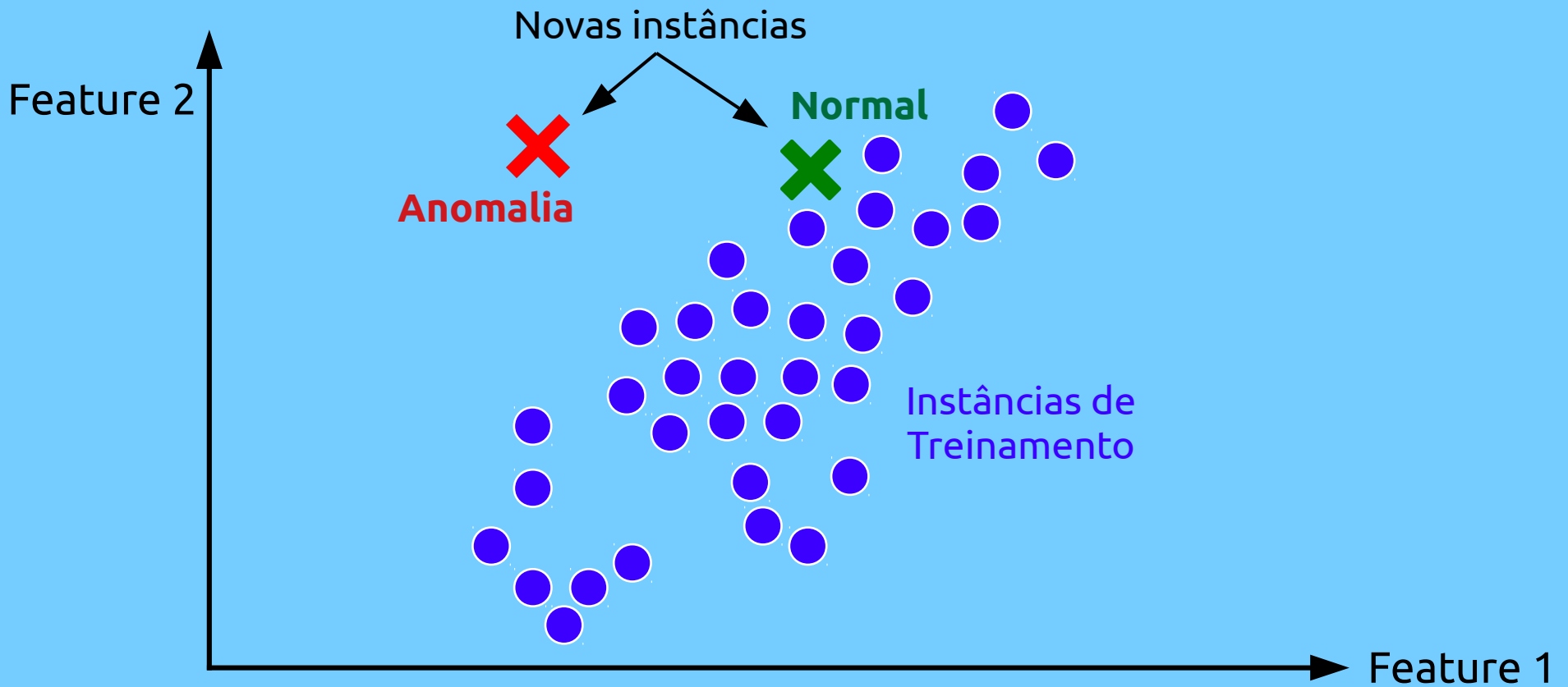
Unsupervised Learning

- Uma tarefa relacionada é a redução da dimensionalidade, na qual o objetivo é simplificar os dados sem perder muitas informações.
- Uma maneira de fazer isso é mesclar vários recursos correlacionados em um.
- Por exemplo, a quilometragem de um carro pode estar fortemente correlacionada com sua idade, então o algoritmo de redução de dimensionalidade irá mesclá-los em um recurso que representa o desgaste do carro. Isso é chamado de feature extraction.
- Geralmente, é uma boa ideia tentar reduzir a dimensão de seus dados de treinamento usando um algoritmo de redução de dimensionalidade antes de alimentá-los para outro algoritmo de Machine Learning (como um algoritmo de supervised learning). Ele será executado com muito mais rapidez, os dados ocuparão menos espaço em disco e memória e, em alguns casos, também poderá ter um desempenho melhor.

Anomaly Detection

- Outra importante tarefa unsupervised é a detecção de anomalias - por exemplo, detectar transações incomuns de cartão de crédito para evitar fraudes, detectar defeitos de fabricação ou remover automaticamente valores discrepantes (outliers) de um conjunto de dados antes de alimentá-lo para outro algoritmo de aprendizagem.
- O sistema é mostrado principalmente instâncias normais durante o treinamento, então ele aprende a reconhecê-los; então, quando vê uma nova instância, pode dizer se parece normal ou se é provavelmente uma anomalia.
- Uma tarefa muito semelhante é a detecção de novidades: seu objetivo é detectar novas instâncias que parecem diferentes de todas as instâncias no conjunto de treinamento. Isso requer um conjunto de treinamento muito "limpo", desprovido de qualquer instância que você gostaria que o algoritmo detectasse.

Anomaly Detection

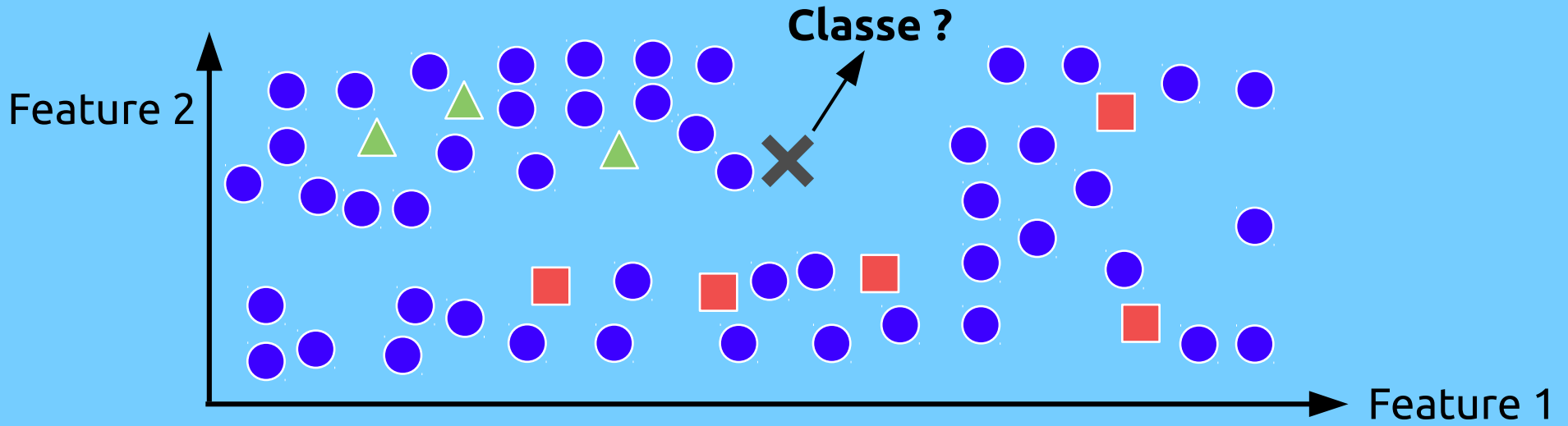


Association Rule Learning

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

Semisupervised Learning

- Uma vez que rotular dados normalmente é demorado e custoso, você geralmente terá muitas instâncias não rotuladas e poucas instâncias rotuladas. Alguns algoritmos podem lidar com dados parcialmente rotulados. Isso é chamado de semisupervised learning.



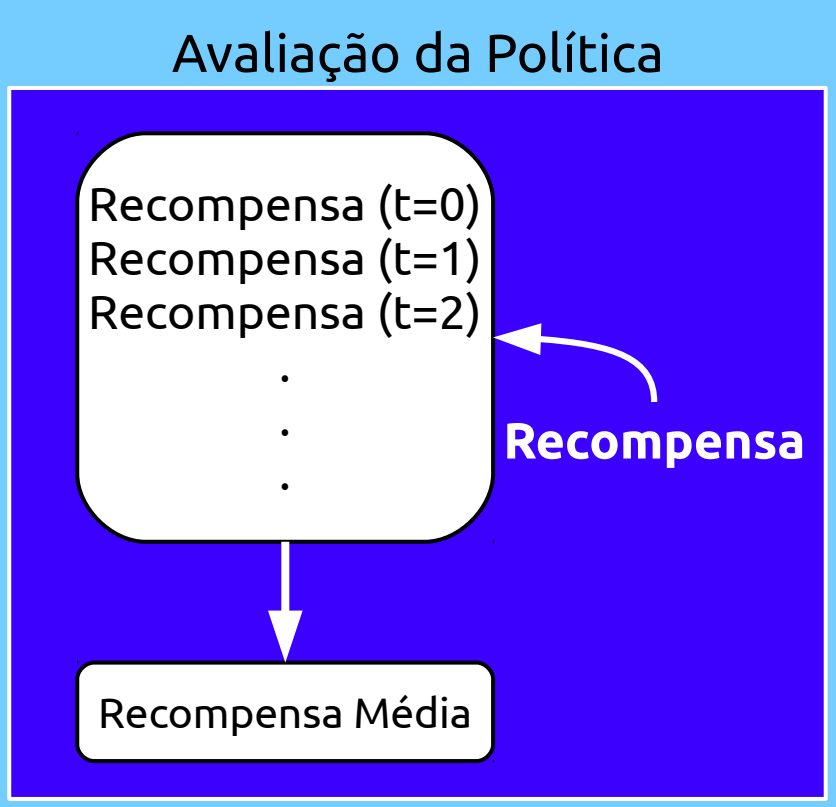
Semisupervised Learning

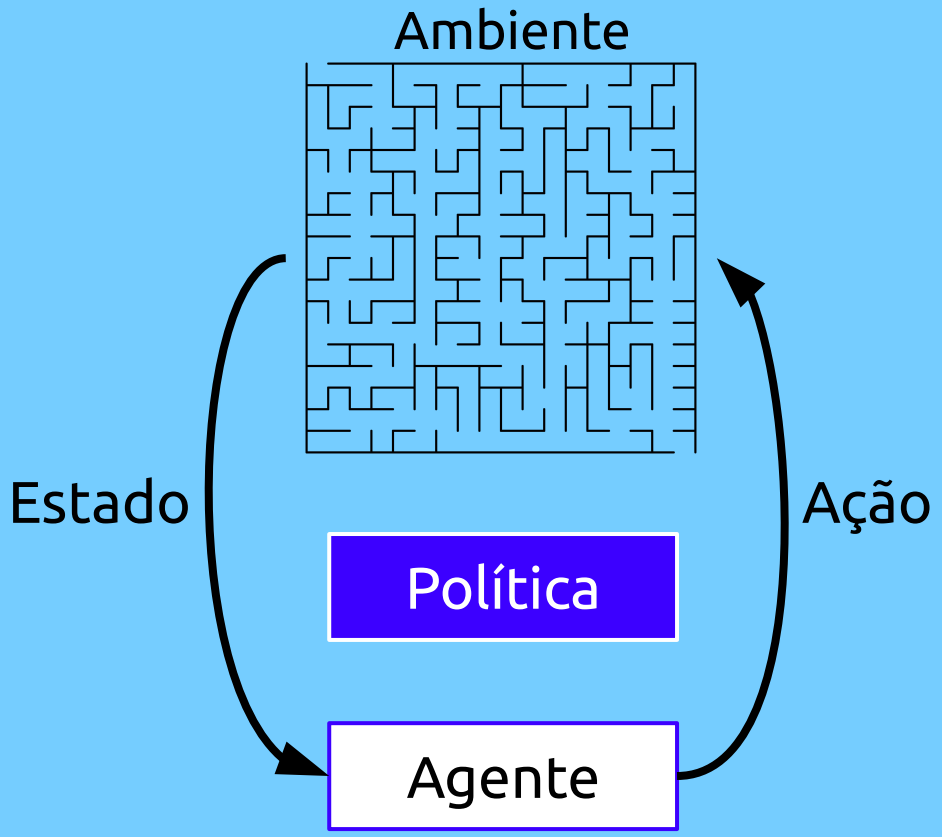
- Figura anterior mostra o semisupervised learning com duas classes (triângulos e quadrados): os exemplos não rotulados (círculos) ajudam a classificar uma nova instância (a cruz) na classe triângulo em vez da classe quadrada, embora seja mais próxima dos quadrados rotulados.
- Alguns serviços de hospedagem de fotos, como o Google Photos, são bons exemplos disso. Depois de fazer upload de todas as fotos de sua 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 unsupervised do algoritmo (clustering). Agora, tudo o que o sistema precisa é que você diga quem são essas pessoas. Basta adicionar um rótulo por pessoa e ele é capaz de nomear todas as pessoas em todas as fotos, o que é útil para pesquisar fotos.

Reinforcement Learning

- Reinforcement Learning é completamente diferente. O sistema de aprendizagem, chamado de agente neste contexto, pode observar o ambiente, selecionar e executar ações e obter recompensas em troca (ou penalidades na forma de recompensas negativas). Ele deve então aprender por si mesmo 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 estiver em uma determinada situação.

Reinforcement Learning





Reinforcement Learning

- 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 ganhou as manchetes em maio de 2017 quando derrotou o campeão mundial Ke Jie no jogo Go. Ele aprendeu sua política de vitórias analisando milhões de jogos e, então, jogando muitos jogos contra si mesmo. Observe que o aprendizado foi desligado durante os jogos contra o campeão; AlphaGo estava apenas aplicando a política que aprendeu.

Batch & Online Learning

- Os dados são um componente vital para a construção de modelos de aprendizagem. Existem duas opções de design para como os dados são usados no pipeline de modelagem. A primeira é construir seu modelo de aprendizagem com dados em repouso (batch learning), e a outra é quando os dados estão fluindo em fluxos para o algoritmo de aprendizagem (online learning). Esse fluxo pode ser como pontos de amostra individuais em seu conjunto de dados ou pode ser em pequenos tamanhos de batch (lotes).

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 de computação, portanto, geralmente é feito offline. Primeiro, o sistema é treinado e, em seguida, é lançado em produção e funciona sem aprender mais; apenas aplica o que aprendeu. Isso é chamado de aprendizagem offline.
- Se você deseja que um sistema de batch learning saiba sobre novos dados (como um novo tipo de spam), você precisa treinar uma nova versão do sistema desde o início com o conjunto de dados completo (não apenas os novos dados, mas também os antigos), só então pare o sistema antigo e substitua-o pelo novo.

Batch Learning

- Felizmente, todo o processo de treinamento, avaliação e lançamento de um sistema de Machine Learning pode ser automatizado com bastante facilidade, portanto, até mesmo um sistema de batch learning pode se adaptar às mudanças. Simplesmente atualize os dados e treine uma nova versão do sistema do zero com a freqüência necessária.
- Essa solução é simples e geralmente funciona bem, mas o treinamento usando o conjunto completo de dados pode levar muitas horas, portanto, você normalmente treinaria um novo sistema apenas a cada 24 horas ou mesmo apenas semanalmente.
- Se o seu sistema precisa se adaptar aos dados que mudam rapidamente (por exemplo, para prever os preços das ações), você precisa de uma solução mais reativa.

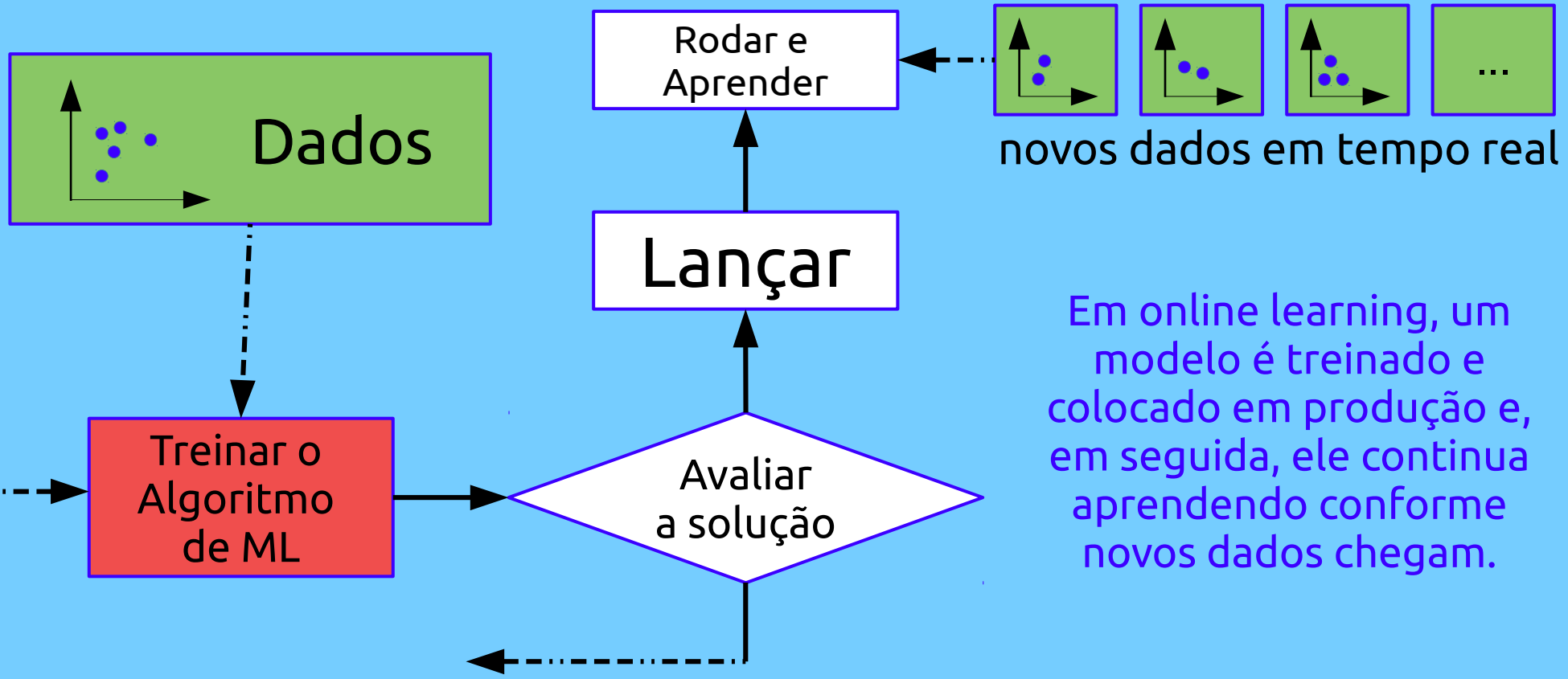
Batch Learning

- Além disso, o treinamento no conjunto completo de dados requer muitos recursos de computação (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 vai acabar 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 precisa ser capaz de aprender de forma autônoma e tem recursos limitados (por exemplo, um aplicativo de smartphone ou um rover em Marte), então carregar grandes quantidades de dados de treinamento ocupará muitos recursos, para treinar por horas a cada dia será um empecilho.

Online Learning

- Em online learning, você treina o sistema de forma incremental, alimentando-o com instâncias de dados sequencialmente, individualmente ou em pequenos grupos chamados mini-batches. Cada etapa de aprendizado é rápida e barata, então o sistema pode aprender sobre novos dados rapidamente, conforme eles chegam.
- Online learning é ótimo para sistemas que recebem dados como um fluxo contínuo (por exemplo, preços de ações) e precisam se adaptar para mudar rapidamente ou de forma autônoma. Também é uma boa opção se você tiver recursos de computação limitados: uma vez que um sistema de online learning tenha aprendido sobre novas instâncias de dados, ele não precisa mais deles, então você pode descartá-los (a menos que queira reverter para um estado anterior e "reproduzir" os dados). Isso pode economizar muito espaço.

