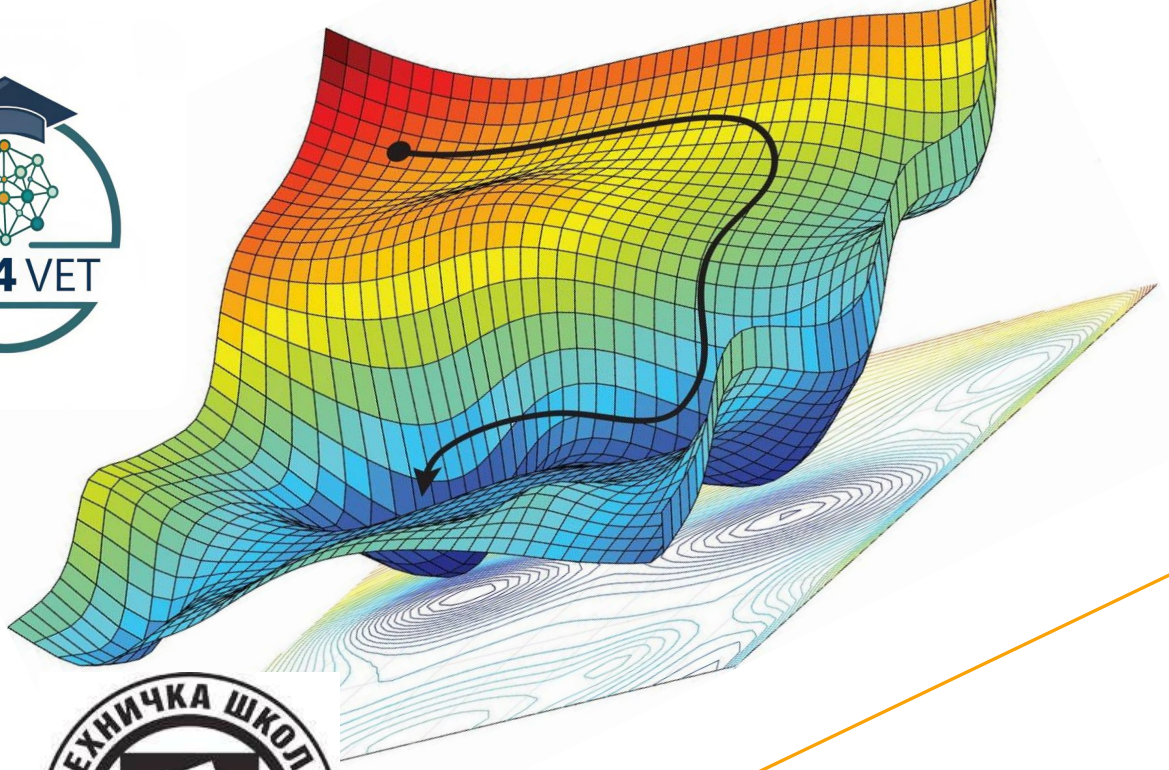


Co-funded by the European Union

FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL E MACHINE LEARNING

Technical school Pirot

KA220-VET - Parcerias de cooperação no ensino e formação profissional

Título do Projeto: AI tools for VET schools

Data do Documento: Jun 2025

Este material foi compilado e preparado para efeitos de projeto Erasmus por:
Technical school Pirot (autor: Boban Blagojević, coautores: Bojan Ćirić, Aleksandar Madić)
ICEP (autor: Ladislav Mariš, coautor: Adelaida Fanfarova)
Agrupamento de Escolas Tomás Cabreira (autor: Sandra Nobre, coautores: Rui Dias, Guilherme Mota, Carla Lima, Maria Torrinha)
Traduzido por: Carla Lima, Guilherme Mota, Rui Dias e Sandra Nobre

INFORMAÇÃO DE MORADA:

Takovska 22, Pirot, Serbia
Web: https://book.tsp.edu.rs

Conteúdos

1. INTELIGÊNCIA ARTIFICIAL

Bem-vindo ao tema da **Inteligência Artificial (IA) (eng. Artificial Intelligence - AI)**! Nesta secção, exploraremos o fascinante mundo da IA, que envolve a criação de sistemas que podem executar tarefas que normalmente exigem inteligência humana. Aprenderá sobre a história e o desenvolvimento da IA, os conceitos fundamentais e as considerações éticas em torno de seu uso. Também discutiremos os diferentes tipos de IA, incluindo estreita, geral e superinteligência, e como a IA está transformando vários aspetos de nossas vidas diárias.