Online Learning

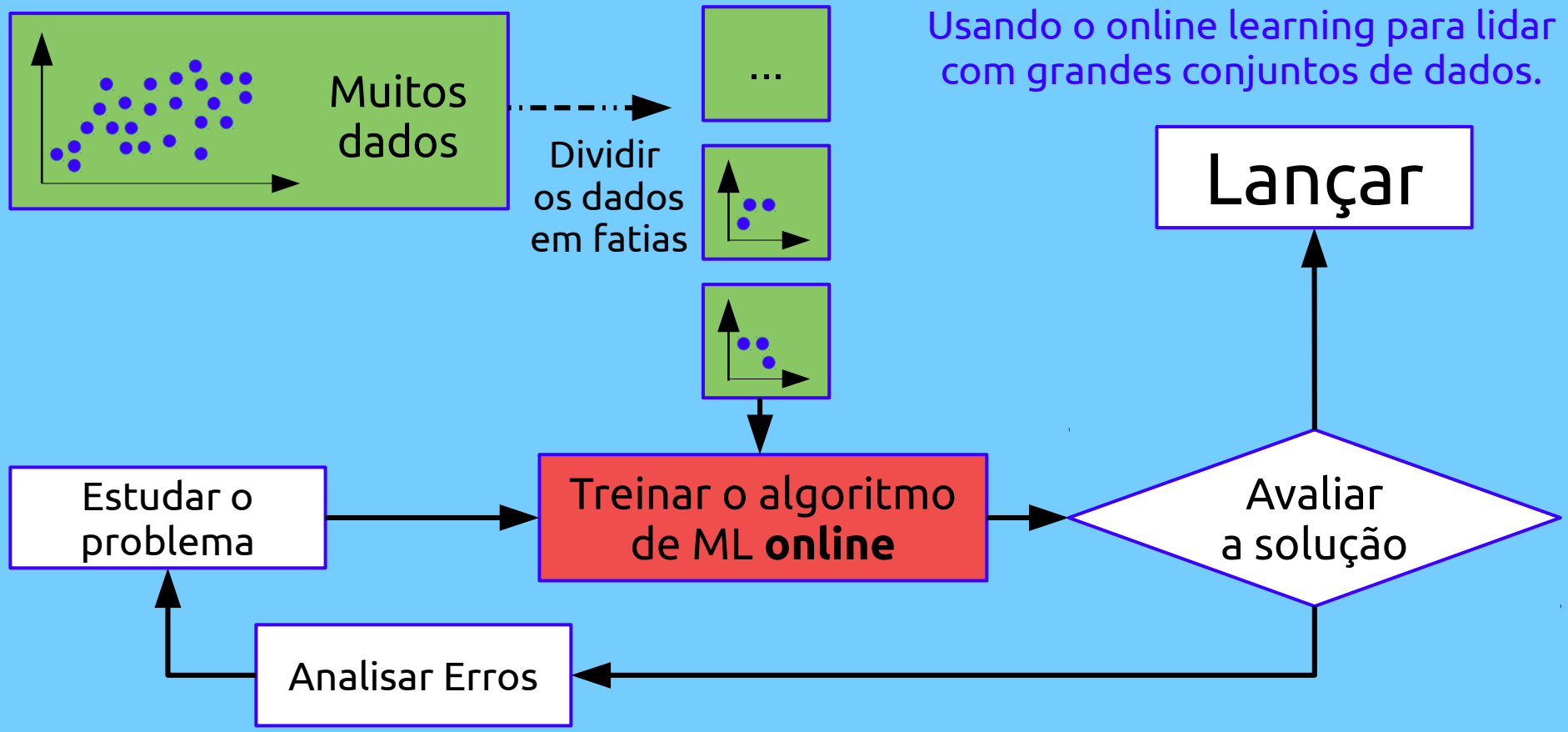


Em online learning, um modelo é treinado e colocado em produção e, em seguida, ele continua aprendendo conforme novos dados chegam.

Online Learning

- Os algoritmos de online learning também podem ser usados para treinar sistemas em enormes conjuntos de dados que não cabem na memória principal de uma 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 que seja executado em todos os dados.
- Um parâmetro importante dos sistemas de online learning é a rapidez com que devem se adaptar aos dados variáveis: isso é chamado de taxa de aprendizagem. Se você definir uma alta taxa de aprendizado, 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 sinalize apenas os tipos mais recentes de spam que foi mostrado).
- Por outro lado, se você definir uma taxa de aprendizado baixa, o sistema terá mais inércia; ou seja, ele aprenderá mais lentamente, mas também será menos sensível a ruídos nos novos dados ou a sequências de pontos de dados não representativos (outliers).

Online Learning



Online Learning

- Um grande desafio com o online learning é que, se dados ruins forem fornecidos ao sistema, o desempenho do sistema diminuirá gradualmente.
- Se for um sistema ativo, seus clientes perceberão. Por exemplo, dados inválidos podem vir de um sensor com defeito em um robô ou de alguém enviando spam em um search engine para tentar obter uma classificação elevada nos resultados da pesquisa.
- Para reduzir esse risco, você precisa monitorar seu sistema de perto e desligar imediatamente o aprendizado (e possivelmente reverter para um estado de funcionamento anterior) se detectar uma queda no desempenho. Você também pode querer monitorar os dados de input e reagir aos dados anormais (por exemplo, usando um algoritmo de detecção de anomalias).

Instance-Based vs Model-Based Learning

- Outra maneira de categorizar os sistemas de Machine Learning é por meio da generalização.
- A maioria das tarefas de Machine Learning envolve fazer previsões.
- Isso significa que, dados vários exemplos de treinamento, o sistema precisa ser capaz de fazer boas previsões para (generalizar) exemplos que 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 generalização: instance-based learning e model-based learning.

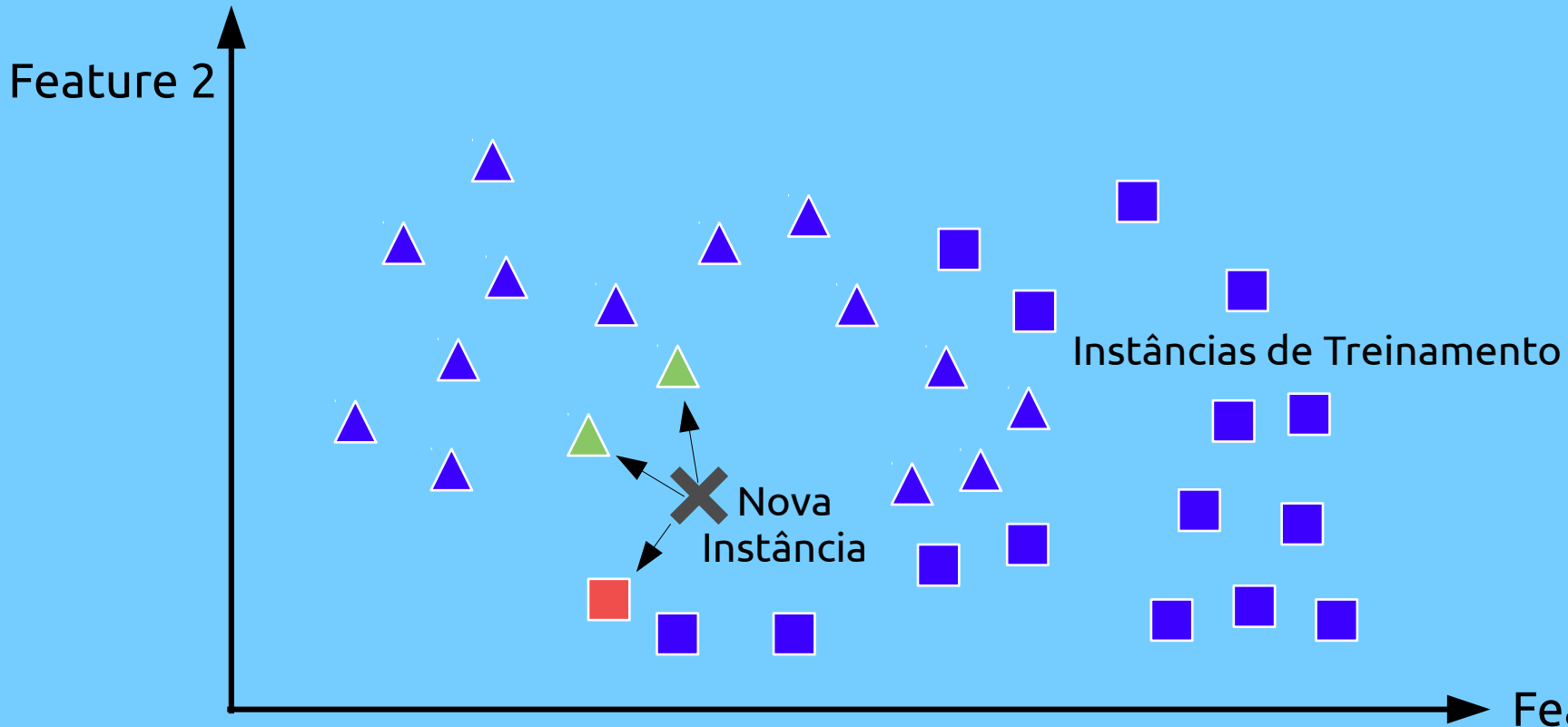
Instance-Based Learning

- Possivelmente, a forma mais trivial de aprendizagem é simplesmente aprender de cor. Se você criasse um filtro de spam dessa maneira, ele apenas sinalizaria todos os emails idênticos aos emails que já foram sinalizados pelos usuários - não é a pior solução, mas certamente não é a melhor.
- Em vez de apenas sinalizar e-mails que são idênticos aos e-mails de spam conhecidos, seu filtro de spam pode ser programado para também sinalizar e-mails que são muito semelhantes aos e-mails de spam conhecidos. Isso requer uma medida de semelhança entre dois e-mails. Uma medida de similaridade (muito básica) entre dois e-mails pode ser contar o número de palavras que eles têm em comum. O sistema sinalizaria um e-mail como spam se ele tiver muitas palavras em comum com um e-mail de spam conhecido.

Instance-Based Learning

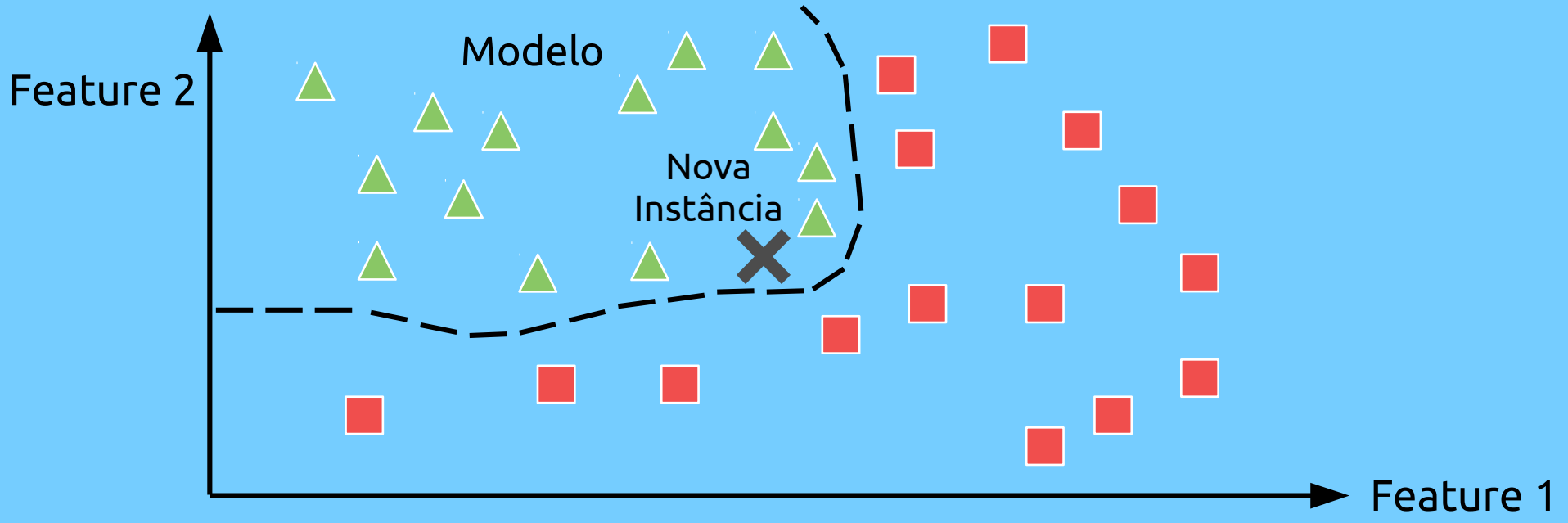
- Isso é chamado de instance-based learning: o sistema aprende os exemplos de cor e, em seguida, generaliza para novos casos usando uma medida de similaridade para compará-los aos exemplos aprendidos (ou um subconjunto deles).
- Por exemplo, na Figura a seguir, a nova instância seria classificada como um triângulo porque a maioria das instâncias mais semelhantes pertencem a essa classe.

Instance-Based Learning



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.



Model-Based Learning

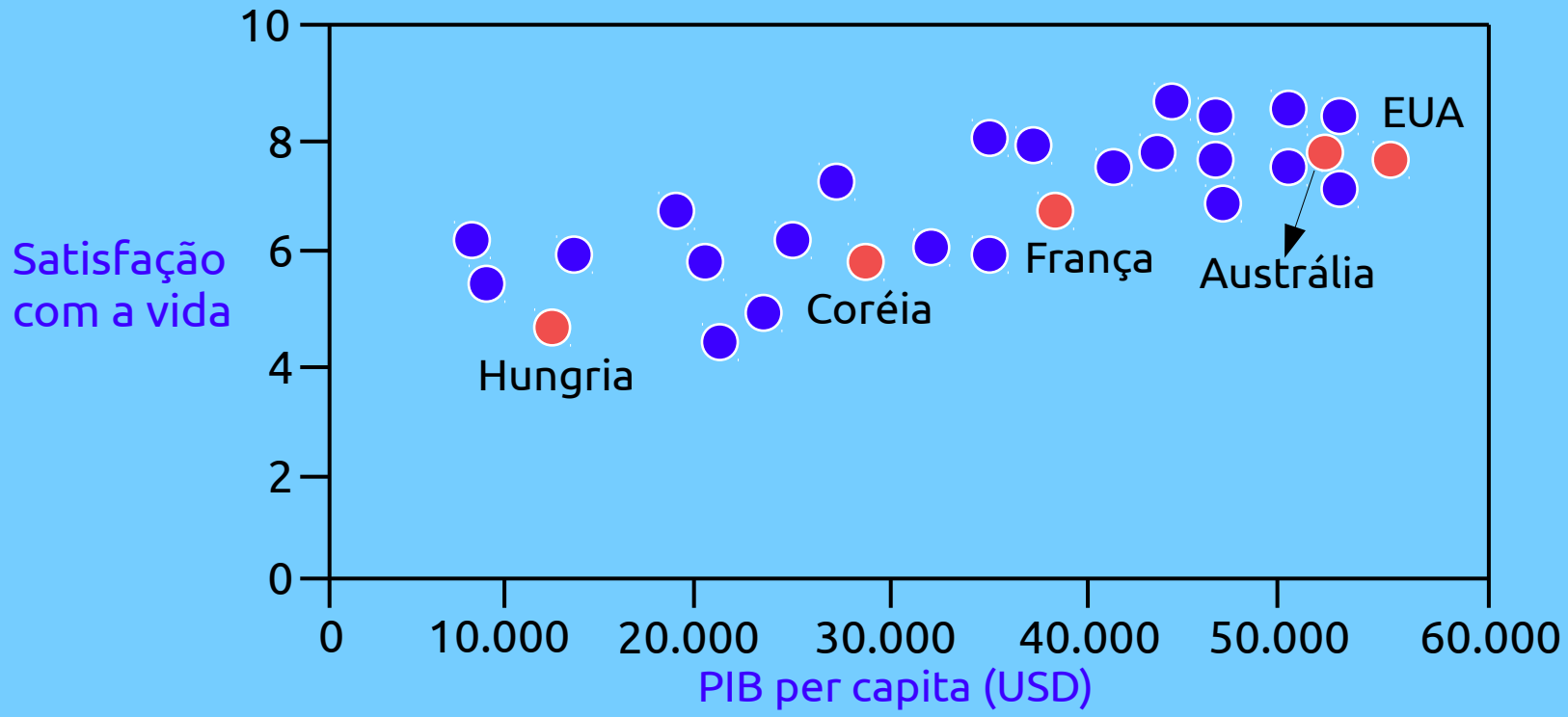
- Por exemplo, suponha que desejamos saber se o dinheiro faz as pessoas felizes, então baixamos os dados do Índice para uma Vida Melhor do site da OCDE e estatísticas sobre produto interno bruto (PIB) per capita do site do FMI. Em seguida, unimos as tabelas e ordenamos pelo PIB per capita. A Tabela a seguir mostra um trecho do que obtemos.

Dinheiro traz felicidade?

País	PIB per capita	Satisfação
Hungria	12.240 USD	4.9
Coréia	27.195 USD	5.8
França	37.675 USD	6.5
Austrália	50.962 USD	7.3
EUA	55.805 USD	7.2

Model-Based Learning

- Vamos agora plotar os dados para esses países.

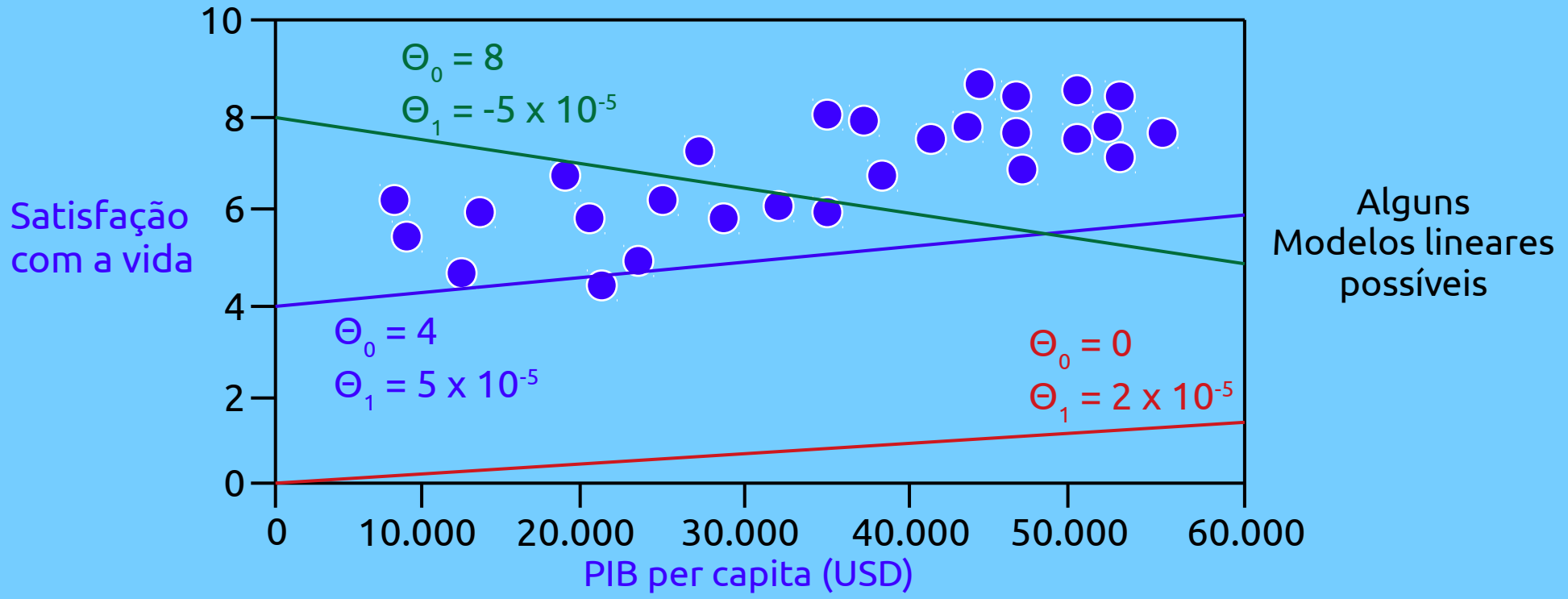


Model-Based Learning

- Parece haver uma tendência aqui! Embora os dados sejam ruidosos (ou seja, parcialmente aleatórios), parece que a satisfação com a vida aumenta mais ou menos linearmente à medida que o PIB per capita do país aumenta. Portanto, podemos decidir modelar a satisfação com a vida como uma função linear do PIB per capita. Esta etapa é chamada de seleção de modelo: selecionamos um modelo linear de satisfação com a vida com apenas um atributo, PIB per capita.
- Equação de um modelo linear simples:
  - $satisfação\_vida = \theta_0 + \theta_1 \times PIB\_per\_capita$

Model-Based Learning

- Este modelo possui dois parâmetros de modelo, θ_0 e θ_1. Ajustando esses parâmetros, podemos fazer o modelo representar qualquer função linear.



Model-Based Learning

- Antes de usar o modelo, precisamos definir os valores dos parâmetros θ_0 e θ_1. Como podemos saber quais valores farão o modelo ter o melhor desempenho?
- Para responder a esta pergunta, precisamos especificar uma medida de desempenho. Podemos definir uma função de utilidade (ou função de adequação) que mede o quão bom é o modelo, ou podemos definir uma função de custo que mede o quão ruim ele é. Para problemas de regressão linear, as pessoas normalmente usam 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.

Model-Based Learning

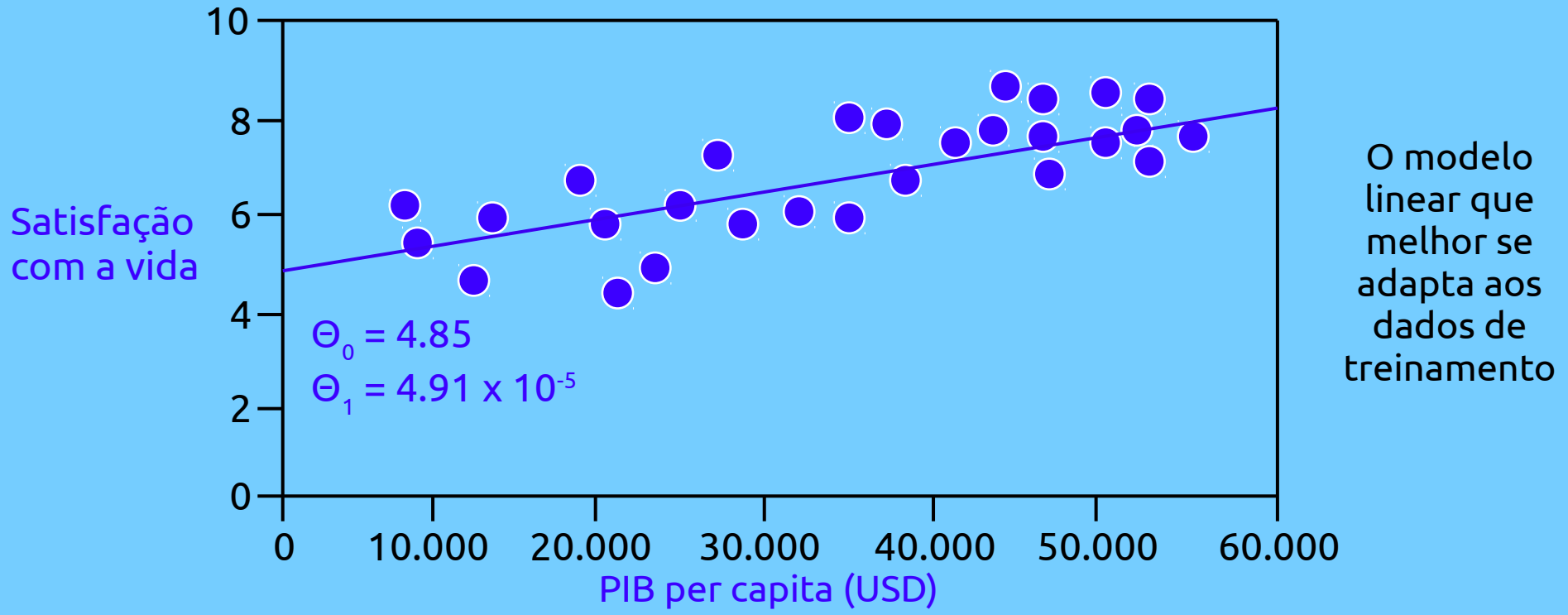
- É aqui que entra o algoritmo de regressão linear: você o alimenta com seus exemplos de treinamento e ele encontra os parâmetros que fazem o modelo linear se ajustar melhor aos seus dados.
- Isso é chamado de **treinamento do modelo**.
- Em nosso caso, o algoritmo descobre que os valores dos parâmetros ideais são $\theta_0 = 4.85$ e $\theta_1 = 4.91 \times 10^{-5}$.

A Palavra “Modelo”

- Surpreendentemente, a mesma palavra "modelo" pode se referir a um tipo de modelo (por exemplo, regressão linear), a uma arquitetura de modelo totalmente especificada (por exemplo, regressão linear com um input e um output) ou ao modelo final treinado pronto para ser usado para previsões (por exemplo, regressão linear com uma entrada e uma saída, usando $\theta_0 = 4.85$ e $\theta_1 = 4.91 \times 10^{-5}$). A seleção do modelo consiste na escolha do tipo de modelo e na especificação completa de sua arquitetura. Treinar um modelo significa executar um algoritmo para encontrar os parâmetros do modelo que o tornarão mais adequado aos dados de treinamento (e, com sorte, fazer boas previsões sobre novos dados).

Model-Based Learning

- Agora, o modelo se ajusta aos dados de treinamento o mais próximo possível (para um modelo linear).



Model-Based Learning

- Finalmente estamos prontos para executar o modelo para fazer previsões.
- Por exemplo, digamos que você queira saber o quanto os cipriotas estão felizes e os dados da OCDE não tenham a resposta.
- Felizmente, você pode usar seu modelo para fazer uma boa previsão: você procura o PIB per capita de Chipre, encontra $22.587 e, em seguida, aplica seu modelo e descobre que a satisfação com a vida provavelmente está em torno de $4.85 + 22.587 \times 4.91 \times 10^{-5} = 5.96$.

Sumarizando

- Em resumo temos então que:
  - Estudamos os dados.
  - Selecionamos um modelo.
  - Treinamos com os dados de treinamento (ou seja, o algoritmo de aprendizagem pesquisou os valores dos parâmetros do modelo que minimizam uma função de custo).
  - Finalmente, aplicamos o modelo para fazer previsões sobre novos casos (isso é chamado de inferência), esperando que esse modelo generalize bem.
- É assim que se parece um típico projeto de Machine Learning.

Desafios de Machine Learning

- Resumindo, como sua tarefa principal é selecionar um algoritmo de aprendizado e treiná-lo com alguns dados, as duas coisas que podem dar errado são “algoritmo ruim” e “dados ruins”. Por exemplo:
  - Quantidade insuficiente de dados de treinamento;
  - Dados de treinamento não representativos;
  - Dados de baixa qualidade;
  - Features irrelevantes;
  - Overfitting dos dados de treinamento;
  - Underfitting dos dados de treinamento;
  - Incompatibilidade de dados;

Quantidade Insuficiente de Dados

- Para uma criança aprender o que é uma maçã, basta você apontar para uma maçã e dizer “maçã” (possivelmente repetindo este procedimento algumas vezes). Agora a criança é capaz de reconhecer maçãs em todos os tipos de cores e formas.
- O Machine Learning ainda não chegou lá; são necessários muitos dados para que a maioria dos algoritmos de Machine Learning funcione corretamente.
- Mesmo para problemas muito simples, você normalmente precisa de milhares de exemplos, e para problemas complexos, como imagem ou reconhecimento de voz, você pode precisar de milhões de exemplos (a menos que você possa reutilizar partes de um modelo existente).

The Unreasonable Effectiveness of Data

- Em um famoso artigo publicado em 2001, os pesquisadores da Microsoft Michele Banko e Eric Brill mostraram que algoritmos de Machine Learning muito diferentes, incluindo alguns bastante simples, tiveram um desempenho quase idêntico em um problema complexo de desambiguação de linguagem natural, uma vez que receberam dados suficientes.
- A ideia de que os dados importam mais do que algoritmos para problemas complexos foi posteriormente popularizada por Peter Norvig et al. em um artigo intitulado “The Unreasonable Effectiveness of Data”, publicado em 2009. Deve-se notar, no entanto, que conjuntos de dados de pequeno e médio porte ainda são muito comuns e nem sempre é fácil ou barato obter dados extras de treinamento. Então não abandone os algoritmos ainda.

Dados Não Representativos

- Para generalizar bem, é crucial que seus dados de treinamento sejam representativos dos novos casos para os quais você deseja generalizar. Isso é verdade quer você use o instance-based learning ou o model-based learning.
- É crucial usar um conjunto de treinamento que seja representativo dos casos para os quais você deseja generalizar. Isso geralmente é mais difícil do que parece: se a amostra for muito pequena, você terá ruído de amostragem (ou seja, dados não representativos como resultado do acaso), mas mesmo as amostras muito grandes podem ser não representativas se o método de amostragem for falho.
- Isso é chamado de sampling bias.

Dados de Baixa Qualidade

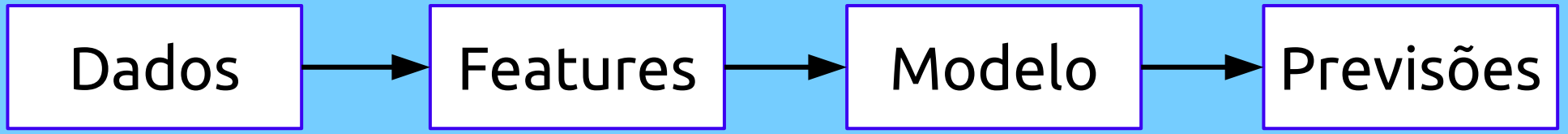
- Obviamente, se seus dados de treinamento estiverem cheios de erros, outliers e ruído (por exemplo, devido a medições de baixa qualidade), será mais difícil para o sistema detectar os padrões subjacentes, portanto, é menos provável que seu sistema tenha um bom desempenho.
- Geralmente, vale a pena gastar tempo limpando seus dados de treinamento. A verdade é que a maioria dos cientistas de dados gasta uma parte significativa de seu tempo fazendo exatamente isso.

Dados de Baixa Qualidade

- A seguir estão alguns exemplos de quando você deseja limpar os dados de treinamento:
  - Se algumas instâncias forem claramente discrepantes, pode ser útil simplesmente descartá-las ou tentar corrigir os erros manualmente.
  - Se algumas instâncias estão faltando alguns features (por exemplo, 5% de seus clientes não especificaram sua idade), você deve decidir se deseja ignorar esse atributo completamente, ignorar essas instâncias, preencher os valores ausentes (por exemplo, com a idade mediana) ou treine um modelo com o feature e um modelo sem ele.

Features Irrelevantes

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



Features Irrelevantes

- Este processo, chamado de feature engineering, envolve o seguintes passos:
  - Seleção de features (selecionar os features mais úteis para treinar entre os features existentes);
  - Extração de features (combinando features existentes para produzir um mais útil - como vimos anteriormente, algoritmos de redução de dimensionalidade podem ajudar);
  - Criação de novos features por meio da coleta de novos dados;

Overfitting

- Modelos complexos, como deep neural networks, podem detectar padrões sutis nos dados, mas se o conjunto de treinamento for barulhento ou muito pequeno (o que introduz ruído de amostragem), o modelo provavelmente detectará padrões no próprio ruído. Obviamente, esses padrões não serão generalizados para novas instâncias.
- O overfitting ocorre quando o modelo é muito complexo em relação à quantidade e ao ruído dos dados de treinamento.

Overfitting

- Possíveis soluções para o overfitting:
  - Simplifique o modelo selecionando um 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.
  - Reúna mais dados de treinamento.
  - Reduza o ruído nos dados de treinamento (por exemplo, corrija erros de dados e remova outliers).

Overfitting

- 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 a altura (θ_0) e a inclinação (θ_1) da linha. Se forçarmos $\theta_1 = 0$, o algoritmo terá apenas um grau de liberdade e terá muito mais dificuldade em ajustar os dados corretamente: tudo o que ele pode fazer é mover a linha para cima ou para baixo para chegar o mais próximo possível das instâncias de treinamento , por isso acabaria em torno da média. Um modelo muito simples! Se permitirmos que o algoritmo modifique θ_1, mas o forçarmos a mantê-lo pequeno, o algoritmo de aprendizado terá efetivamente algo entre um e dois graus de liberdade. Ele produzirá um modelo mais simples do que um com dois graus de liberdade, mas mais complexo do que um com apenas um. Você deseja encontrar o equilíbrio certo entre ajustar os dados de treinamento perfeitamente e manter o modelo simples o suficiente para garantir que ele generalize bem.

Overfitting

- 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 aprendizagem (não do modelo).
- Como tal, não é afetado pelo próprio algoritmo de aprendizagem; deve ser definido antes do treinamento e permanece constante durante o treinamento. Se você definir a regularização de hiperparâmetro para um valor muito grande, você obterá um modelo quase plano (uma inclinação próxima de zero); o algoritmo de aprendizado quase certamente não superestimará os dados de treinamento, mas será menos provável que encontre uma boa solução.
- O ajuste de hiperparâmetros é uma parte importante da construção de um sistema de Machine Learning.

Underfitting

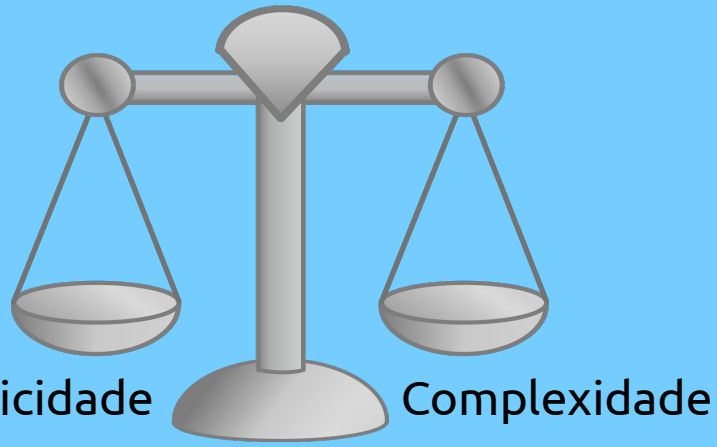
- Como você pode imaginar, underfitting é o oposto de overfitting: ele ocorre quando seu modelo é muito simples para aprender a estrutura subjacente dos dados. Por exemplo, um modelo linear de satisfação com a vida tende a ser insuficiente; a realidade é apenas mais complexa do que o modelo, portanto, suas previsões tendem a ser imprecisas, mesmo nos exemplos de treinamento.
- Aqui estão as principais opções para corrigir esse problema:
  - Selecione um modelo mais poderoso, com mais parâmetros.
  - Alimente melhores features para o algoritmo de aprendizagem (feature engineering).
  - Reduza as restrições no modelo (por exemplo, reduza o hiperparâmetro de regularização).

Revisando

- Vamos agora revisar os conceitos mais importantes de Machine Learning para adquirirmos uma visão ampla:
  - Machine Learning é fazer com que as máquinas se tornem melhores em algumas tarefas, aprendendo a partir dos dados, em vez de termos que codificar regras explicitamente.
  - Existem muitos tipos diferentes de sistemas de ML: supervised ou não, em batch ou online, instance-based ou model-based.
  - Em um projeto de ML, você reúne dados em um conjunto de treinamento e alimenta o conjunto de treinamento em um algoritmo de aprendizado. Se o algoritmo for model-based, ele ajustará alguns parâmetros para ajustar o modelo ao conjunto de treinamento (ou seja, para fazer boas previsões no próprio conjunto de treinamento), e então, esperançosamente, será capaz de fazer boas previsões também em novos casos. Se o algoritmo for instance-based, ele apenas aprende os exemplos de cor e generaliza para novas instâncias usando uma medida de similaridade para compará-los com as instâncias aprendidas.

Revisando

- O sistema não funcionará bem se o seu conjunto de treinamento for muito pequeno ou se os dados não forem representativos, forem ruidosos ou estiverem poluídos com features irrelevantes (entrada de lixo, saída de lixo). Por último, o seu modelo não precisa ser nem muito simples (nesse caso, será underfit) nem muito complexo (nesse caso, ele ficará excessivamente overfit).



Testando e Validando

- A única maneira de saber quão bem um modelo irá generalizar para novos casos é realmente testá-lo em novos casos. Uma maneira de fazer isso é colocar seu modelo em produção e monitorar seu desempenho. Isso funciona bem, mas se o seu modelo for terrivelmente ruim, os usuários reclamarão - o que não é a melhor ideia.
- A melhor opção é dividir seus dados em dois conjuntos: o conjunto de treinamento e o conjunto de teste. Como esses nomes indicam, você treina seu 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 out-of-sample error) e, avaliando seu modelo no conjunto de teste, você obtém uma estimativa desse erro. Este valor informa o quão bem o seu modelo executará em instâncias que ele nunca viu antes.

Testando e Validando

- 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, significa que seu modelo está overfitting aos dados de treinamento.
- É comum usar 80% dos dados para treinamento e reservar 20% para teste.
- No entanto, isso depende do tamanho do conjunto de dados: se ele contiver 10 milhões de instâncias, manter 1% significa que seu conjunto de teste conterá 100.000 instâncias, provavelmente mais do que o suficiente para obter uma boa estimativa do erro de generalização.

Ajuste de Hiperparâmetros e Seleção de Modelo

- Avaliar um modelo é bastante simples: basta usar um conjunto de teste.
- Mas suponha que você esteja hesitando entre dois tipos de modelos (digamos, um modelo linear e um modelo polinomial): como você pode decidir entre eles? Uma opção é treinar ambos e comparar quão bem eles generalizam usando o conjunto de teste.
- Agora suponha que o modelo linear generalize melhor, mas você deseja aplicar alguma regularização para evitar overfitting. A questão é: como você escolhe o valor do hiperparâmetro de regularização?
- Uma opção é 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, apenas 5% de erro. Você coloca este modelo em produção, mas infelizmente ele não funciona tão bem quanto o esperado e produz 15% de erros. O que acabou de acontecer?

Ajuste de Hiperparâmetros e Seleção de Modelo

- O problema é que você mediu o erro de generalização várias vezes no conjunto de teste e adaptou o modelo e os hiperparâmetros para produzir o melhor modelo para aquele conjunto específico. Isso significa que é improvável que o modelo tenha um desempenho tão bom com novos dados.
- Uma solução comum para esse problema é chamada holdout validation: você simplesmente exibe parte do conjunto de treinamento para avaliar vários modelos candidatos e selecionar o melhor. O novo conjunto retido é chamado de conjunto de validação (ou às vezes o conjunto de desenvolvimento, ou conjunto dev). Mais especificamente, você treina vários modelos com vários hiperparâmetros no conjunto de treinamento reduzido (ou seja, o conjunto de treinamento completo menos o conjunto de validação) e seleciona o modelo com melhor desempenho no conjunto de validação. Após esse processo de validação de validação, você treina o melhor modelo no conjunto de treinamento completo (incluindo o conjunto de validação) e isso fornece o modelo final. Por último, você avalia este modelo final no conjunto de testes para obter uma estimativa do erro de generalização.

Ajuste de Hiperparâmetros e Seleção de Modelo

- Essa solução geralmente funciona muito bem. No entanto, se o conjunto de validação for muito pequeno, as avaliações do modelo serão imprecisas: você pode acabar selecionando um modelo abaixo do ideal por engano. Por outro lado, se o conjunto de validação for muito grande, o conjunto de treinamento restante será muito menor do que o conjunto de treinamento completo. Por que isso é ruim? Bem, como o modelo final será treinado no conjunto de treinamento completo, não é ideal comparar os modelos candidatos treinados em um conjunto de treinamento muito menor. Uma maneira de resolver esse problema é realizar cross-validation repetida, usando muitos pequenos conjuntos de validação.
- Cada modelo é avaliado uma vez por conjunto de validação após ser treinado no restante dos dados. Calculando a média de todas as avaliações de um modelo, você obtém uma medida muito mais precisa de seu desempenho. Porém, há uma desvantagem: o tempo de treinamento é multiplicado pelo número de conjuntos de validação.

Incompatibilidade de Dados

- Em alguns casos, é fácil obter uma grande quantidade de dados para treinamento, mas esses dados provavelmente não serão perfeitamente representativos dos dados que serão usados na produção. Por exemplo, suponha que você queira criar um aplicativo móvel para tirar fotos de flores e determinar automaticamente suas espécies. Você pode facilmente baixar milhões de fotos de flores na web, mas elas não serão perfeitamente representativas das fotos que realmente serão tiradas usando o aplicativo em um dispositivo móvel. Talvez você tenha apenas 10.000 fotos representativas (ou seja, realmente tiradas com o aplicativo).
- Nesse caso, a regra mais importante a lembrar é que o conjunto de validação e o conjunto de teste devem ser tão representativos quanto possível dos dados que você espera usar na produção, então eles devem ser compostos exclusivamente de imagens representativas: você pode embaralhá-los e colocá-los metade no conjunto de validação e metade no conjunto de teste (certificando-se de que nenhuma duplicata ou quase duplicata acabe em ambos os conjuntos).

Incompatibilidade de Dados

- Mas depois de treinar seu modelo nas imagens da web, se você observar que o desempenho do modelo no conjunto de validação é decepcionante, você não saberá se isso é porque seu modelo overfitted ao conjunto de treinamento ou se isso é apenas devido a incompatibilidade entre as imagens da web e as imagens do aplicativo móvel. Uma solução é exibir algumas das fotos de treinamento (da web) em outro conjunto que Andrew Ng chama de conjunto train-dev.
- Depois que o modelo é treinado (no conjunto de treinamento, não no conjunto train-dev), você pode avaliá-lo no conjunto train-dev. Se tiver um bom desempenho, o modelo não está overfitting ao conjunto de treinamento. Se o desempenho for ruim no conjunto de validação, o problema deve estar na incompatibilidade de dados. Você pode tentar resolver esse problema pré-processando as imagens da web para torná-las mais parecidas com as fotos que serão tiradas pelo aplicativo móvel e, em seguida, treinando novamente o modelo. Por outro lado, se o modelo tiver um desempenho insatisfatório no conjunto train-dev, ele deve ter overfitted ao conjunto de treinamento, então você deve tentar simplificar ou regularizar o modelo, obter mais dados de treinamento e limpar os dados de treinamento.

No Free Lunch Theorem

- Um modelo é uma versão simplificada das observações.
- As simplificações têm o objetivo de descartar os detalhes supérfluos que provavelmente não serão generalizados para novas instâncias. Para decidir quais dados descartar e quais dados manter, você deve fazer suposições. Por exemplo, um modelo linear pressupõe que os dados são fundamentalmente lineares e que a distância entre as instâncias e a linha reta é apenas ruído, que pode ser ignorado com segurança.

No Free Lunch Theorem

- Em um famoso artigo de 1996, David Wolpert demonstrou que, se você não fizer absolutamente nenhuma suposição sobre os dados, não há razão para preferir um modelo a qualquer outro. Isso é chamado de teorema No Free Lunch (NFL). Para alguns conjuntos de dados, o melhor modelo é um modelo linear, enquanto para outros conjuntos de dados é uma rede neural. Não há modelo que é garantido a priori para funcionar melhor (daí o nome do teorema). A única maneira de saber com certeza qual é o melhor modelo é avaliar todos eles. Como isso não é possível, na prática você faz algumas suposições razoáveis sobre os dados e avalia apenas alguns modelos razoáveis. Por exemplo, para tarefas simples, você pode avaliar modelos lineares com vários níveis de regularização, e para um problema complexo, você pode avaliar várias redes neurais.

Machine Learning na Prática

- Os algoritmos de Machine Learning são apenas uma pequena parte do uso do ML na prática como analista ou cientista de dados. Na prática, o processo geralmente se parece com:
- Iniciar Loop ∞

  1. Compreenda o domínio, conhecimentos prévios e objetivos. Fale com especialistas de domínio. Freqüentemente, os objetivos não são claros. Geralmente, você tem mais coisas para tentar do que possivelmente implementar.

  2. Integração, seleção, limpeza e pré-processamento de dados. Geralmente, essa é a parte mais demorada. É importante ter dados de alta qualidade. Quanto mais dados você tem, mais difícil, porque os dados estão possivelmente sujos. Entra lixo, sai lixo.