1.1 O Conceito da Inteligência Artificial

A Inteligência Artificial (*IA*) é uma área que se dedica ao desenvolvimento de programas que, pelas suas capacidades, dão a impressão de um comportamento inteligente. Estes programas caraterizam-se pela capacidade de identificar relações complexas e tirar conclusões com base nessas relações. Veremos que estas atividades têm fundamentos em disciplinas como a matemática, a ciência da computação e a robótica. Devido à sua ampla aplicação, este campo também se relaciona com outras disciplinas que estudam a inteligência, como a neurociência, a filosofia e a arte, bem como com os seus efeitos na sociedade, como a sociologia, o direito e a ética.

É importante salientar que nem todos os programas que apresentam algum tipo de comportamento *inteligente* precisam de ser baseados em inteligência artificial. Vejamos um programa que abre uma porta quando se entra num edifício. Esta funcionalidade pode ser ativada por sensores de proximidade, que detetam a presença, ou pode requerer a introdução de um código que deve corresponder ao código esperado. Ambos os cenários podem ser abordados com técnicas de programação clássica, comparando, por exemplo, a distância medida pelo sensor com uma distância limite ou verificando se o código introduzido corresponde ao código correto. Por outro lado, se for necessário que uma câmara reconheça o nosso rosto para permitir a entrada no edifício, então, como veremos em breve, será necessário recorrer à inteligência artificial.

História da Inteligência Artificial

A história da inteligência artificial (IA) é assinalada por marcos significativos que refletem a evolução da tecnologia e da compreensão humana sobre a inteligência. Desde os seus conceitos iniciais até às aplicações atuais, a IA passou por várias transformações, influenciadas por avanços na investigação, necessidades sociais e desenvolvimentos tecnológicos.

Primeiras bases (década de 1950)

A jornada da IA começou nos anos 50, década em que se lançou as bases nesse campo. Em 1950, Alan Turing publicou o seu artigo seminal "Computing Machinery and Intelligence", introduzindo o Teste de Turing, cujo objetivo era avaliar a capacidade de uma máquina exibir um comportamento inteligente indistinguível do de um ser humano. No ano seguinte, Marvin Minsky e Dean Edmonds criaram a SNARC, a primeira rede neural artificial (ANN), que simulava uma rede de neurónios usando válvulas de vácuo. Em 1956, durante um workshop organizado por John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon, foi criado o termo "inteligência artificial". Este evento é amplamente considerado o momento fundador da IA como campo distinto.

A ascensão do *Machine Learning* (anos 60-70)

Nos anos 60, surgiram os primeiros programas de IA, como o Eliza, um chatbot capaz de manter conversas simples, e o Shakey, o primeiro robô móvel com capacidades de IA. No entanto, este período também enfrentou desafios. As limitações das redes neurais iniciais foram destacadas em 1969 quando Marvin Minsky e Seymour Papert publicaram "*Perceptrons*", o que levou a um declínio na investigação em redes neurais em favor de abordagens simbólicas de IA. Os anos 70 marcaram um "inverno da IA", caraterizado pela redução de financiamento e interesse devido às expectativas não correspondidas. Um relatório fundamental de James Lighthill em 1973 criticou a investigação em IA no Reino Unido, levando a cortes significativos no apoio governamental.

Renascimento e expansão (anos 80-90)

A IA viveu um renascimento nos anos 80, com a comercialização das máquinas Lisp e o renovado interesse pelos sistemas especialistas. Este período assistiu a avanços na representação do conhecimento e nas técnicas de raciocínio, permitindo aplicações de IA mais sofisticadas. A introdução de algoritmos de retropropagação para o treino de redes neurais multicamadas revitalizou a investigação na área. Na década de 90, a IA começou a ser integrada em aplicações práticas, como o reconhecimento de voz e o processamento de vídeo. Em 1997, o Deep Blue da IBM ganhou destaque ao derrotar o campeão mundial de xadrez Garry Kasparov, demonstrando o potencial da IA no pensamento estratégico.