  3. Modelos de aprendizagem. A parte divertida.

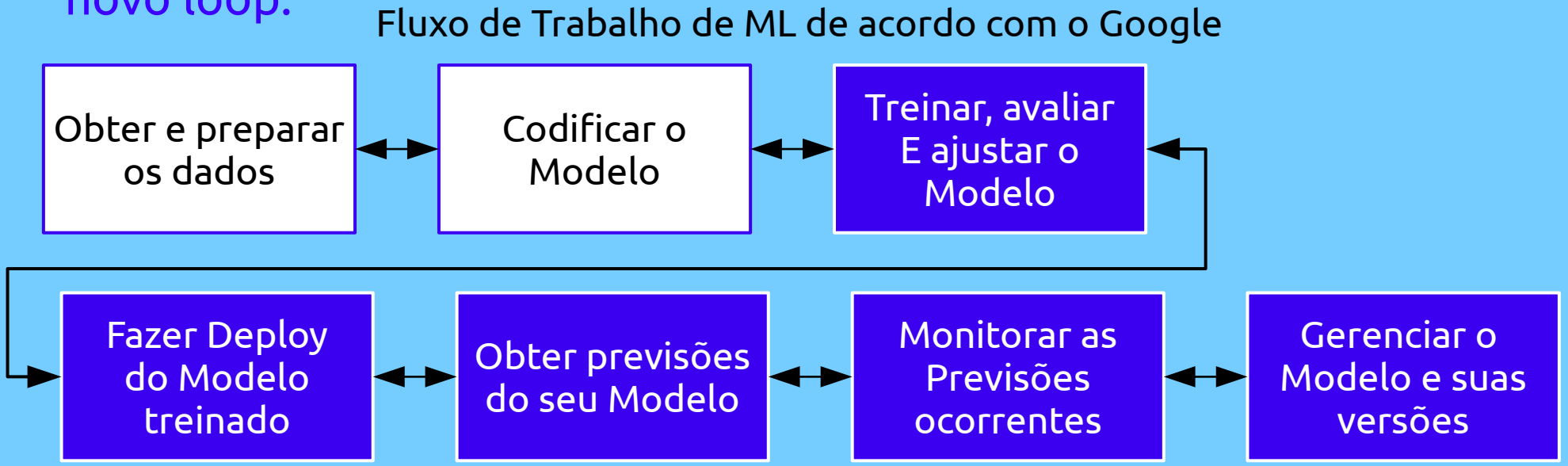
  4. Interpretando resultados. Às vezes, não importa como o modelo funciona, desde que forneça resultados. Outros domínios exigem que o modelo seja compreensível. Você será desafiado por especialistas humanos.

  5. Consolidar e implantar o conhecimento descoberto. A maioria dos projetos bem-sucedidos no laboratório não é usada na prática.

- Finalizar Loop ∞

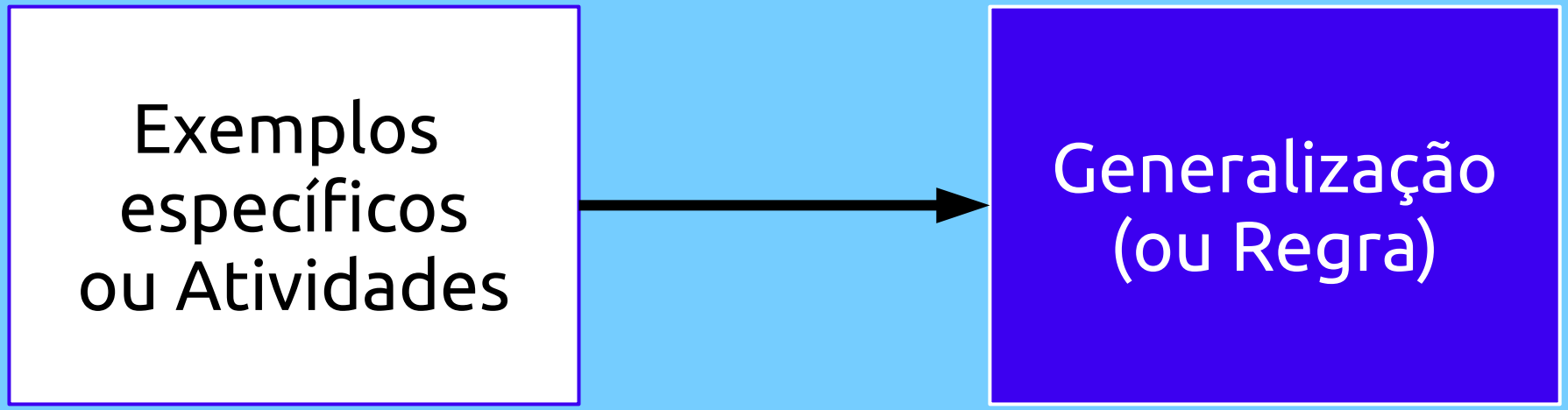
Machine Learning na Prática

- Machine Learning não é um processo único, é um ciclo.
- Você precisa executar o loop até obter um resultado que possa ser usado na prática. Além disso, os dados podem mudar, exigindo um novo loop.



Aprendizagem Indutiva

- É importante também abordamos o tópico de aprendizagem indutiva.
- Esta é a teoria geral por trás de supervised learning.



Aprendizagem Indutiva

- Da perspectiva da aprendizagem indutiva, recebemos amostras de entrada (x) e amostras de saída (f(x)) e o problema é estimar a função (f).
- Especificamente, o problema é generalizar a partir das amostras e do mapeamento para ser útil para estimar a saída de novas amostras no futuro.
- Na prática, quase sempre é muito difícil estimar a função, por isso estamos procurando por aproximações muito boas da função.

Uma função $f: X \rightarrow Y$ é **consistente com** um conjunto de exemplos de treinamento rotulados S se e somente se $f(x) = y$ para todos os $(x, y) \in S$.

Aprendizagem Indutiva

Objetivo Principal (Formal)

Dado um grande número suficiente de exemplos de treinamento,
e um espaço de hipótese H,
aprenda uma hipótese $h \in H$ que se aproxime de $f(\cdot)$

- Nossa esperança: Existe um $h \in H$ que se aproxima de f.
- Nossa estratégia: Esperamos que nossos exemplos de treinamento são representativos da realidade.
  - Nós vimos muitos exemplos.
  - Eles foram selecionados sem qualquer viés.
  - Qualquer hipótese que seja consistente com os dados de treinamento seria precisa em instâncias não observadas.

Aprendizagem Indutiva

- Alguns exemplos práticos de indução são:
- Avaliação de risco de crédito:
  - O x são as propriedades do cliente.
  - O f(x) é o crédito aprovado ou não.
- Diagnóstico de doenças:
  - O x são as propriedades do paciente.
  - O f(x) é a doença de que sofrem.
- Reconhecimento facial:
  - O x são bitmaps de rostos de pessoas.
  - O f(x) é para atribuir um nome ao rosto.
- Direção automática:
  - O x são imagens de bitmap de uma câmera na frente do carro.
  - O f(x) é o grau em que o volante deve ser girado.

Essência da Aprendizagem Indutiva

- Podemos escrever um programa que funcione perfeitamente com os dados que temos.
- Esta função será ajustada ao máximo. Mas não temos ideia de como funcionará bem com novos dados, provavelmente será muito ruim porque talvez nunca mais veremos os mesmos exemplos.
- Os dados não são suficientes. Você pode prever o que quiser. E isso seria ingênuo não assumir nada sobre o problema.

Essência da Aprendizagem Indutiva

- Na prática, não somos ingênuos. Existe um problema subjacente e estamos interessados em uma aproximação precisa da função.
- Há um número exponencial duplo de classificadores possíveis no número de estados de entrada. Encontrar uma boa aproximação para a função é muito difícil.
- Existem classes de hipóteses que podemos experimentar. Essa é a forma que pode assumir a solução ou a representação. Não podemos saber de antemão qual é o mais adequado para o nosso problema.
- Precisamos usar a experimentação para descobrir o que funciona no problema.

Essência da Aprendizagem Indutiva

- Duas perspectivas de aprendizagem indutiva:
  - Aprender é a remoção da incerteza. Ter dados remove algumas incertezas. Selecionando uma classe de hipóteses, estamos removendo mais incertezas.
  - Aprender é adivinhar uma classe de hipóteses boa e pequena. Requer adivinhação. Não sabemos a solução, devemos usar um processo de tentativa e erro. Se você conhece o domínio com certeza, não precisa aprender. Mas não estamos adivinhando no escuro.

Podemos estar errados:

Nosso conhecimento prévio pode estar errado.

Nossa suposição da classe de hipóteses pode estar errada.

Principais Problemas de Machine Learning

- O que é um bom espaço de hipótese?
- Quais algoritmos funcionam com esse espaço?
- O que posso fazer para otimizar a accuracy de dados não vistos?
- Como podemos ter confiança no modelo?
- Existem problemas de aprendizagem que são intratáveis computacionalmente?
- Como podemos formular problemas de aplicações como problemas de Machine Learning?

Terminologia de Machine Learning

- Training example: uma amostra de x incluindo sua saída da função de destino.
- Target function: a função de mapeamento f de x para f(x).
- Hipótese: aproximação de f, uma função candidata.
- Classificador: o programa de aprendizagem produz um classificador que pode ser usado para classificar.
- Aprendizagem: processo que cria o classificador.
- Espaço de hipóteses: conjunto de aproximações possíveis de f que o algoritmo pode criar.
- Espaço de versão: subconjunto do espaço de hipótese que é consistente com os dados observados.

Terminologia de Machine Learning

- Labels: um label é o que estamos prevendo - a variável y na regressão linear simples. O label (rótulo) pode ser o preço futuro do trigo, o tipo de animal mostrado em uma imagem, o significado de um clipe de áudio ou qualquer coisa.
- Features: um feature é uma variável de entrada - a variável x na regressão linear simples. Um projeto simples de Machine Learning pode usar um único feature, enquanto um projeto de Machine Learning mais sofisticado pode usar milhões de features, especificados como:

$$x_1, x_2, x_3, \ldots, x_n$$

- No exemplo do detector de spam, os features 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".

Terminologia de Machine Learning

- Um exemplo é uma instância particular de dados, **x**. (Colocamos x em negrito para indicar que é um vetor.) Dividimos os exemplos em duas categorias:
  - Exemplos com labels
  - Exemplos sem labels
- Um exemplo com labels inclui ambos feature(s) e o label, ou seja:

Exemplos com label: {features, label}: (x, y)

Use exemplos com labels para treinar o modelo. Em nosso exemplo de detector de spam, os exemplos marcados seriam emails individuais que os usuários marcaram explicitamente como "spam" ou "não spam".

Terminologia de Machine Learning

- Por exemplo, a tabela a seguir mostra 5 exemplos com labels de um conjunto de dados contendo informações sobre os preços de imóveis na Califórnia:

Idade Média da casa	Salas Totais	Quartos Totais	Banheiros	Valor da Casa
15	5	5	4	80.700
19	8	6	4	85.000
17	3	2	1	66.900
14	9	3	2	73.400
20	2	3	2	70.500

Terminologia de Machine Learning

- Um exemplo sem labels contem features, mas não contém o label, ou seja:

Exemplos sem label: {features, ?}: (x, ?)

Aqui estão três exemplos sem labels do mesmo conjunto de dados de habitação, que excluem Valor da Casa:

Idade Média da casa	Salas Totais	Quartos Totais	Banheiros
10	2	3	3
22	3	4	3
13	5	2	1

Depois de treinar nosso modelo com exemplos com labels, usamos esse modelo para prever o label de exemplos sem labels.

Modelos

- Um modelo define a relação entre features e labels.
- Por exemplo, um modelo de detecção de spam pode associar determinados features fortemente a "spam". Vamos destacar duas fases da vida de um modelo:
  - Treinar significa criar ou aprender o modelo. Ou seja, você mostra os exemplos com labels ao modelo e permite que o modelo aprenda gradualmente as relações entre os features e labels.
  - Inferência significa aplicar o modelo treinado a exemplos sem labels. Ou seja, você usa o modelo treinado para fazer previsões úteis (y'). Por exemplo, durante a inferência, você pode prever Valor da Casa para novos exemplos sem labels.

O Algoritmo K-Nearest Neighbors

- O algoritmo k-Nearest Neighbors (k-NN) é um método de classificação não paramétrico desenvolvido pela primeira vez por Evelyn Fix e Joseph Hodges em 1951, e posteriormente expandido por Thomas Cover.
- É usado para classificação e regressão.
- Em ambos os casos, a entrada (input) consiste nos k exemplos de treinamento mais próximos no conjunto de dados.
- A saída (output) depende se k-NN é usado para classificação ou regressão.

O Algoritmo K-Nearest Neighbors

- Na classificação k-NN, a saída é uma associação de classe. Um objeto é classificado por uma pluralidade de votos de seus vizinhos, com o objeto sendo atribuído à classe mais comum entre seus k vizinhos mais próximos (k é um número inteiro positivo, tipicamente pequeno). Se k = 1, então o objeto é simplesmente atribuído à classe daquele único vizinho mais próximo.
- Na regressão k-NN, a saída é o valor da propriedade do objeto. Este valor é a média dos valores dos k vizinhos mais próximos.

O Algoritmo K-Nearest Neighbors

- k-NN é um tipo de classificação onde a função é aproximada apenas localmente e todos os cálculos são adiados até a avaliação da função. Como esse algoritmo depende da distância para classificação, se os features representarem unidades físicas diferentes ou vierem em escalas muito diferentes, a normalização dos dados de treinamento pode melhorar drasticamente sua precisão.
- Tanto para classificação quanto para regressão, uma técnica útil pode ser atribuir pesos às contribuições dos vizinhos, de modo que os vizinhos mais próximos contribuam mais para a média do que os mais distantes. Por exemplo, um esquema de ponderação comum consiste em dar a cada vizinho um peso de 1/d, onde d é a distância ao vizinho.

O Algoritmo K-Nearest Neighbors

- Os vizinhos são obtidos de um conjunto de objetos para os quais a classe (para classificação k-NN) ou o valor da propriedade do objeto (para regressão k-NN) é conhecido. Isso pode ser considerado o conjunto de treinamento para o algoritmo, embora nenhuma etapa de treinamento explícita seja necessária.
- Os exemplos de treinamento são vetores em um espaço de features multidimensional, cada um com um label de classe. A fase de treinamento do algoritmo consiste apenas em armazenar os vetores de features e labels de classe das amostras de treinamento.

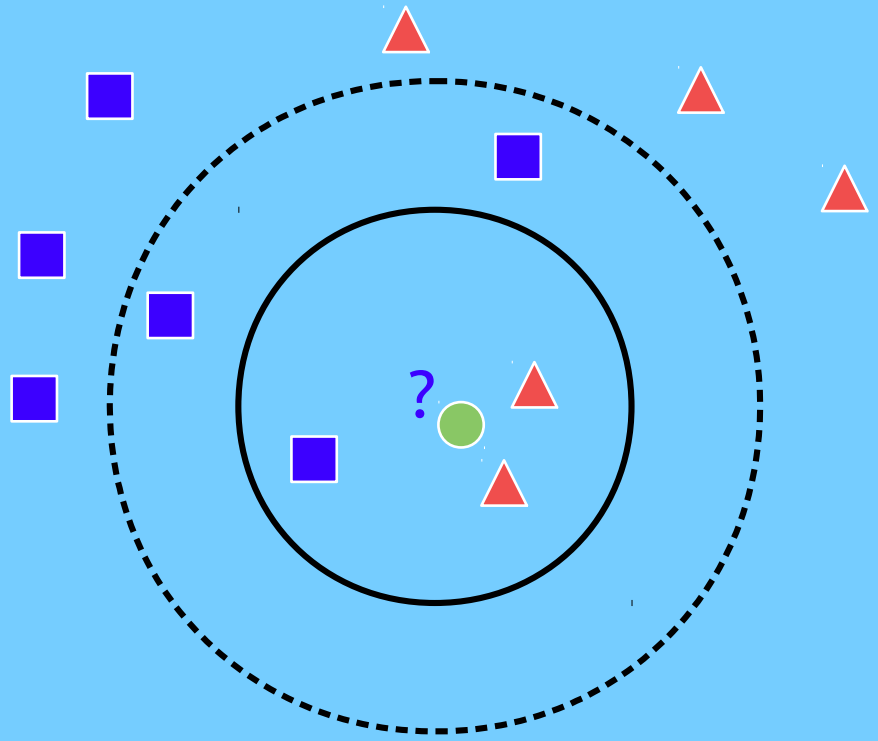
O Algoritmo K-Nearest Neighbors

- Na fase de classificação, k é uma constante definida pelo usuário, e um vetor sem labels (uma consulta ou ponto de teste) é classificado atribuindo o label que é mais frequente entre as k amostras de treinamento mais próximas desse ponto de consulta.
- Uma métrica de distância comumente usada para variáveis contínuas é a distância euclidiana. Para variáveis discretas, como para classificação de texto, outra métrica pode ser usada, como a overlap metric (ou distância de Hamming).
- No contexto de dados de microarray de expressão gênica, por exemplo, k-NN foi empregado com coeficientes de correlação, como Pearson e Spearman, como uma métrica. Freqüentemente, a precisão da classificação de k-NN pode ser melhorada significativamente se a métrica de distância for aprendida com algoritmos especializados, como a Large Margin Nearest Neighbor ou Neighbourhood components analysis.

O Algoritmo K-Nearest Neighbors

- A seguir temos um exemplo de classificação k-NN.
- A amostra de teste (ponto verde) deve ser classificada em quadrados roxos ou triângulos vermelhos. Se k = 3 (círculo de linha sólida), ele é atribuído aos triângulos vermelhos porque há 2 triângulos e apenas 1 quadrado dentro do círculo interno. Se k = 5 (círculo de linha tracejada), é atribuído aos quadrados roxos (3 quadrados vs 2 triângulos dentro do círculo externo).

O Algoritmo K-Nearest Neighbors



O Algoritmo K-Nearest Neighbors

- Uma desvantagem da classificação básica de "votação majoritária" ocorre quando a distribuição de classes é distorcida.
- Ou seja, exemplos de uma classe mais frequente tendem a dominar a previsão do novo exemplo, pois tendem a ser comuns entre os k vizinhos mais próximos devido ao seu grande número.
- Uma forma de contornar esse problema é ponderar a classificação, levando em consideração a distância do ponto de teste a cada um de seus k vizinhos mais próximos. A classe (ou valor, em problemas de regressão) de cada um dos k pontos mais próximos é multiplicada por um peso proporcional ao inverso da distância daquele ponto ao ponto de teste. Outra maneira de superar a distorção é por abstração na representação de dados.
- Por exemplo, em um self-organizing map (SOM), cada nó é um representante (um centro) de um cluster de pontos semelhantes, independentemente de sua densidade nos dados de treinamento originais. K-NN pode então ser aplicado ao SOM.

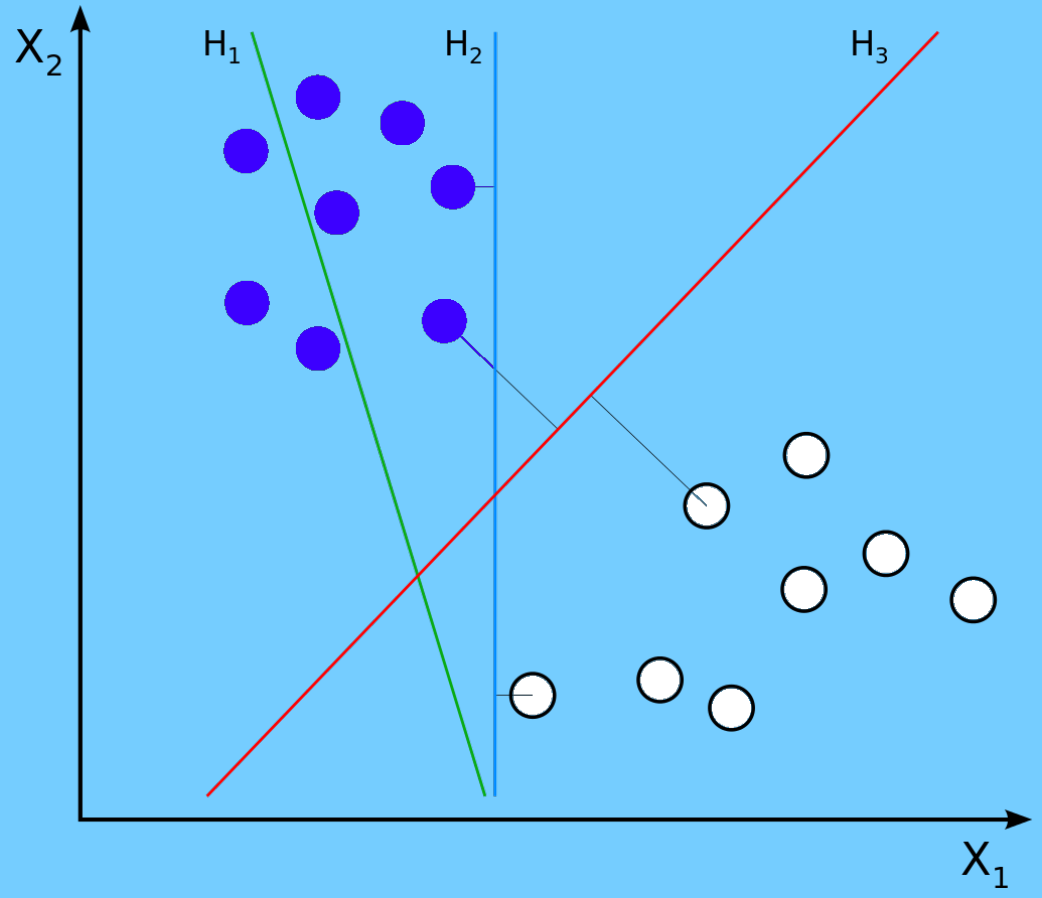
Support-vector Machine

- Em Machine Learning, support-vector machines (SVMs, também support-vector networks) são modelos de supervised learning com algoritmos de associated learning que analisam dados para classificação e análise de regressão.
- Desenvolvido na AT&T Bell Laboratories por Vladimir Vapnik com colegas (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), SVMs são um dos métodos de previsão mais robustos, sendo baseados em estruturas de aprendizagem estatística ou Teoria VC proposta por Vapnik (1982, 1995) e Chervonenkis (1974).
- Dado um conjunto de exemplos de treinamento, cada um marcado como pertencente a uma das duas categorias, um algoritmo de treinamento SVM constrói um modelo que atribui novos exemplos a uma categoria ou outra, tornando-o um classificador linear binário não probabilístico (embora métodos como Platt scaling existem para usar SVM em uma configuração de classificação probabilística).

Support-vector Machine

- O SVM mapeia exemplos de treinamento para pontos no espaço de modo a maximizar a largura da lacuna entre as duas categorias.
- Novos exemplos são mapeados nesse mesmo espaço e considerados pertencentes a uma categoria com base em qual lado da lacuna eles se enquadram.
- Além de realizar a classificação linear, os SVMs podem realizar com eficiência uma classificação não linear usando o que é chamado de kernel trick, mapeando implicitamente suas entradas em espaços de recursos de alta dimensão.

Support-vector Machine



H1 não separa as classes.

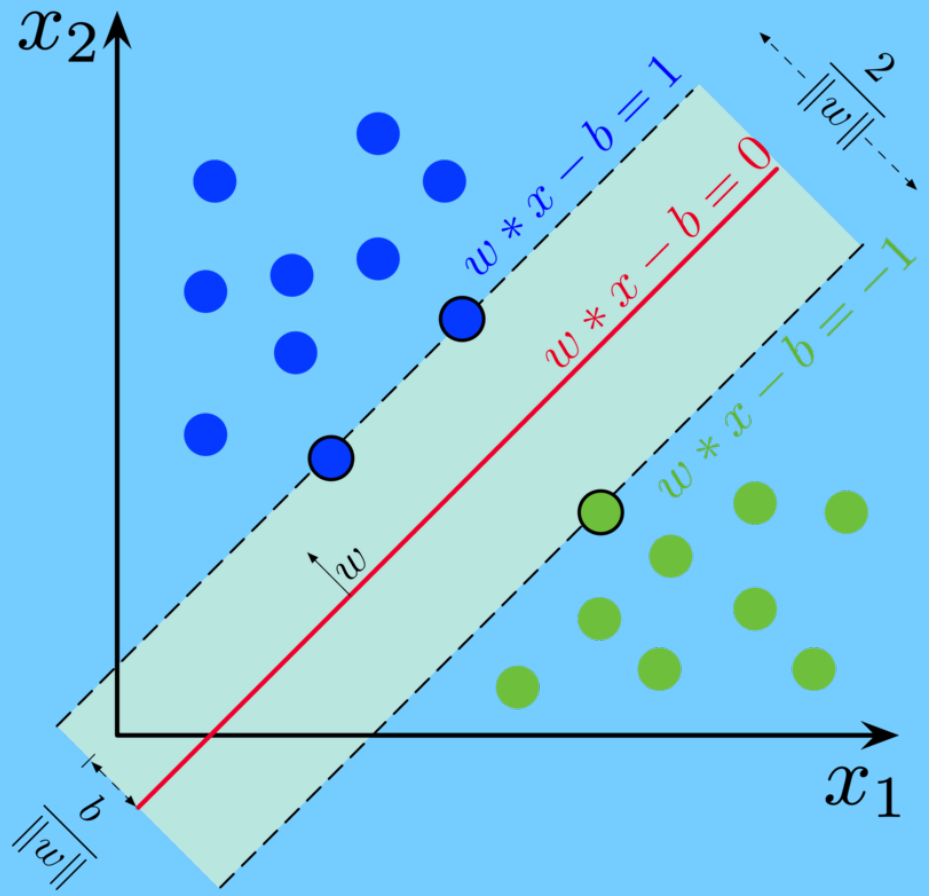
H2 separa, mas apenas com uma pequena margem.

H3 separa as classes com a margem máxima.

Support-vector Machine

- **Motivação do SVM**: Classificar dados é uma tarefa comum no Machine Learning. Suponha que alguns pontos de dados dados pertençam a uma de duas classes, e o objetivo é decidir em qual classe um novo ponto de dados estará. No caso de support-vector machines, um ponto de dados é visto como um vetor p-dimensional (uma lista de p números), e queremos saber se podemos separar tais pontos com um hiperplano (p-1)-dimensional. Isso é chamado de classificador linear. Existem muitos hiperplanos que podem classificar os dados. Uma escolha razoável como o melhor hiperplano é aquela que representa a maior separação, ou margem, entre as duas classes. Portanto, escolhemos o hiperplano de forma que a distância dele até o ponto de dados mais próximo em cada lado seja maximizada. Se tal hiperplano existe, ele é conhecido como maximum-margin hyperplane e o classificador linear que ele define é conhecido como um maximum-margin classifier; ou equivalentemente, o perceptron of optimal stability.

Support-vector Machine

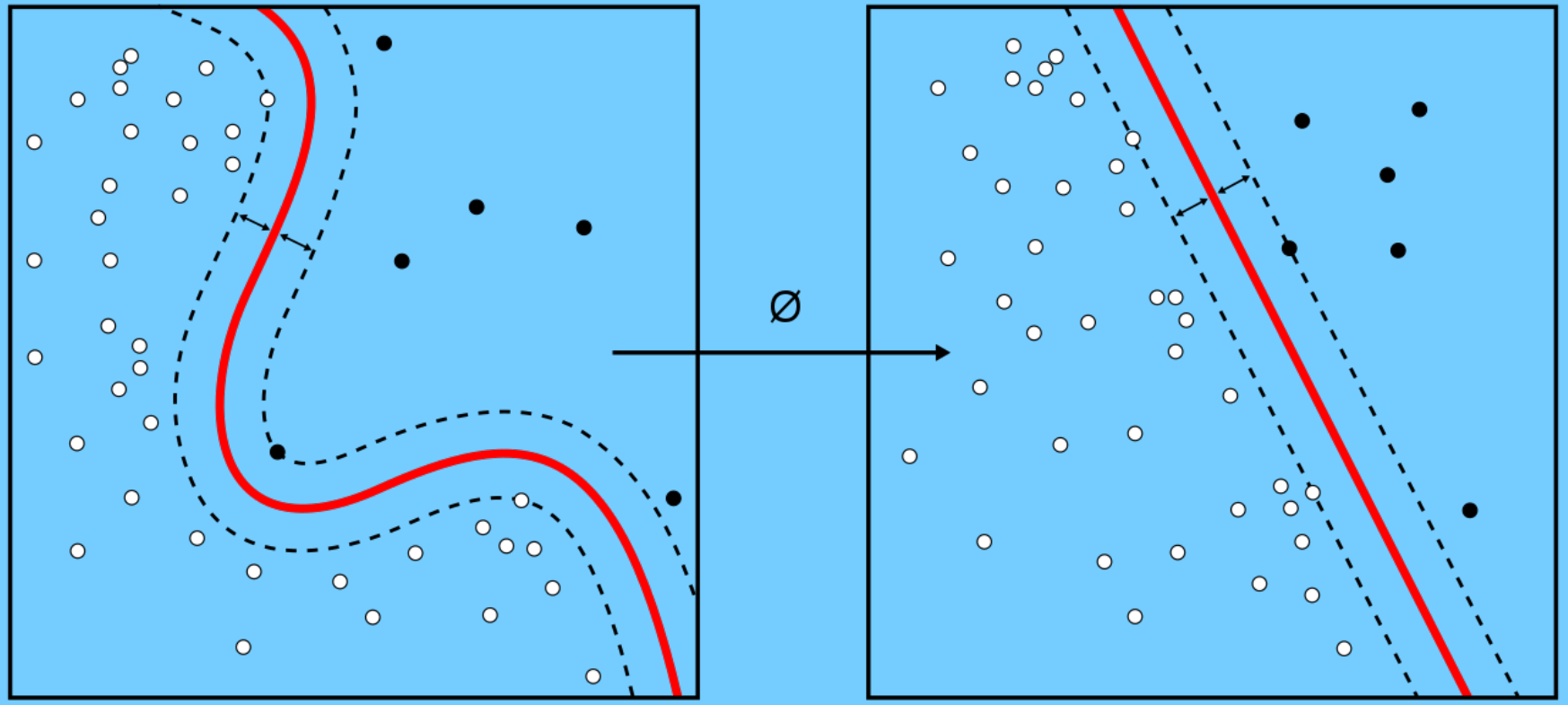


Maximum-margin hyperplane e margens para um SVM treinado com amostras de duas classes. As amostras na margem são chamadas de support vectors.

Support-vector Machine

- Mais formalmente, uma support-vector machine constrói um hiperplano ou conjunto de hiperplanos em um espaço de dimensão alta ou infinita, que pode ser usado para classificação, regressão ou outras tarefas como detecção de outliers.
- Intuitivamente, uma boa separação é obtida pelo hiperplano que possui a maior distância ao ponto de dados de treinamento mais próximo de qualquer classe (a chamada margem funcional), pois em geral quanto maior a margem, menor o erro de generalização do classificador.

Support-vector Machine



Kernel machine

Support-vector Machine

- Enquanto o problema original pode ser colocado em um espaço de dimensão finita, freqüentemente acontece que os conjuntos a discriminar não são linearmente separáveis naquele espaço.
- Por esta razão, foi proposto que o espaço de dimensão finita original fosse mapeado em um espaço de dimensão muito superior, provavelmente facilitando a separação naquele espaço.
- Para manter a carga computacional razoável, os mapeamentos usados pelos esquemas SVM são projetados para garantir que produtos escalares de pares de vetores de dados de entrada possam ser calculados facilmente em termos das variáveis no espaço original, definindo-as em termos de uma função kernel k(x, y) selecionado para se adequar ao problema.
- Os hiperplanos no espaço de dimensão superior são definidos como o conjunto de pontos cujo produto escalar com um vetor naquele espaço é constante, onde tal conjunto de vetores é um conjunto ortogonal (e, portanto, mínimo) de vetores que define um hiperplano.

Support-vector Machine

- Os vetores que definem os hiperplanos podem ser escolhidos como combinações lineares com parâmetros α_i de imagens de vetores de features x_i que ocorrem na base de dados.
- Com esta escolha de um hiperplano, os pontos x no espaço de features que são mapeados no hiperplano são definidos pela relação $\Sigma_i \alpha_i k(x_i,x) = \text{constante}$. Observe que se $k(x,y)$ se torna pequeno à medida que y se afasta de x, cada termo na soma mede o grau de proximidade do ponto de teste x ao ponto de base de dados correspondente.
- Desta forma, a soma dos kernels acima pode ser usada para medir a proximidade relativa de cada ponto de teste aos pontos de dados originados em um ou outro dos conjuntos a serem discriminados. Observe o fato de que o conjunto de pontos x mapeado em qualquer hiperplano pode ser bastante complicado como resultado, permitindo uma discriminação muito mais complexa entre conjuntos que não são convexos no espaço original.

Support-vector Machine

- Os SVMs podem ser usados para resolver vários problemas do mundo real:
  - Os SVMs são úteis na categorização de texto e hipertexto, pois sua aplicação pode reduzir significativamente a necessidade de instâncias de treinamento com labels em configurações indutivas e transdutivas padrão. Alguns métodos de shallow semantic parsing são baseados em support vector machines.
  - A classificação de imagens também pode ser realizada usando SVMs. Os resultados experimentais mostram que os SVMs alcançam uma accuracy de pesquisa significativamente maior do que os esquemas de refinamento de consulta tradicionais após apenas três a quatro rodadas de feedback de relevância. Isso também é verdadeiro para sistemas de segmentação de imagem, incluindo aqueles que usam uma versão modificada do SVM que usa a abordagem privilegiada sugerida por Vapnik.

Support-vector Machine

- Os SVMs podem ser usados para resolver vários problemas do mundo real:
  - Classificação de dados de satélite como dados synthetic-aperture radar (SAR) usando SVM supervisionado.
  - Os caracteres escritos à mão podem ser reconhecidos usando o SVM.
  - O algoritmo SVM tem sido amplamente aplicado nas ciências biológicas e outras ciências. Eles têm sido usados para classificar proteínas com até 90% dos compostos classificados corretamente. Testes de permutação baseados em pesos de SVM foram sugeridos como um mecanismo para interpretação de modelos de SVM. Pesos de support-vector machine também foram usados para interpretar modelos SVM no passado. A interpretação posthoc de modelos de support-vector machine para identificar features usados pelo modelo para fazer previsões é uma área de pesquisa relativamente nova com significado especial nas ciências biológicas.

Support-vector Clustering

- Quando os dados não estão com labels, o supervised learning não é possível e uma abordagem de unsupervised learning é necessária, que tenta encontrar o agrupamento natural dos dados para grupos e, em seguida, mapeia novos dados para esses grupos formados.
- O algoritmo de support-vector clustering, criado por Hava Siegelmann e Vladimir Vapnik, aplica a estatística de support vectors, desenvolvida no algoritmo de support vector machines, para categorizar dados sem labels, e é um dos algoritmos de agrupamento mais utilizados em aplicações industriais.

Decision Trees

- Uma árvore de decisão é uma ferramenta de suporte à decisão que usa um modelo de decisão semelhante a uma árvore e suas possíveis consequências, incluindo *chance event outcomes*, custos de recursos e utilidade.
- É uma maneira de exibir um algoritmo que contém apenas instruções de controle condicional.
- Árvores de decisão são comumente usadas em pesquisa operacional, especificamente na análise de decisão, para ajudar a identificar uma estratégia com maior probabilidade de atingir uma meta, mas também são uma ferramenta popular no Machine Learning.

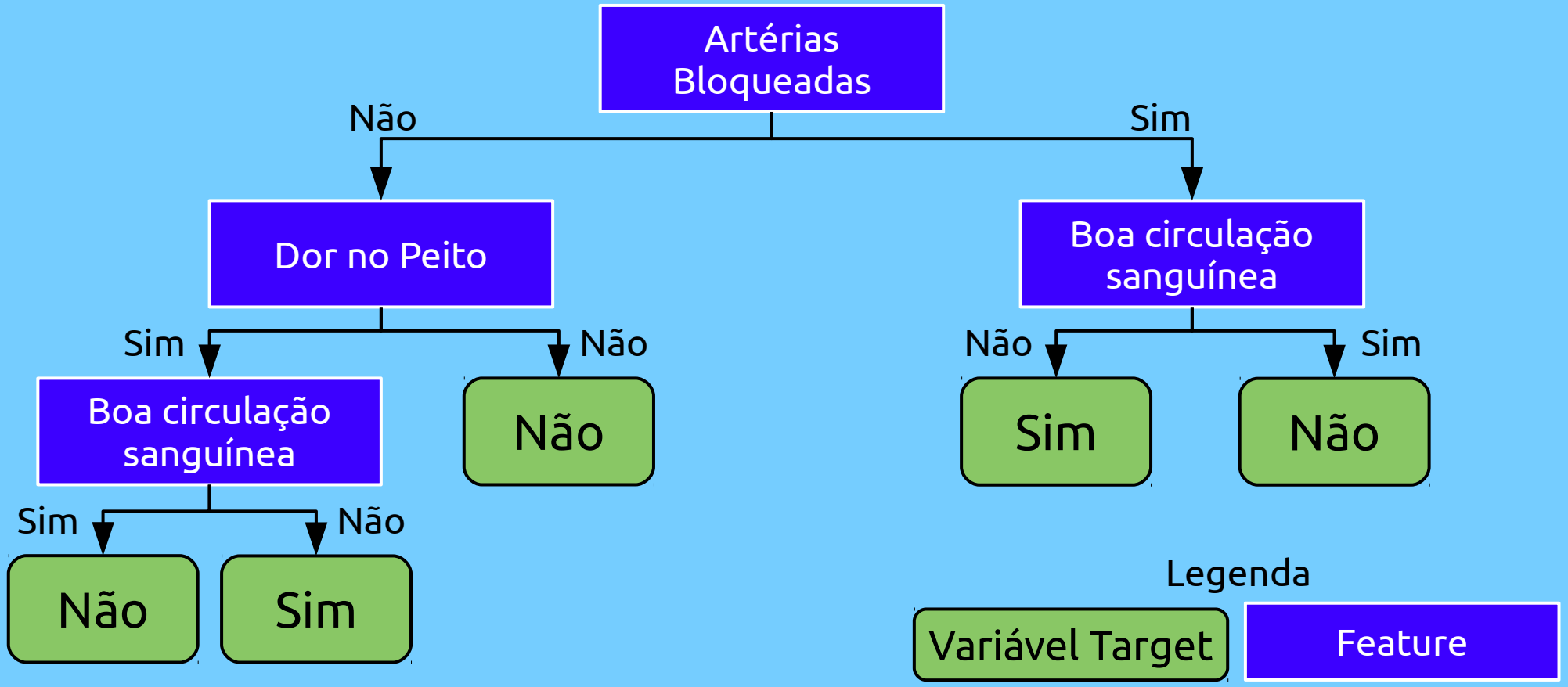
Decision Trees

- Uma árvore de decisão é uma estrutura semelhante a um fluxograma em que cada nó interno representa um "teste" em um atributo (por exemplo, se um lançamento de moeda dá cara ou coroa), cada ramo representa o resultado do teste, e cada nó folha representa um label de classe (decisão tomada depois de computar todos os atributos).
- Os caminhos da raiz à folha representam as regras de classificação.

Decision Trees

- Na análise de decisão, uma árvore de decisão e o diagrama de influência intimamente relacionado são usados como uma ferramenta de suporte de decisão visual e analítica, onde os valores esperados (ou utilidade esperada) de alternativas concorrentes são calculados.
- Uma árvore de decisão consiste em três tipos de nós:
    1. Nós de decisão
    2. Nós de chance
    3. Nós finais

Decision Trees



Decision Trees

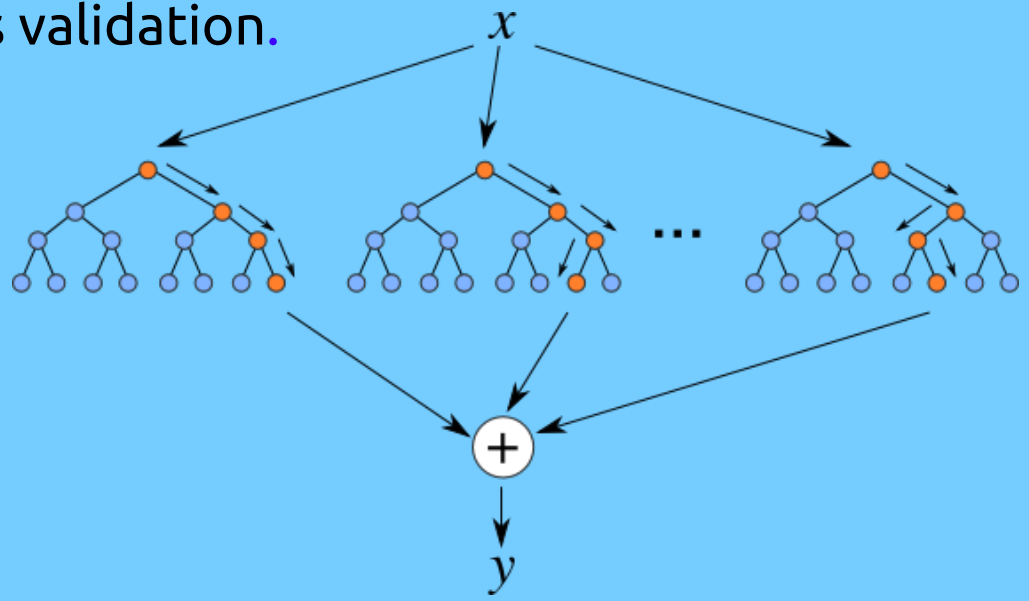
- Entre as ferramentas de suporte à decisão, as árvores de decisão (e diagramas de influência) têm várias vantagens:
  - São simples de entender e interpretar. As pessoas são capazes de entender os modelos de árvore de decisão após uma breve explicação.
  - Tem valor mesmo com poucos dados concretos. Insights importantes podem ser gerados com base em especialistas que descrevem uma situação (suas alternativas, probabilidades e custos) e suas preferências de resultados.
  - Ajudam a determinar os piores, melhores e esperados valores para diferentes cenários.
  - Podem ser combinadas com outras técnicas de decisão.

Random Forest

- Random forests ou random decision forests são um método de ensemble learning para classificação, regressão e outras tarefas que operam construindo uma infinidade de árvores de decisão no momento do treinamento e gerando a classe que é a moda das classes (classificação) ou previsão média (regressão) das árvores individuais.
- As random decision forests corrigem o hábito das árvores de decisão de se ajustar muito (overfitting) ao seu conjunto de treinamento. As random forests geralmente superam as árvores de decisão.
- O primeiro algoritmo de random decision forests foi criado em 1995 por Tin Kam Ho usando o método do subespaço aleatório, que, na formulação de Ho, é uma forma de implementar a abordagem de "discriminação estocástica" para classificação proposta por Eugene Kleinberg.