Era Moderna (Anos 2000-Presente)

O século XXI testemunhou uma explosão das capacidades da IA, impulsionada pelos avanços no *machine learning*, particularmente no *deep learning* (aprendizagem profunda). Tecnologias como o IBM Watson, assistentes pessoais como a Siri e a Alexa, sistemas de reconhecimento facial e modelos generativos como o GPT tornaram-se parte integrante da vida quotidiana. O surgimento do *big data* e o aumento do poder computacional permitiram a estes sistemas aprender com vastas quantidades de informação, levando a melhorias significativas no desempenho em vários domínios. Hoje, as discussões sobre IA também abrangem considerações éticas e impactos sociais. À medida que os sistemas de IA se tornam mais prevalentes, questões relacionadas com privacidade, preconceito e responsabilidade são cada vez mais debatidas.

Marcos Principais

- **1950**: Alan Turing propõe o Teste de Turing.
- **1956**: A Conferência de Dartmouth estabelece a IA como campo de estudo.
- **1966**: Criado o ELIZA, um dos primeiros programas de processamento de linguagem natural.
- **1997**: Deep Blue da IBM derrota Garry Kasparov.
- **2012**: Avanços na aprendizagem profunda (deep learning) com a vitória do AlexNet na competição ImageNet.
- **Anos 2020**: A IA torna-se essencial em várias indústrias, levantando questões éticas e regulatórias.

1.2 Teste de Turing

O teste de Turing, proposto por Alan Turing (1950), foi concebido como uma experiência mental que evitasse a ambiguidade filosófica da questão "Pode uma máquina pensar?". Um computador passa no teste se um interrogador humano, após colocar algumas perguntas por escrito, não conseguir distinguir se as respostas escritas vêm de uma pessoa ou de um computador. O Capítulo 28 discute em detalhe o teste e se

um computador seria realmente inteligente se o passasse. Para já, salientamos que programar um computador para passar num teste aplicado de forma rigorosa é um enorme desafio. O computador necessitaria das seguintes capacidades:

- processamento de linguagem natural para comunicar com sucesso numa linguagem humana;
- representação do conhecimento para armazenar o que sabe ou ouve;
- raciocínio automatizado para responder a perguntas e tirar novas conclusões;
- machine learning para se adaptar a novas circunstâncias e para detetar e extrapolar padrões.

Turing considerava desnecessária a simulação física de uma pessoa para demonstrar inteligência. No entanto, outros investigadores propuseram um teste de Turing total, que exige interação com objetos e pessoas no mundo real. Para passar neste teste total, um robô necessitaria de:

- visão computacional e reconhecimento de voz para perceber o mundo;
- robótica para manipular objetos e movimentar-se.

Estas seis disciplinas compõem a maior parte da inteligência artificial. No entanto, os investigadores da área têm dedicado pouco esforço a passar o teste de Turing, acreditando ser mais importante estudar os princípios subjacentes da inteligência. A busca pelo "voo artificial" só teve sucesso quando os engenheiros deixaram de imitar os pássaros e começaram a usar túneis de vento e a estudar a aerodinâmica. Os manuais de engenharia aeronáutica não definem o objetivo do campo como a criação de "máquinas que voem exatamente como pombos ao ponto de enganarem outros pombos".

Uma máquina pode pensar?

Curiosamente, o desenvolvimento da inteligência artificial tem sido repleto de obstáculos.

A pergunta é: "Pode uma máquina pensar?". Em 1950, o matemático inglês Alan Turing marcou o início do desenvolvimento do que hoje conhecemos como inteligência artificial. Poucos anos após esta grande ideia, em 1956, os cientistas de renome John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon reuniram-se numa conferência em Dartmouth, que durou um mês, com o objetivo de definir os objetivos e protocolos de investigação na área. Foi então que o termo "inteligência artificial" foi criado e oficialmente apresentado.

Na altura do surgimento da IA, os computadores eram muito diferentes dos de hoje: tinham menor capacidade e velocidade, e eram bastante mais caros. Por isso, a investigação nesta área exigia soluções de design diferentes e dependia de apoio oficial e de fontes de financiamento estáveis.

Durante a primeira vaga de desenvolvimento da IA, que durou até ao início dos anos 70 do século 20, surgiram programas interessantes. Um dos primeiros foi o *programa Logic Theorist*, escrito em 1956, que explorava as capacidades da lógica matemática e da dedução. Este programa conseguiu provar 38 dos primeiros 52 teoremas do famoso livro *Principia Mathematica*. Outro exemplo é o *ELIZA*, que simulava conversas com utilizadores seguindo regras simples em inglês. A motivação para investigar comunicação surgiu da proposta de Turing de que máquinas inteligentes deveriam ser capazes de comunicar sem que o interlocutor percebesse que falava com uma máquina — este teste é hoje conhecido como o Teste de Turing.

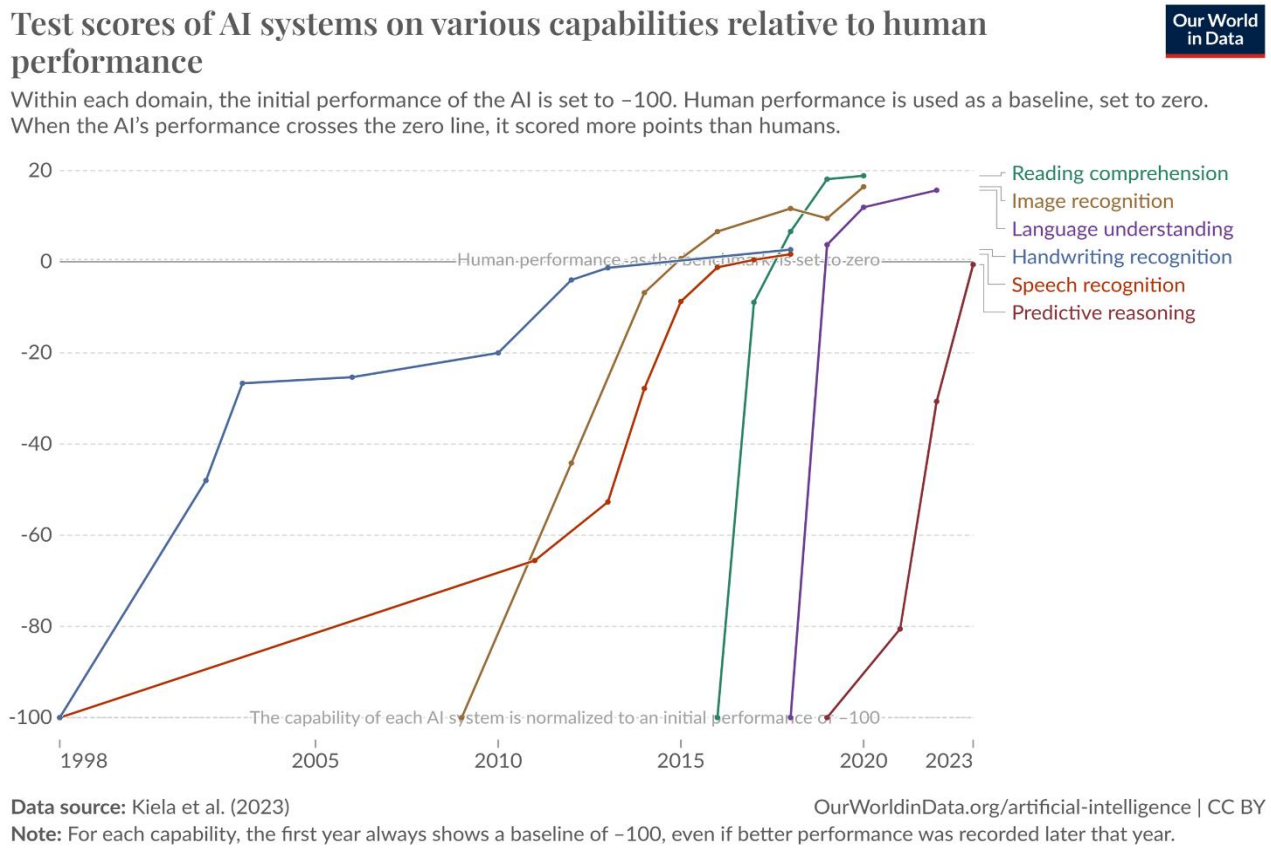
Até ao início da década de 1970, muitas ideias emergiram e mais tarde permitiram avanços na IA moderna. Um exemplo é a ideia do perceptron, a base das redes neurais de hoje, introduzida em 1957 por Frank Rosenblatt. Segundo Rosenblatt, o perceptron tinha capacidade de aprender, tomar decisões e traduzir línguas — embora tenha levado muito tempo a confirmar tais capacidades.

Devido à falta de financiamento, esta primeira vaga foi seguida pelo que se chama o primeiro "inverno da IA", causado em parte por projetos demasiado ambiciosos cujos resultados foram limitados pelas capacidades dos computadores e pela escassez de dados disponíveis.