Random Forest

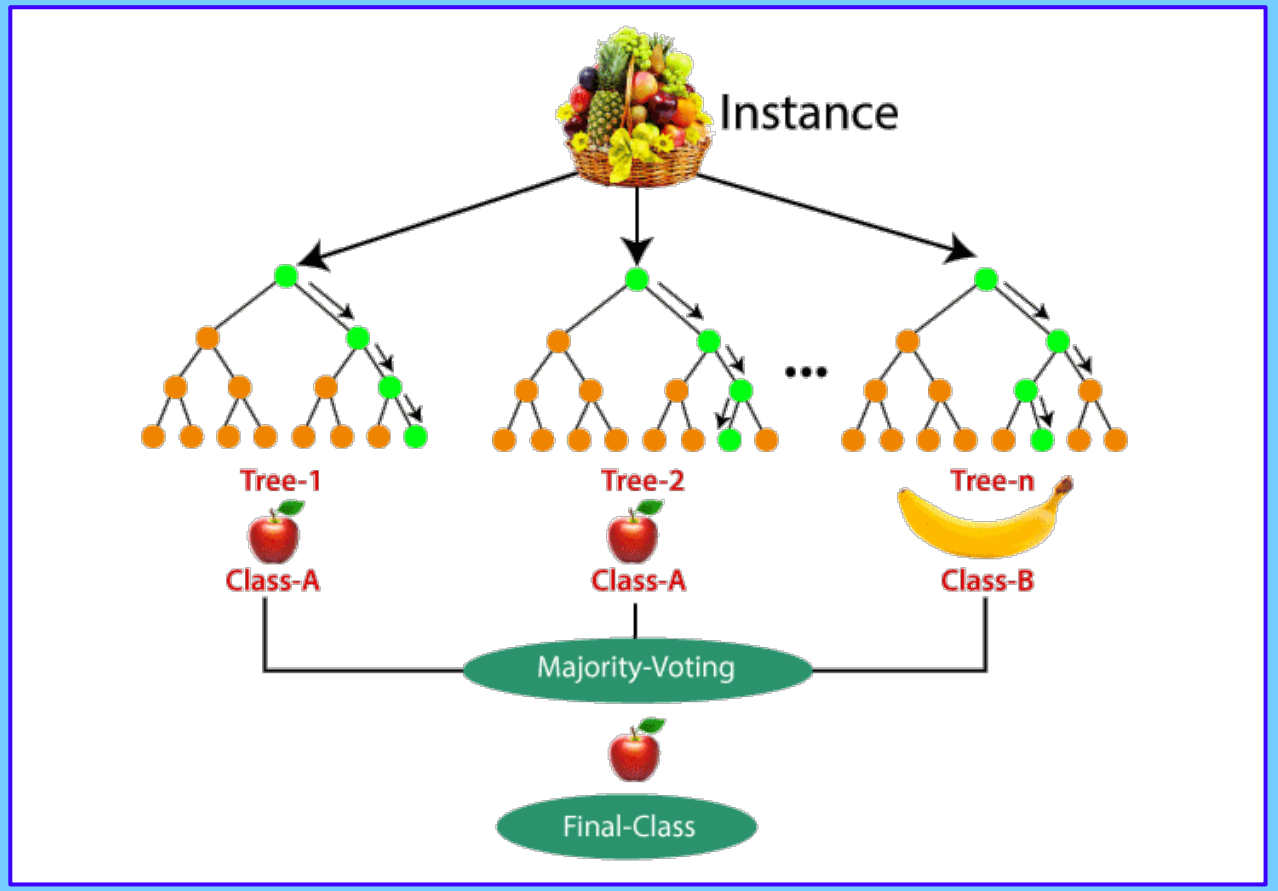
- As florestas são como reunir esforços de algoritmos de árvore de decisão.
- Fazer o trabalho em equipe de muitas árvores, melhorando assim o desempenho de uma única árvore aleatória.
- Embora não sejam muito semelhantes, as florestas fornecem os efeitos de uma K-fold cross validation.



Random Forest

- Vejamos um exemplo de um conjunto de dados de treinamento que consiste em várias frutas, como banana, maçã, abacaxi e manga.
- O classificador random forest divide este conjunto de dados em subconjuntos.
- Esses subconjuntos são oferecidos a cada árvore de decisão no sistema random forest. Cada árvore de decisão produz sua saída específica. Por exemplo, a previsão para as árvores 1 e 2 é maçã.
- Outra árvore de decisão^{n} previu a banana como resultado. O classificador random forest coleta a maioria dos votos para fornecer a previsão final. A maioria das árvores de decisão escolheu a maçã como sua previsão. Isso faz com que o classificador escolha maçã como a previsão final.

Random Forest



Regressão Logística

- O modelo logístico é usado para modelar a probabilidade de uma determinada classe ou evento existir, como aprovação/reprovação, vitória/derrota, vivo/morto ou saudável/doente.
- Isso pode ser estendido para modelar várias classes de eventos, como determinar se uma imagem contém um gato, cachorro, leão, etc. Cada objeto sendo detectado na imagem seria atribuído a uma probabilidade entre 0 e 1, com a soma de um.
- A regressão logística é um modelo estatístico que em sua forma básica usa uma função logística para modelar uma variável dependente binária, embora existam muitas extensões mais complexas.
- Matematicamente, um modelo logístico binário possui uma variável dependente com dois valores possíveis, como aprovação / reprovação que é representada por uma variável indicadora, onde os dois valores são rotulados como "0" e "1".

Regressão Logística

- Vejamos a probabilidade de passar em um exame versus horas de estudo. Para responder à seguinte pergunta:

  Um grupo de 20 alunos gasta entre 0 e 6 horas estudando para um exame. Como o número de horas gastas estudando afeta a probabilidade de o aluno ser aprovado no exame?

- A razão para usar a regressão logística para este problema é que os valores da variável dependente, passar e falhar, enquanto representados por "1" e "0", não são números cardinais.
- Se o problema foi alterado de forma que aprovação / reprovação foi substituído pela nota 0–100 (números cardinais), então a análise de regressão simples pode ser usada.

Regressão Logística

- A tabela mostra o número de horas que cada aluno passou estudando e se eles foram aprovados (1) ou reprovados (0).

Hours	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

- A análise de regressão logística fornece a seguinte saída.

	Coefficient	Std.Error	z-value	P-value (Wald)
Intercept	−4.0777	1.7610	−2.316	0.0206
Hours	1.5046	0.6287	2.393	0.0167

- O resultado indica que as horas de estudo estão significativamente associadas à probabilidade de passar no exame (p = 0.0167, teste de Wald). A saída também fornece os coeficientes para Intercept = -4.0777 e Hours = 1.5046.

Regressão Logística

- Esses coeficientes são inseridos na equação de regressão logística para estimar as chances (probabilidade) de passar no exame:

$$\begin{aligned}\text{Log-odds of passing exam} &= 1.5046 \cdot \text{Hours} - 4.0777 = 1.5046 \cdot (\text{Hours} - 2.71)\\ \text{Odds of passing exam} &= \exp(1.5046 \cdot \text{Hours} - 4.0777) = \exp(1.5046 \cdot (\text{Hours} - 2.71))\\ \text{Probability of passing exam} &= \frac{1}{1 + \exp(-(1.5046 \cdot \text{Hours} - 4.0777))}\end{aligned}$$

- Estima-se que uma hora adicional de estudo aumente as probabilidades logísticas de passar em 1.5046, portanto, multiplicar as probabilidades de passar por exp(1.5046) ≈ 4.5. O formulário com a interceptação-x (2.71) mostra que isso estima as probabilidades pares (log-odds 0, odds 1, probabilidade 1/2) para um aluno que estuda 2.71 horas.

Regressão Logística

- Por exemplo, para um aluno que estuda 2 horas, inserir o valor Hours = 2 na equação dá a probabilidade estimada de passar no exame de 0.26:

$$\text{Probability of passing exam} = \frac{1}{1 + \exp(-(1.5046 \cdot 2 - 4.0777))} = 0.26$$

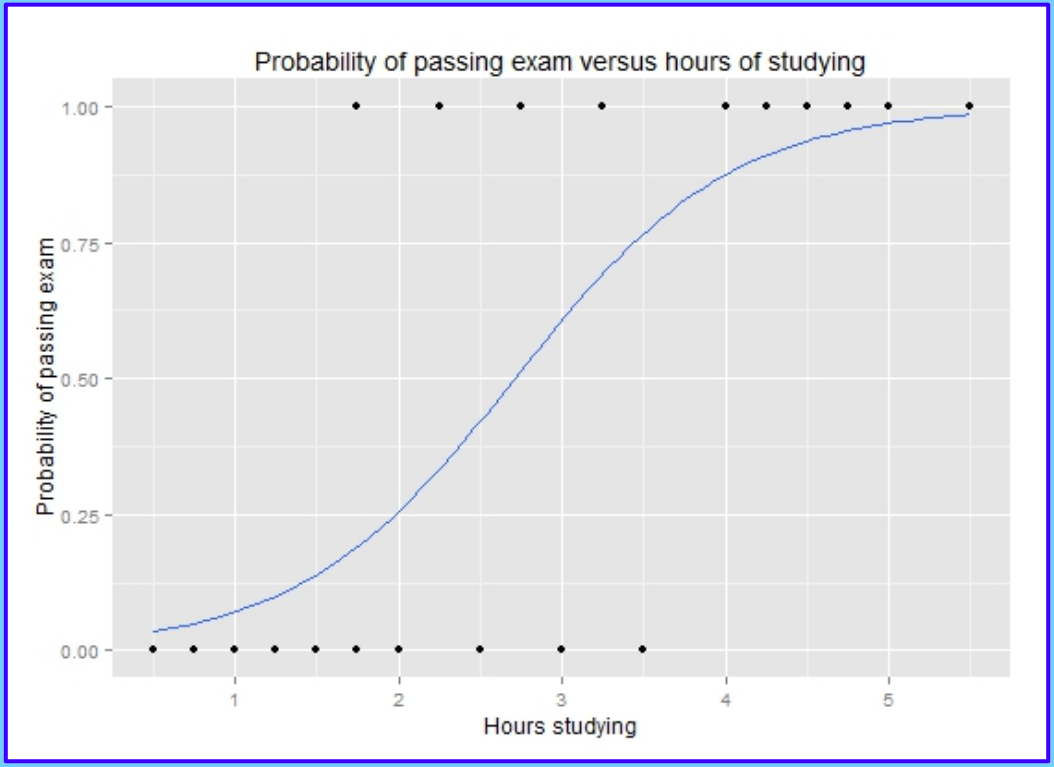
- Da mesma forma, para um aluno que estuda 4 horas, a probabilidade estimada de aprovação no exame é de 0.87:

$$\text{Probability of passing exam} = \frac{1}{1 + \exp(-(1.5046 \cdot 4 - 4.0777))} = 0.87$$

- A saída da análise de regressão logística fornece um valor p de p = 0.0167, que é baseado no Wald z-score. Em vez do método de Wald, o método recomendado para calcular o valor p para a regressão logística é o likelihood-ratio test (LRT), que para esses dados dá p = 0.0006.

Regressão Logística

- O gráfico mostra a probabilidade de aprovação no exame versus o número de horas de estudo, com a curva de regressão logística ajustada aos dados.



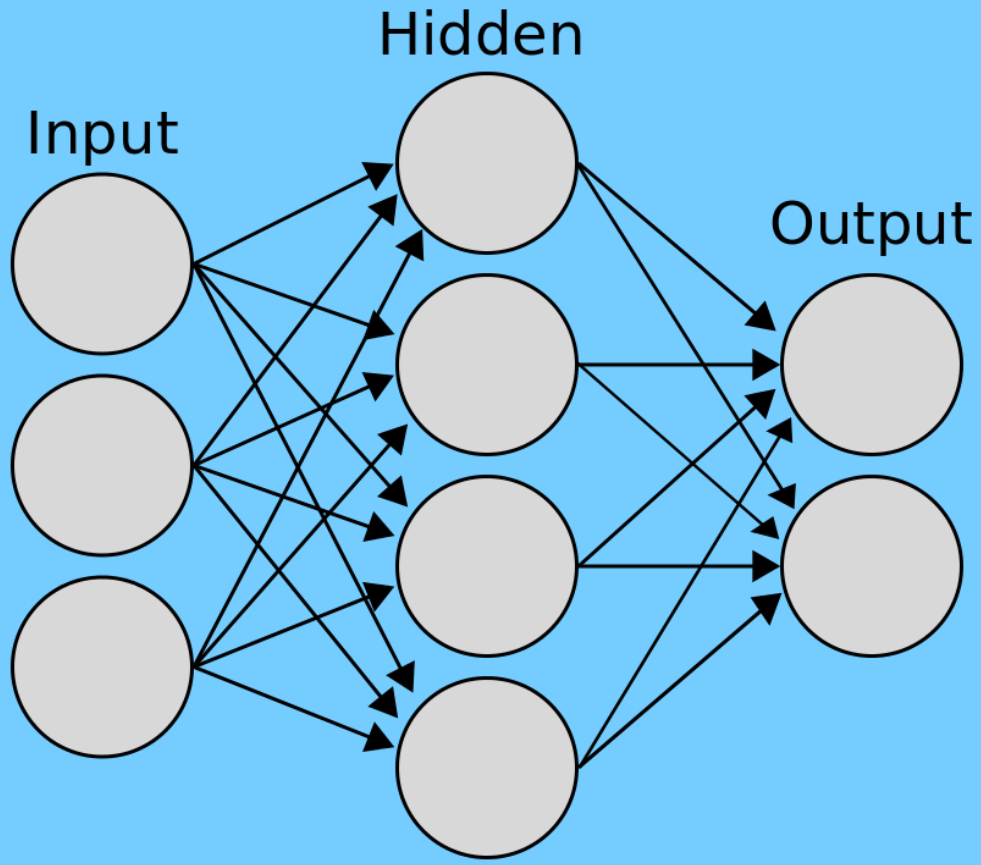
Redes Neurais Artificiais

- Redes neurais artificiais (RNAs), ou sistemas conexionistas, são sistemas de computação vagamente inspirados nas redes neurais biológicas que constituem os cérebros dos animais.
- Esses sistemas "aprendem" a realizar tarefas considerando exemplos, geralmente sem serem programados com nenhuma regra específica de tarefa.
- Uma RNA é um modelo baseado em uma coleção de unidades conectadas ou nós chamados "neurônios artificiais", que modelam vagamente os neurônios em um cérebro biológico.
- Cada conexão, como as sinapses em um cérebro biológico, pode transmitir informações, um "sinal", de um neurônio artificial para outro.
- Um neurônio artificial que recebe um sinal, pode processá-lo e, em seguida, sinalizar neurônios artificiais adicionais conectados a ele.

Redes Neurais Artificiais

Uma rede neural artificial é um grupo interconectado de nós, semelhante à vasta rede de neurônios em um cérebro.

Nela, cada nó circular representa um neurônio artificial e uma seta representa uma conexão da saída de um neurônio artificial para a entrada de outro.



Redes Neurais Artificiais

- Em implementações comuns de RNA, o sinal em uma conexão entre neurônios artificiais é um número real, e a saída (output) de cada neurônio artificial é calculada por alguma função não linear da soma de suas entradas (inputs).
- As conexões entre neurônios artificiais são chamadas de "edges".
- Neurônios artificiais e edges normalmente têm um peso (weight) que se ajusta à medida que o aprendizado avança. O peso aumenta ou diminui a força do sinal em uma conexão.
- Os neurônios artificiais podem ter um limite (threshold), tal que o sinal só é enviado se o sinal agregado cruzar esse limite.
- Normalmente, os neurônios artificiais são agregados em camadas (layers).

Redes Neurais Artificiais

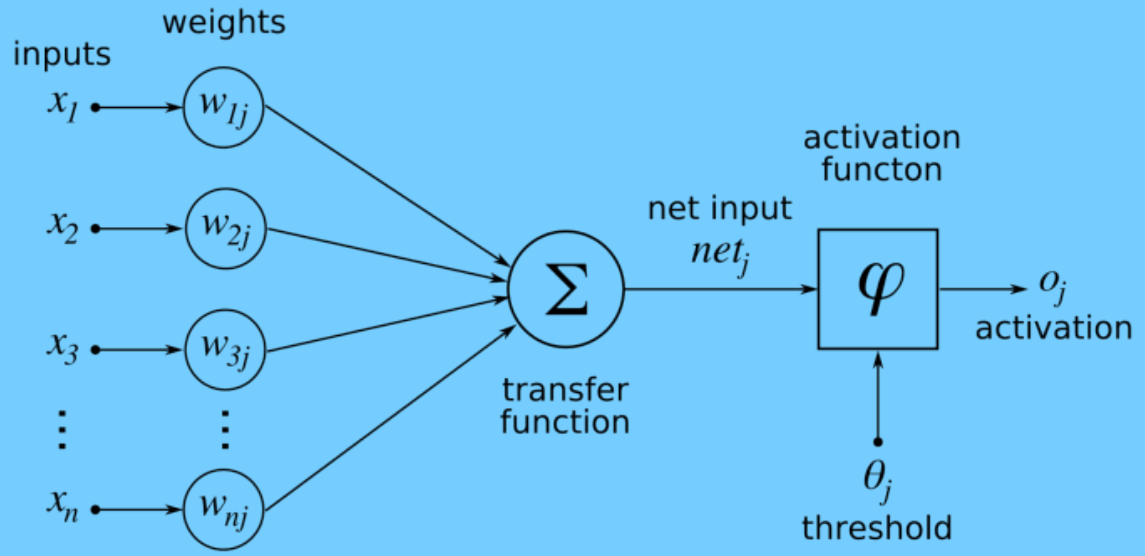
- Diferentes camadas podem realizar diferentes tipos de transformações em suas entradas.
- Os sinais viajam da primeira camada (a camada de entrada) para a última camada (a camada de saída), possivelmente depois de atravessar as camadas várias vezes.
- O objetivo original da abordagem da RNA era resolver problemas da mesma forma que um cérebro humano faria.
- No entanto, com o tempo, a atenção mudou para a execução de tarefas específicas, levando a desvios da biologia.

Redes Neurais Artificiais

- As redes neurais artificiais têm sido usadas em uma variedade de tarefas, incluindo visão computacional, reconhecimento de fala, tradução automática, filtragem de rede social, jogos de tabuleiro e videogames e diagnóstico médico.
- Deep Learning consiste em várias camadas ocultas em uma rede neural artificial. Essa abordagem tenta modelar a maneira como o cérebro humano processa luz e som em visão e audição. Algumas aplicações bem-sucedidas de deep learning são visão computacional e reconhecimento de fala.

Redes Neurais Artificiais

- O principal componente das RNAs são os neurônios artificiais.
- Cada neurônio recebe entradas de vários outros neurônios, multiplica-os por pesos atribuídos, adiciona-os e passa a soma para um ou mais neurônios. Alguns neurônios artificiais podem aplicar uma função de ativação à saída antes de passá-la para a próxima variável.



Algoritmos de Machine Learning

- Existem muitos algoritmos de Machine Learning, por este motivo, é inviável demonstrar todos eles nesta apresentação.
- A seguir temos listas com diversos algoritmos importantes que podem nos ajudar a resolver diversos problemas:

https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms

https://en.wikipedia.org/wiki/Outline_of_machine_learning

Métricas de Machine Learning

- Escolher a métrica certa é crucial ao avaliar os modelos de Machine Learning.
- Várias métricas são propostas para avaliar modelos de ML em diferentes aplicações.
- Uma métrica de avaliação quantifica o desempenho de um modelo preditivo.
- Isso normalmente envolve o treinamento de um modelo em um conjunto de dados, usando o modelo para fazer previsões em um conjunto de dados de validação não usado durante o treinamento e, em seguida, comparando as previsões aos valores esperados no conjunto de dados de validação.
- Para problemas de classificação, as métricas envolvem comparar o label de classe esperado com o label de classe previsto ou interpretar as probabilidades previstas para os labels de classe do problema.

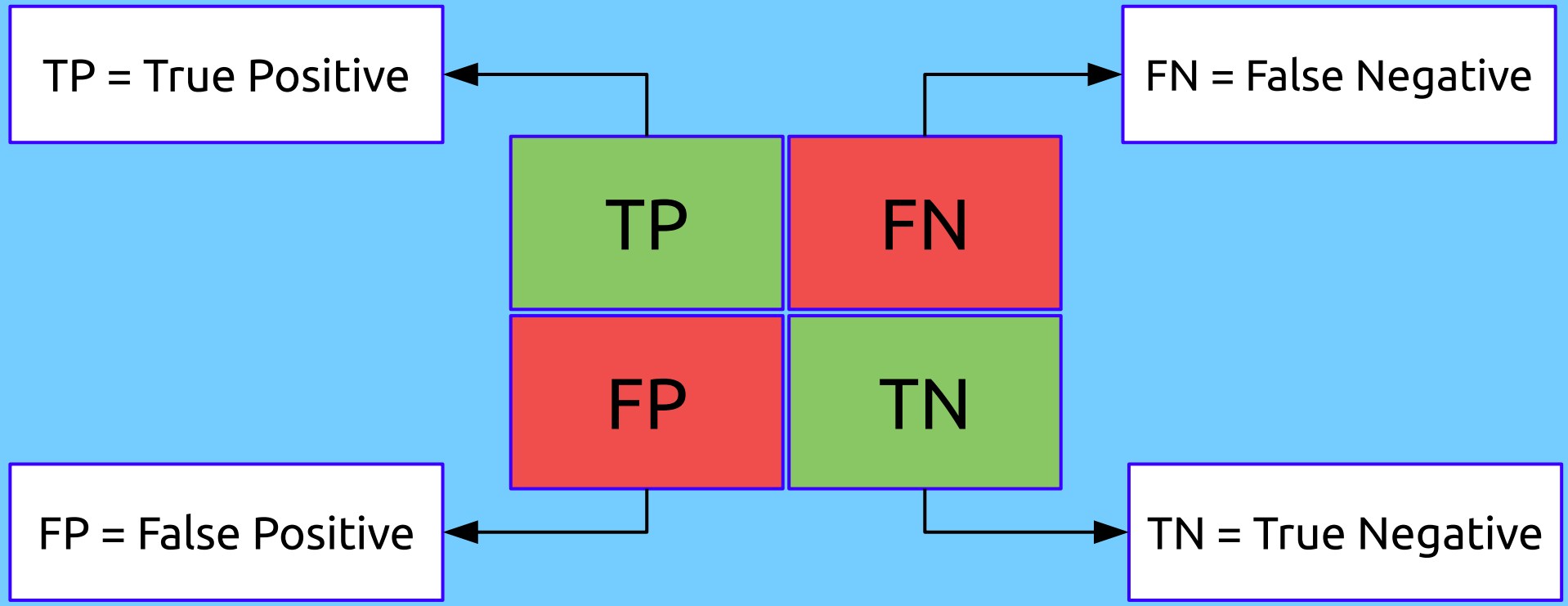
Métricas de Machine Learning

- Como uma nota, também vale a pena mencionar que a métrica é diferente da função de perda (loss function).
- Funções de perda são funções que mostram uma medida do desempenho do modelo e são usadas para treinar um modelo de Machine Learning (usando algum tipo de otimização) e geralmente são diferenciáveis nos parâmetros do modelo.
- Por outro lado, as métricas são usadas para monitorar e medir o desempenho de um modelo (durante o treinamento e teste) e não precisam ser diferenciáveis. No entanto, se para algumas tarefas a métrica de desempenho é diferenciável, ela pode ser usada como uma função de perda (talvez com algumas regularizações adicionadas a ela) e uma métrica, como MSE.