Um momento marcante na história da IA foi em 1997, quando o *sistema Deep Blue* da IBM venceu o campeão mundial de xadrez Garry Kasparov. O *sistema Deep Blue* era um exemplo de sistema especialista, que usava uma base de dados com inúmeras regras do tipo "se-então" e lógica para imitar o raciocínio de especialistas e chegar a respostas corretas. Um efeito semelhante no desenvolvimento da inteligência artificial foi provocado quase 20 anos depois, em 2016, pelo sistema AlphaGo da DeepMind (Google), ao derrotar o campeão mundial Lee Sedol no jogo de Go.

Em 2011, a capacidade das máquinas para responder a perguntas em inglês foi demonstrada pelo sistema *Watson da IBM*, no concurso *Jeopardy*!. *Watson* venceu dois campeões anteriores, ao responder corretamente e mais rapidamente às perguntas. Segundo fontes da altura, *Watson* era capaz de processar 500 GB de conteúdo por segundo — o equivalente a cerca de um milhão de livros.

Por outro lado, a capacidade das máquinas para reconhecer e distinguir objetos em imagens foi demonstrada em 2012 pela equipa *Google X*, que criou um programa capaz de reconhecer gatos em imagens. Depois de ver mais de 10 milhões de imagens em 3 dias, o programa aprendeu a reconhecer gatos. Desde então, a capacidade dos sistemas de reconhecimento tem melhorado bastante e, em muitas aplicações, superam a precisão humana. A imagem abaixo mostra a tendência de desenvolvimento em áreas como reconhecimento de texto manuscrito, reconhecimento de fala, reconhecimento de imagens e, mais recentemente, tarefas de compreensão de linguagem.



A imagem foi retirada de https://ourworldindata.org/brief-history-of-ai

Estas conquistas foram o prelúdio para um desenvolvimento muito mais promissor da IA, graças à Internet, à Web e à maior disponibilidade de dados, bem como ao aumento exponencial da capacidade de processamento dos computadores face aos anos 50. Isto levou também a uma mudança de paradigma no campo, passando-se de sistemas baseados em lógica para sistemas baseados em estatística.

A história da IA também está ligada aos robôs — não só em romances e filmes de ficção científica, mas também na criação de robôs reais. Em 1950, o cientista americano Claude Shannon criou um rato capaz de encontrar a saída de um labirinto, chamado Theseus, em homenagem à mitologia grega. Em 1966, uma equipa do Stanford Research Institute iniciou o desenvolvimento do robô Shakey, o primeiro robô capaz de se mover e fazer inferências sobre o ambiente. Em 1989, a equipa da Universidade Carnegie Mellon criou o primeiro veículo autónomo, o ALVINN (*Autonomous Land Vehicle In a Neural Network*), que percorreu com sucesso 145 quilómetros a uma velocidade de 110 km/h entre outros carros.

1.3 Áreas da Inteligência Artificial

Nesta secção, vamos explorar algumas áreas da inteligência artificial. As fronteiras entre elas não são rígidas, e muitas vezes as técnicas utilizadas para resolver problemas de uma área podem ser úteis para resolver problemas de outra. O verdadeiro poder da inteligência artificial está, na verdade, em conectar todas as áreas.

Visão computacional

A visão computacional é um campo da inteligência artificial que se dedica ao desenvolvimento de algoritmos e ferramentas que conferem aos computadores a capacidade de compreender o mundo visual como os humanos. Inclui tarefas como o reconhecimento de objectos em imagens, a compreensão das suas relações, o reconhecimento de cores e texturas, bem como o reconhecimento de movimentos, acções e suas características. Dado que este campo se foca principalmente na análise de imagens e vídeos, iremos conhecer algumas das tarefas mais comuns.

A classificação de imagens é utilizada para determinar que tipo de objecto está presente numa imagem. Por exemplo, determinar se existe ou não um cão numa imagem é uma tarefa de classificação. **A detecção de objectos** localiza objetos e responde à questão de onde estão situados na imagem. Por exemplo, enquadrar um cão e um gato presentes numa imagem. **A segmentação de imagem** permite determinar a forma exacta dos objectos que aparecem na imagem — uma separação mais precisa dos contornos do cão e do gato.

As três principais tarefas da visão computacional no trabalho com imagens

Estas tarefas são aplicáveis em diversas áreas como a condução autónoma, análise de imagens médicas ou de satélite, e permitem uma pesquisa e organização mais eficiente de imagens.