Confusion Matrix

- Um dos principais conceitos no desempenho da classificação é a matriz de confusão (também conhecida como matriz de erro).
- Ela é uma visualização tabular das previsões do modelo versus os labels de verdade.
- Cada linha da matriz de confusão representa as instâncias em uma classe prevista e cada coluna representa as instâncias em uma classe real.
- O nome vem do fato de que torna mais fácil ver se o sistema está confundindo duas classes (ou seja, comumente rotulando uma como a outra).

Confusion Matrix



Confusion Matrix

- Dada uma amostra de 12 fotos, 8 de gatos e 4 de cachorros, onde os gatos pertencem à classe 1 e os cães pertencem à classe 0:

Verdadeiras = 1 1 1 1 1 1 1 1 0 0 0 0

- Suponha que um classificador que distingue entre cães e gatos seja treinado, e nós tiramos as 12 fotos e as examinamos através do classificador, e o classificador faz 9 previsões precisas e erra 3: 2 gatos erroneamente previstos como cães (2 primeiras previsões) e 1 cão erroneamente previsto como um gato (última previsão).

Previsões = 0 0 1 1 1 1 1 1 0 0 0 1

Confusion Matrix

- Com esses dois conjuntos rotulados (verdadeiras e previsões), podemos criar uma matriz de confusão que resumirá os resultados do teste do classificador:

		Classe Prevista	
		Gatos	Cães
Classe Verdadeira	Gatos	6	2
	Cães	1	3

Confusion Matrix

- Nesta matriz de confusão, das 8 imagens de gatos, o sistema julgou que 2 eram cães e, das 4 imagens de cães, previu que 1 era de gatos.
- Todas as previsões corretas estão localizadas na diagonal da tabela (destacada em verde), portanto, é fácil inspecionar visualmente a tabela para erros de previsão, pois eles serão representados por valores fora da diagonal (destacados em vermelho).

Accuracy

- A accuracy da classificação é talvez a métrica mais simples que se possa imaginar e é definida como o número de previsões corretas dividido pelo número total de previsões, multiplicado por 100. Portanto, no exemplo acima, de 12 amostras, 9 são previstas corretamente, resultando em uma accuracy de classificação de:

Accuracy = Previsões Corretas / Previsões Totais

$$(6 + 3) / (12) * 100 = 75\%$$

Precision

- Existem muitos casos em que a accuracy da classificação não é um bom indicador do desempenho do seu modelo. Um desses cenários é quando sua distribuição de classe é desequilibrada (uma classe é mais frequente do que outras).
- Nesse caso, mesmo se você prever todas as amostras como a classe mais frequente, você obterá uma alta taxa de accuracy, o que não faz sentido (porque seu modelo não está aprendendo nada e está apenas prevendo tudo como a classe superior).
- Portanto, precisamos examinar as métricas de desempenho específicas da classe também. A precision é uma dessas métricas, que é definida como:

$$\text{Precision} = TP / (TP + FP)$$

Precision

- A precision da classe Gato e Cachorro no exemplo anterior pode ser calculada como:

$$Precision\_Gato = (6 / (6 + 1)) * 100 = 85.7\%$$

$$Precision\_Cão = (3 / (3 + 2)) * 100 = 60\%$$

- Como podemos ver, o modelo tem uma precisão muito maior na previsão de amostras de gatos, em comparação com cachorros.
- Isso não é surpreendente, já que a modelo viu mais exemplos de imagens de gatos durante o treinamento, tornando-o melhor na classificação dessa classe.

Recall

- Recall é outra métrica importante, que é definida como a fração de amostras de uma classe que são previstas corretamente pelo modelo. Mais formalmente:

$$\text{Recall} = TP / (TP + FN)$$

- Portanto, para nosso exemplo acima, a taxa de recall de classes de gatos e cães pode ser encontrada como:

$$\text{Recall\_Gato} = (6 / (6 + 2)) * 100 = 75\%$$

$$\text{Recall\_Cão} = (3 / (3 + 1)) * 100 = 75\%$$

F1 Score

- Dependendo da aplicação, você pode desejar dar maior prioridade ao recall ou precision. Mas existem muitas aplicações nas quais a recall e a precision são importantes.
- Portanto, é natural pensar em uma maneira de combinar esses dois em uma única métrica. Uma métrica popular que combina precision e recall é chamada de F1 score, que é a média harmônica de precision e recall definida como:

$$F1\_Score = 2 * Precision * Recall / (Precision + Recall)$$

- Portanto, para o nosso exemplo de classificação com a matriz de confusão apresentada antes, o F1 Score pode ser calculada como:

$$F1\_gato = 2 * 0.857 * 0.75 / (0.857 + 0.75) = 79.99\%$$

Métricas para Classificação Desequilibrada

- As métricas de classificação estão mais preocupadas com a avaliação de classificadores com base em sua eficácia na separação de classes.

  "Métricas baseadas em quão bem o modelo classifica os exemplos [...] Estas são importantes para muitas aplicações [...] onde classificadores são usados para selecionar as melhores n instâncias de um conjunto de dados ou quando uma boa separação de classes é crucial."

  — An Experimental Comparison Of Performance Measures For Classification, 2008.

- Essas métricas requerem que um classificador preveja uma pontuação ou probabilidade de associação à classe.

Métricas para Classificação Desequilibrada

- A partir dessa pontuação, diferentes limites podem ser aplicados para testar a eficácia dos classificadores.
- Os modelos que mantêm uma boa pontuação em uma faixa de limites terão uma boa separação de classes e serão classificados em posições mais altas.

  "…Considere um classificador que fornece uma pontuação numérica para uma instância a ser classificada na classe positiva. Portanto, em vez de uma simples previsão positiva ou negativa, a pontuação introduz um nível de granularidade"

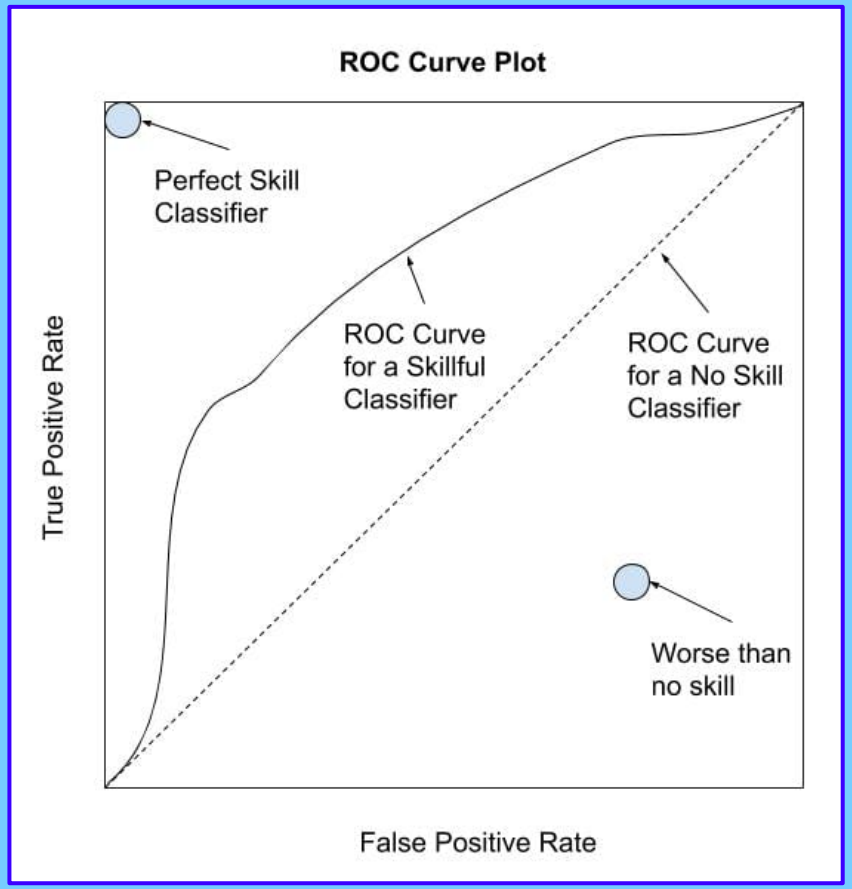
  — Page 53, Learning from Imbalanced Data Sets, 2018.
- A métrica de classificação mais comumente usada é a **Curva ROC** ou **Análise ROC**.

ROC

- ROC é um acrônimo que significa Receiver Operating Characteristic e resume um campo de estudo para analisar classificadores binários com base em sua capacidade de discriminar classes.
- Uma curva ROC é um gráfico de diagnóstico para resumir o comportamento de um modelo, calculando a taxa de falso positivo (false positive rate) e a taxa de verdadeiro positivo (true positive rate) para um conjunto de previsões do modelo sob diferentes limites.
- A true positive rate é o recall ou sensibilidade:
  - **TruePositiveRate** = TruePositive / (TruePositive + FalseNegative)
- A false positive rate é calculada como:
  - **FalsePositiveRate** = FalsePositive / (FalsePositive + TrueNegative)
- Cada limite é um ponto no gráfico e os pontos são conectados para formar uma curva. Um classificador que não tem nenhuma habilidade (por exemplo, prevê a classe majoritária em todos os limites) será representado por uma linha diagonal da parte inferior esquerda para a superior direita.

Gráfico da Curva ROC

- Quaisquer pontos abaixo desta linha têm pior do que nenhuma habilidade. Um modelo perfeito será um ponto no canto superior esquerdo do gráfico.

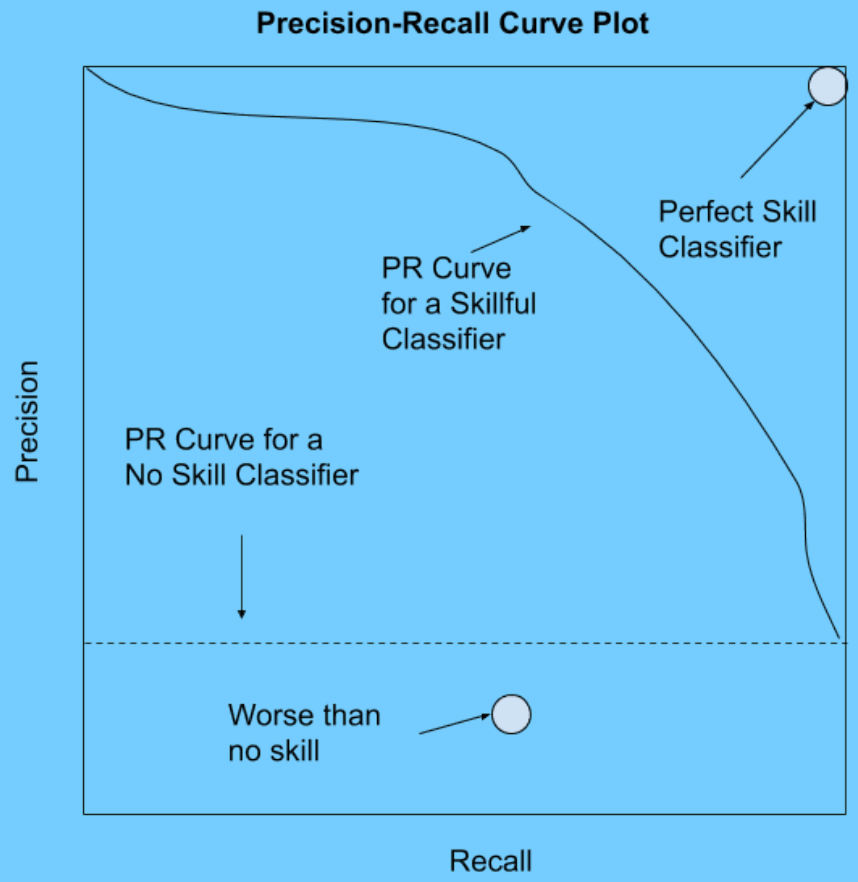


Curva ROC

- A curva ROC é um diagnóstico útil para um modelo.
- A área sob a curva ROC pode ser calculada e fornece uma pontuação única para resumir o gráfico que pode ser usado para comparar modelos.
- Um classificador sem habilidade terá uma pontuação de 0.5, enquanto um classificador perfeito terá uma pontuação de 1.0.
  - ROC AUC = ROC Area Under Curve
- Embora geralmente eficazes, a Curva ROC e a AUC ROC podem ser otimistas sob um desequilíbrio de classe severo, especialmente quando o número de exemplos na classe minoritária é pequeno.
- Uma alternativa à Curva ROC é a curva de precision-recall que pode ser usada de maneira semelhante, embora se concentre no desempenho do classificador na classe minoritária.

Gráfico da Curva Precision-Recall

- Novamente, diferentes limites são usados em um conjunto de previsões por um modelo e, neste caso, a precision e o recall são calculadas. Os pontos formam uma curva e os classificadores com melhor desempenho em uma faixa de limites diferentes serão classificados mais alto.
- Um classificador sem habilidade será uma linha horizontal no gráfico com uma precision que é proporcional ao número de exemplos positivos no conjunto de dados. Para um conjunto de dados balanceado, isso será 0.5. Um classificador perfeito é representado por um ponto no canto superior direito.



Curva Precision-Recall

- Como a Curva ROC, a Curva Precision-Recall é uma ferramenta de diagnóstico útil para avaliar um único classificador, mas desafiadora para comparar classificadores.
- E como o ROC AUC, podemos calcular a área sob a curva como uma pontuação e usar essa pontuação para comparar os classificadores.
- Nesse caso, o foco na classe minoritária torna o AUC de Precision-Recall mais útil para problemas de classificação desequilibrada.
  - PR AUC = Precision-Recall Area Under Curve
- Para mais detalhes você pode visitar o seguinte tutorial:

  https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/

Métricas Relacionadas à Regressão

- Os modelos de regressão são outra família de modelos estatísticos e de Machine Learning, que são usados para prever valores-alvo contínuos.
- Eles têm uma ampla gama de aplicações, desde previsão de preços de casas, sistemas de preços de comércio eletrônico, previsão do tempo, previsão do mercado de ações até super resolução de imagem, aprendizado de features por meio de codificadores automáticos e compactação de imagem.
- Modelos como regressão linear, random forest, XGboost, convolutional neural network, recurrent neural network são alguns dos modelos de regressão mais populares.
- As métricas usadas para avaliar esses modelos devem ser capazes de funcionar em um conjunto de valores contínuos (com cardinalidade infinita) e, portanto, são ligeiramente diferentes das métricas de classificação.

Mean Squared Error (MSE)

- Mean Squared Error é talvez a métrica mais popular usada para problemas de regressão. Basicamente, ele encontra o erro quadrático médio entre os valores previstos e reais.
- Vamos supor que temos um modelo de regressão que prevê o preço das casas em São Paulo (mostre-as com $\hat{y}_i$) e, para cada casa, também temos o preço real pelo qual a casa foi vendida (denotado com y_i). Então, o MSE pode ser calculado como:

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Mean Absolute Error (MAE)

- O Mean Absolute Error (ou desvio absoluto médio) é outra métrica que encontra a distância absoluta média entre os valores previstos e de destino. MAE é definido como:

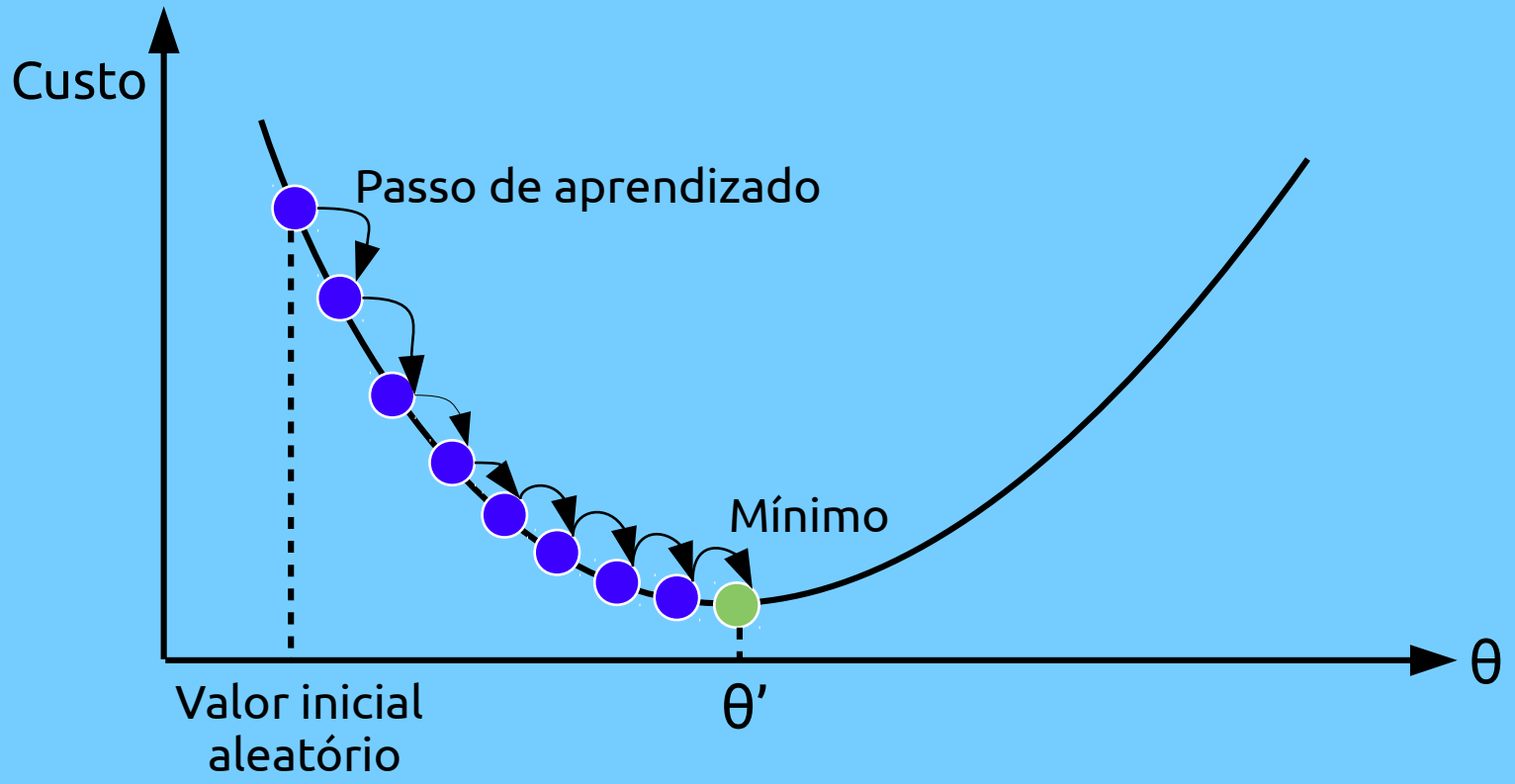
$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- O MAE é conhecido por ser mais robusto para os outliers do que o MSE. A principal razão é que no MSE, ao elevar os erros ao quadrado, os outliers (que geralmente apresentam erros maiores do que outras amostras) recebem mais atenção e dominância no erro final e impactam os parâmetros do modelo.

Gradient Descent

- Gradient Descent é um algoritmo de otimização genérico capaz de encontrar soluções otimizadas para uma ampla gama de problemas. A ideia geral do Gradient Descent é ajustar os parâmetros de forma iterativa para minimizar uma função de custo.
- Suponha que você esteja perdido nas montanhas em uma névoa densa e só possa sentir a inclinação do solo abaixo de seus pés. Uma boa estratégia para chegar rapidamente ao fundo do vale é descer na direção da encosta mais íngreme. Isso é exatamente o que Gradient Descent faz: mede o gradiente local da função de erro em relação ao vetor de parâmetros θ, e vai na direção do gradiente descendente. Assim que o gradiente for zero, você atingiu o mínimo!
- Concretamente, você começa preenchendo θ com valores aleatórios (isso é chamado de inicialização aleatória). Em seguida, você o melhora gradualmente, dando um pequeno passo de cada vez, cada passo tentando diminuir a função de custo (por exemplo, o MSE), até que o algoritmo convirja para um mínimo.