Em que tarefas se enquadram os seguintes problemas:

- Determinar se há um peão na imagem.
- Separar os contornos dos semáforos, passeios e peões na imagem,
- Identificar onde está o sinal de trânsito na imagem?

No processamento de vídeo, as tarefas mais comuns são o seguimento de objetos, o reconhecimento de ações e o posicionamento.

O seguimento de objetos, como o nome sugere, permite rastrear objetos em vídeo em tempo real. Por exemplo, seguir um carro vizinho durante uma condução autónoma ou acompanhar os movimentos de um jogador num jogo.

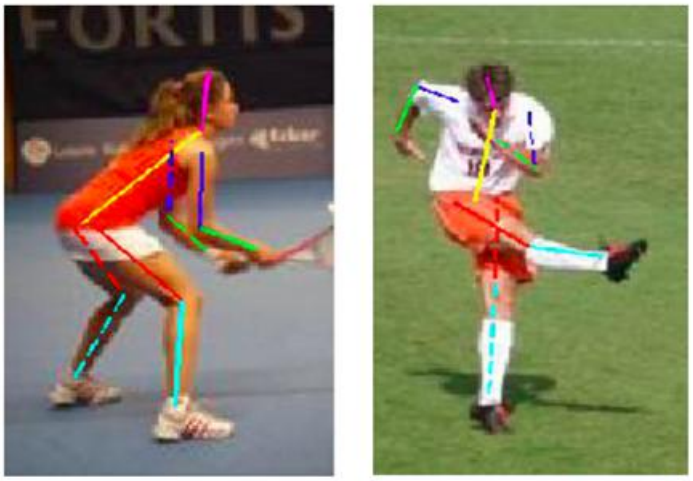
Seguimento de objetos

(imagem retirada de https://docs.ultralytics.com/modes/track/)

A tarefa **de reconhecer ações** (eng. ***action recognition***) é a capacidade de reconhecer e nomear uma ação que está presente num vídeo, por exemplo, saltar para dentro de água ou fechar uma janela. Estas tarefas ajudam-nos a compreender melhor o conteúdo de vídeo e a facilitar a sua pesquisa.

Exemplos de reconhecimento de ações em vídeos

A estimativa de pose é uma tarefa que identifica a figura humana num vídeo em tempo real, extraindo pontos-chave do esqueleto (olhos, nariz, boca, ombros, cotovelos, ancas, mãos, joelhos e pés). Estas tarefas são úteis em animações interativas, realidade aumentada e outras aplicações..



Tarefa de Reconhecimento de Posição de Objeto: Imagens do conjunto de dados Leeds Sports Pose

Que tarefa precisamos de resolver para:

- *Analisar se estamos sentados corretamente,*
- *Reconhecer a saída para o quintal do animal de estimação,*
- *Acompanhar os movimentos de um cliente numa loja?*

Mais à frente, iremos abordar os conjuntos de dados utilizados na visão por computador e as redes neurais convolucionais, um tipo especial de rede usada em tarefas de processamento de imagem e vídeo.

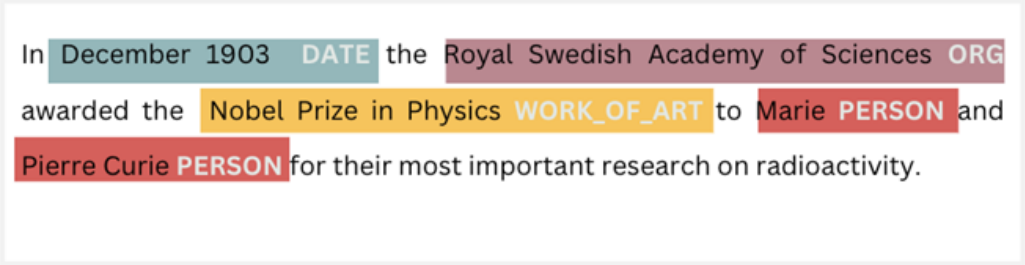
Processamento de Linguagem Natural

O Processamento de Linguagem Natural *(PNL)* é um campo da inteligência artificial que lida com tarefas relacionadas com a compreensão e geração de linguagem natural. Existem mais de 7.000 línguas no mundo, cada uma com vocabulário, regras gramaticais e significados próprios. Algumas tarefas comuns encontradas no processamento de linguagem natural serão descritas abaixo.

Assim como na classificação de imagens, na **classificação** de texto tentamos determinar se um texto pertence a uma categoria ou não. Por exemplo, é um artigo de jornal sobre o tema do desporto, está escrito em espanhol, é positivo, ou seja, contém algum comentário elogioso, seja verdadeiro ou falso, e coisas do género.

Classificação de texto

O Reconhecimento de Entidade Nomeada é uma tarefa relacionada ao reconhecimento de algumas partes do texto que são relevantes para sua análise posterior. Estes são geralmente os nomes das pessoas que aparecem nele, datas, nomes de geolocalização ou em alguns textos profissionais, por exemplo, no campo da medicina, sintomas ou nomes de doenças. Por um único nome, essas partes do texto são chamadas de entidades.



Exemplo de marcação de entidades nomeadas

A tarefa da **Máquina de Tradução** é desenvolver ferramentas que nos permitam traduzir conteúdo de uma língua para o conteúdo de outra língua. Concordaremos que esta tarefa é a base para uma comunicação bem sucedida e a disponibilidade de informação, mas também que é complicada porque cada língua e cada cultura que a língua representa tem as suas próprias peculiaridades, tais como frases, expressões idiomáticas, gírias ou sarcasmo que são muito difíceis de traduzir (como traduzir *um cérebro estéril para pasto?*).

Os sistemas de perguntas e respostas lidam com a questão de como encontrar uma resposta concreta para uma determinada pergunta. São generalizações dos sistemas clássicos de recuperação de informação e permitem-nos obter mais facilmente a informação de que precisamos.

Resumir todas as informações importantes de várias fontes diferentes é conhecido como **uma tarefa de sumarização**. Tal como na tarefa anterior, os resumos resultantes das tarefas de sumarização devem facilitar a leitura de uma grande quantidade de conteúdo ou servir para nos relembrar informações e detalhes importantes do que lemos.

Sumarização

Além de tarefas relacionadas ao texto e conteúdo textual, o processamento de linguagem natural também lida com análise de fala. Há duas tarefas em particular: *fala para texto (speech-to-text)* e vice-versa, *texto para fala (text-to-speech)*. Estes dois grupos de tarefas são especialmente importantes para o desenvolvimento de assistentes pessoais, programas como *Siri*, *Cortana* ou *Alexa*, que podem entender mensagens de voz e executar uma tarefa solicitada, como por exemplo, definir um alarme ou ligar para alguém da lista telefônica.

Em que tarefas se enquadram os seguintes problemas:

Extraindo o nome da organização no texto,

Descobrir quem é o autor do livro,

Qual é o significado da palavra Netizen ?

A Inteligência Artificial Generativa (AGI) é um campo da inteligência artificial que lida com a geração de conteúdo como imagens, texto, áudio ou vídeo. Ao longo dos últimos anos, os avanços neste campo têm sido impressionantes.

ChatGPT é um programa que marcou um avanço significativo na área da geração de conteúdos textuais. De acordo com as instruções do utilizador, os chamados *prompts*, pode gerar conteúdo de texto apropriados. No entanto, é importante lembrar que os textos gerados desta forma não têm de ser absolutamente rigorosos — podem conter dados incorretos, referências inventadas ou conteúdo ofensivo. Por isso, tudo o que for gerado pelo programa deve ser verificado antes de ser utilizado. Se criares uma conta em chat.openai.com , podes experimentar por ti próprio como funciona o programa *ChatGPT*. Por trás *do programa ChatGPT* está a comunidade *OpenAI*.

O programa StableDiffusion, ao contrário *do ChatGPT*, que gera texto, gera imagens com base em instruções. Por exemplo, todas as imagens listadas abaixo foram geradas por este programa. É um software de código aberto (open source) e pode ser descarregado a partir do repositório oficial no GitHub juntamente com o respetivo código. Podes testar o próprio modelo em https://stablediffusionweb.com/. Deve ter em consideração que este serviço é utilizado por um grande número de pessoas de forma gratuita e, por vezes, pode não estar disponível. O nome do próprio programa corresponde a uma técnica bastante popular utilizada nesta área.

Exemplos de imagens geradas pelo StableDiffusion

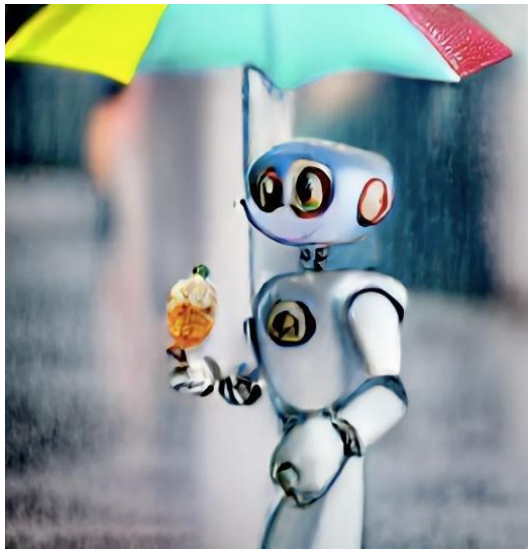
Muitas vezes, ao gerar imagens, também é possível selecionar o estilo desejado da nova imagem. Esta técnica é conhecida como **transferência de estilo** (**style transfer**). Pode ver um exemplo na imagem abaixo.

Além de imagens e texto, a inteligência artificial também pode gerar conteúdo de áudio. Neste link, podes testar o *programa MusicGen da Meta*, descrevendo por palavras que tipo de música desejas gerar e, eventualmente, fornecendo um exemplo para transferência de estilo. Depois, podes ouvir o conteúdo que criaste. Também pode experimentar programas que realizam transferência de estilo ao gerar voz (imitando a voz de outra pessoa) ou que compõem música com base no que já 'ouviram' nos dados. Um desses projetos é o Magenta. Podes aceder através do link https://magenta.github.io/listen-to-transformer/.

Atividades sugeridas:

- Peça ao ChatGPT para criar um quiz sobre IA e veja quantas questões consegue responder.

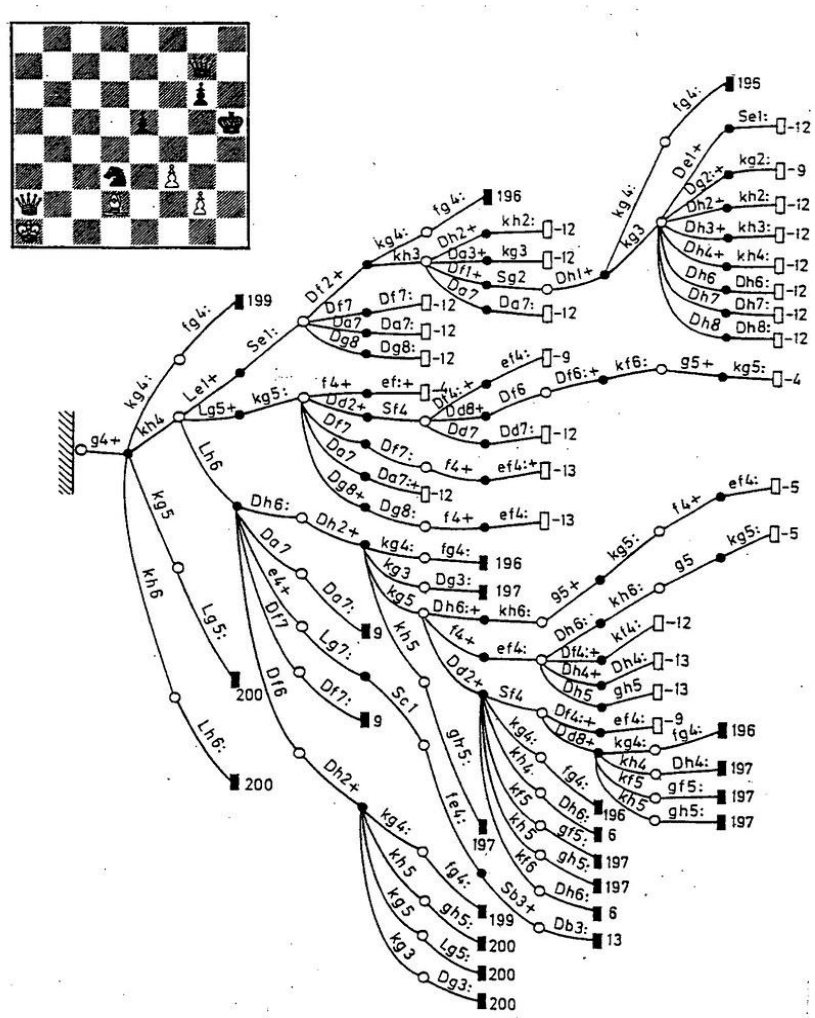
- Experimente dar instruções ao StableDiffusion ou