Gradient Descent

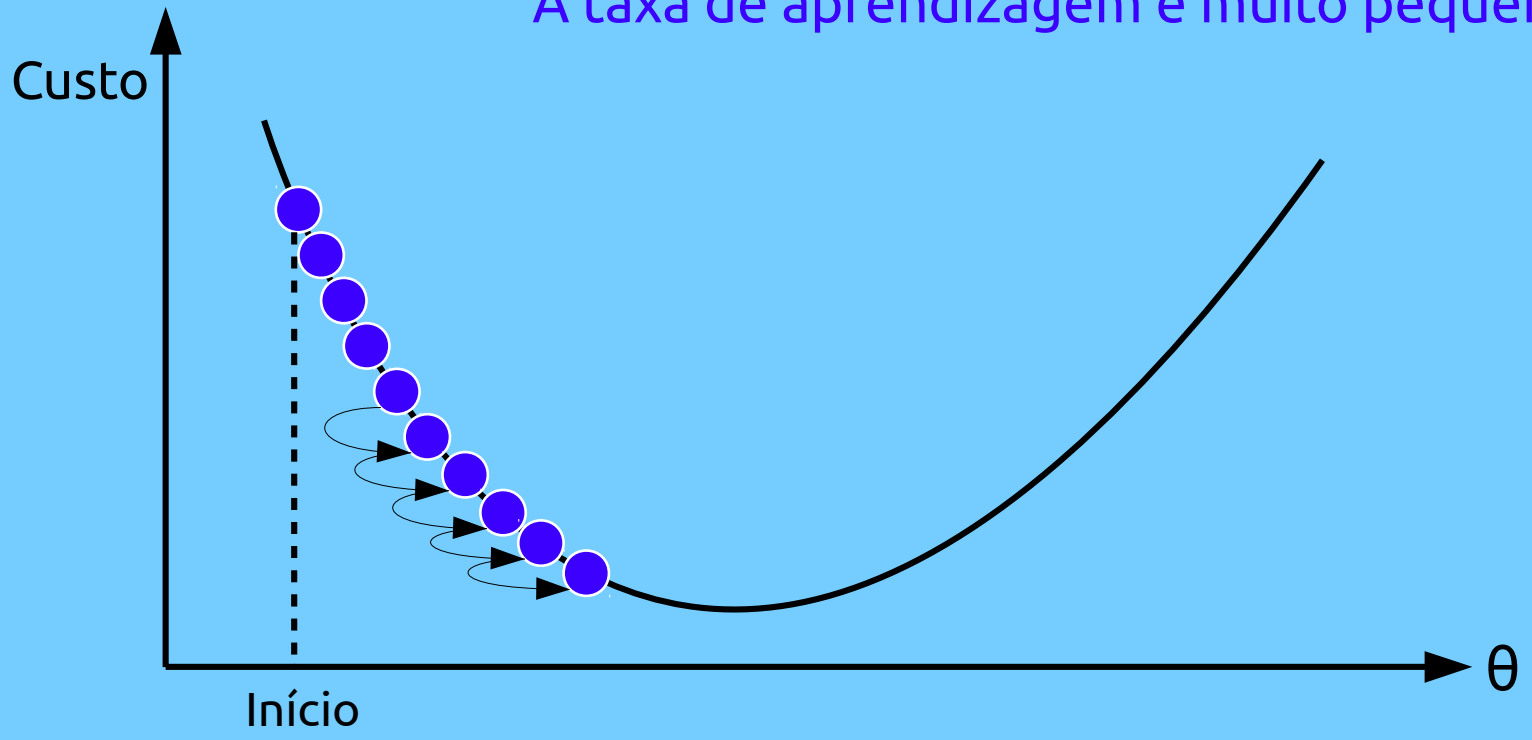


Gradient Descent

- **Figura anterior**: Nesta representação de Gradient Descent, os parâmetros do modelo são inicializados aleatoriamente e são ajustados repetidamente para minimizar a função de custo; o tamanho do passo de aprendizagem é proporcional à inclinação da função de custo, então os passos ficam gradualmente menores conforme os parâmetros se aproximam do mínimo.
- Um parâmetro importante no Gradient Descent é o tamanho das etapas, determinado pelo hiperparâmetro da taxa de aprendizagem.
- Se a taxa de aprendizado for muito pequena, o algoritmo terá que passar por muitas iterações para convergir, o que levará muito tempo.

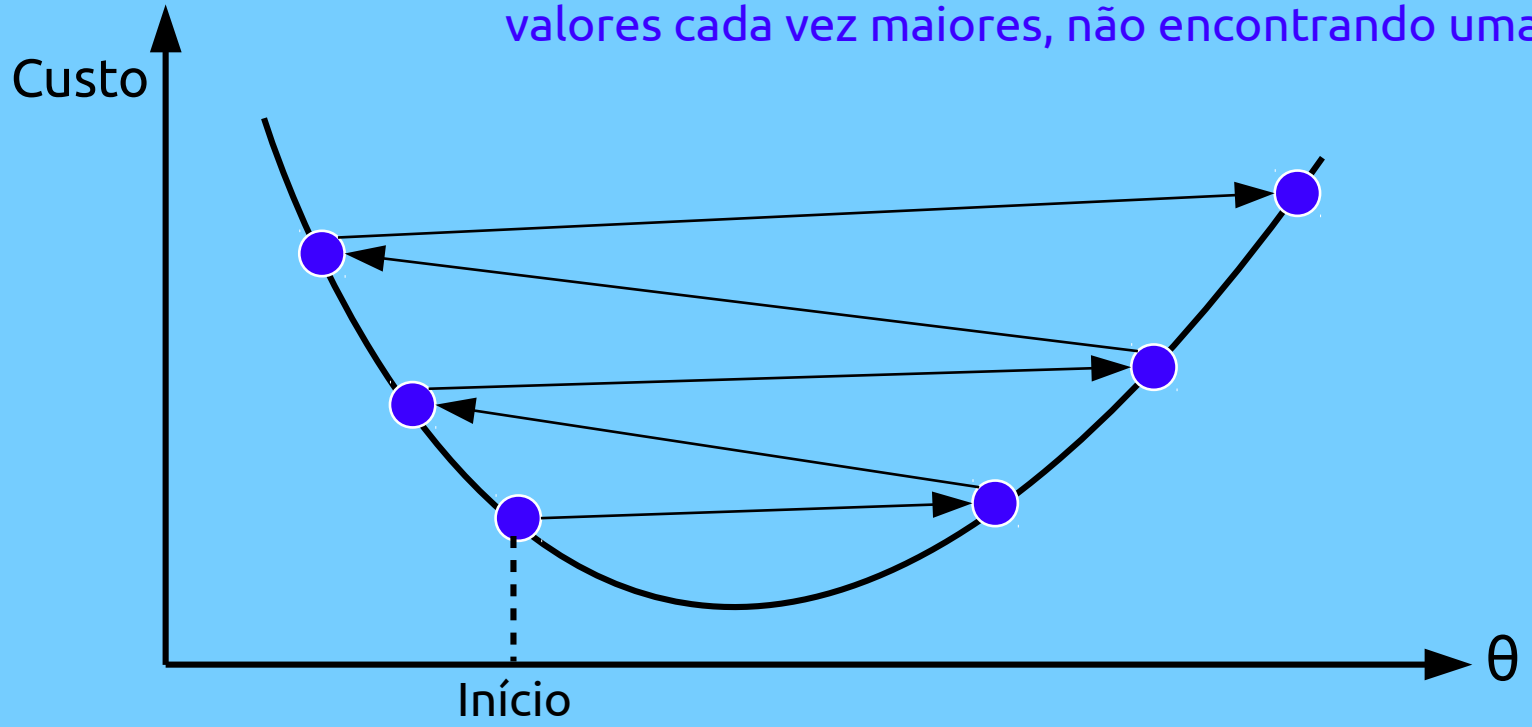
Gradient Descent

A taxa de aprendizagem é muito pequena



Gradient Descent

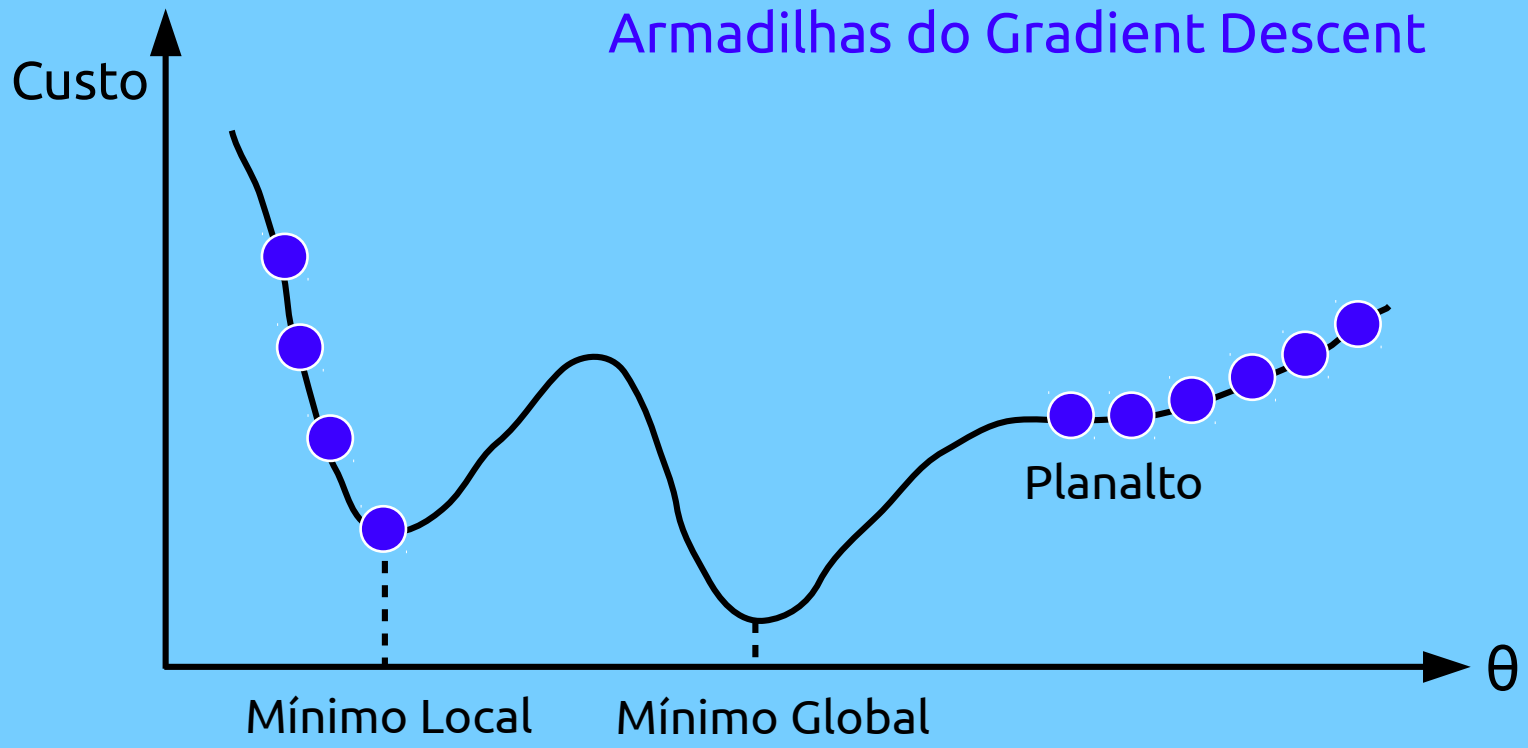
Por outro lado, se a taxa de aprendizado for muito alta, você pode pular o vale e acabar do outro lado, possivelmente ainda mais alto do que estava antes. Isso pode fazer o algoritmo divergir, com valores cada vez maiores, não encontrando uma boa solução



Gradient Descent

- Por fim, nem todas as funções de custo parecem bacias regulares e legais.
- Pode haver buracos, cristas, planaltos e todo tipo de terreno irregular, dificultando a convergência ao mínimo. A Figura a seguir mostra os dois principais desafios do Gradient Descent.
- Se a inicialização aleatória iniciar o algoritmo à esquerda, ele convergirá para um mínimo local, que não é tão bom quanto o mínimo global.
- Se começar da direita, demorará muito para cruzar o planalto. E se você parar muito cedo, nunca alcançará o mínimo global.

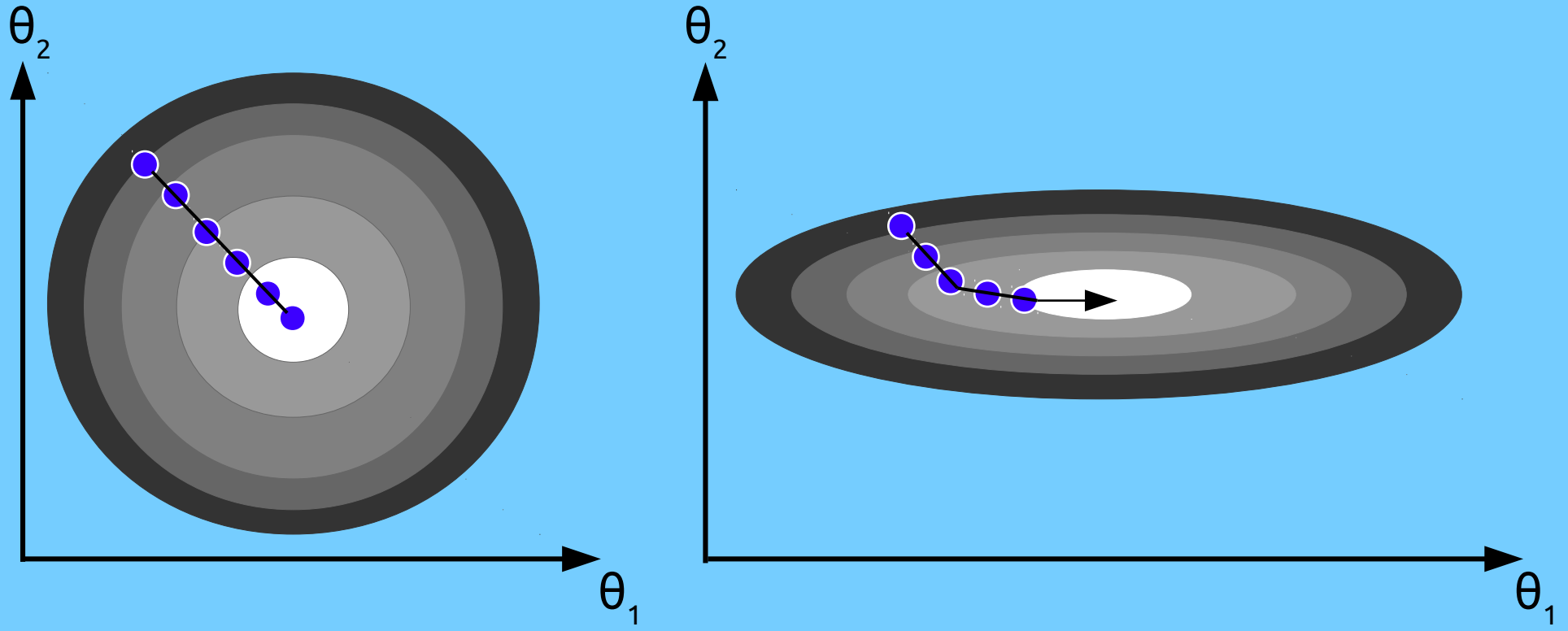
Gradient Descent



Gradient Descent

- Felizmente, a função de custo MSE para um modelo de regressão linear passa a ser uma função convexa, o que significa que se você escolher dois pontos quaisquer na curva, o segmento de linha que os une nunca cruzará a curva.
- Isso implica que não há mínimos locais, apenas um mínimo global.
- É também uma função contínua com uma inclinação que nunca muda abruptamente.
- Esses dois fatos têm uma grande consequência: o Gradient Descent tem a garantia de se aproximar arbitrariamente do mínimo global (se você esperar o tempo suficiente e se a taxa de aprendizagem não for muito alta).
- Na verdade, a função de custo tem o formato de uma tigela, mas pode ser uma tigela alongada se os recursos tiverem escalas muito diferentes. A Figura a seguir mostra o Gradient Descent em um conjunto de treinamento onde os recursos 1 e 2 têm a mesma escala (à esquerda) e em um conjunto de treinamento onde o recurso 1 tem valores muito menores do que o recurso 2 (à direita).

Gradient Descent



Gradient Descent com (esquerda) e sem (direita) feature scaling

Gradient Descent

- Como você pode ver, à esquerda o algoritmo Gradient Descent vai direto ao mínimo, alcançando-o rapidamente, enquanto à direita ele primeiro segue em uma direção quase ortogonal à direção do mínimo global e termina com uma longa marcha descendo um vale quase plano. Eventualmente atingirá o mínimo, mas levará muito tempo.
- Ao usar Gradient Descent, você deve garantir que todos os features têm uma escala semelhante (por exemplo, usando a classe StandardScaler do Scikit-Learn), ou então vai demorar muito mais para convergir.
- Este diagrama também ilustra o fato de que treinar um modelo significa procurar uma combinação de parâmetros do modelo que minimiza uma função de custo (sobre o conjunto de treinamento). É uma busca no espaço de parâmetros do modelo: quanto mais parâmetros um modelo tem, mais dimensões esse espaço tem e mais difícil é a busca: procurar uma agulha em um palheiro de 300 dimensões é muito mais complicado do que em 3 dimensões. Felizmente, como a função de custo é convexa no caso da regressão linear, a agulha está simplesmente no fundo da tigela.

Ferramentas de Machine Learning

- Com a ajuda de sistemas de Machine Learning, podemos examinar dados, aprender com eles e tomar decisões.
- O Machine Learning envolve algoritmos e uma biblioteca de ML é um pacote de algoritmos.
- Existem vários softwares de ML disponíveis no mercado.
- Listaremos a seguir os mais populares entre eles.

Scikit-Learn

Scikit-Learn traz as revolucionárias capacidades do Machine Learning para a linguagem Python.

- Possui ferramentas eficientes e simples para data mining e análise de dados, acessível para todos e reusável em vários contextos. Construída em NumPy, SciPy e matplotlib. Open source, usável comercialmente com licença-BSD. Dentre algumas tarefas que a biblioteca conta: classificação, regressão, clusterização, redução de dimensionalidade, seleção de modelos e pré-processamento.

Tensorflow

- Tensorflow é uma plataforma open source end-to-end para machine learning.
- Possui um ecossistema compreensivo e flexível de ferramentas, bibliotecas e recursos de comunidade que permite os pesquisadores empurrarem o state-of-art em machine learning e para desenvolvedores construirem aplicações com machine learning.

PyTorch

- PyTorch é uma plataforma open source de Deep Learning que fornece um caminho rápido de protótipos de pesquisa para implantação de produção. Características chaves e capacidades: Front-End Híbrido, Treinamento Distribuído, Python-Primeiro, ferramentas e bibliotecas.

H2O

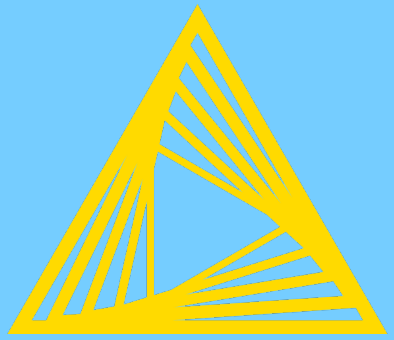
- A plataforma de Machine Learning open source para empresas. H2O é plataforma de machine learning completamente open source distribuída em memória com escalabilidade linear. H2O suporta os algoritmos mais utilizados em estatística e machine learning, incluindo gradient boosted machines, generalized linear models, deep learning e mais. H2O também tem uma funcionalidade de AutoML líder na indústria, que é capaz de rodar automaticamente através de todos os algoritmos e seus hiperparâmetros para produzir uma tabela de liderança dos melhores modelos. A plataforma H2O é usada por mais de 18.000 organizações globalmente e é extremamente popular em ambas as comunidades de R e Python.

Weka

- Waikato Environment for Knowledge Analysis (Weka), desenvolvido na Universidade de Waikato, Nova Zelândia, é um software livre licenciado sob a GNU General Public License, e o software acompanhante do livro "Data Mining: Practical Machine Learning Tools and Techniques".
- Weka contém uma coleção de ferramentas de visualização e algoritmos para análise de dados e modelagem preditiva, junto com interfaces gráficas de usuário para fácil acesso a essas funções.

KNIME

- KNIME (Konstanz Information Miner) é uma plataforma de integração, relatórios e análise de dados gratuita e de código aberto. O KNIME integra vários componentes para Machine Learning e mineração de dados por meio de seu conceito de pipelining de dados modular "Lego of Analytics".
- Uma interface gráfica de usuário e o uso de JDBC permitem a montagem de nós combinando diferentes fontes de dados, incluindo pré-processamento (ETL: Extraction, Transformation, Loading), para modelagem, análise de dados e visualização sem, ou apenas com um mínimo de programação.



Open for Innovation

KNIME

Considerações Finais

"Artificial intelligence is defined as the branch of science and technology that is concerned with the study of software and hardware to provide machines the ability to learn insights from data and the environment, and the ability to adapt in changing situations with high precision, accuracy and speed."

Amit Ray

Referências

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
- https://cloud.google.com/ai-platform/docs/ml-solutions-overview
- https://machinelearningmastery.com/basic-concepts-in-machine-learning
- https://en.wikipedia.org/wiki/Machine_learning
- https://www.sciencedirect.com/science/article/pii/S2405844018353404
- https://as595.github.io/classification/
- https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
- https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff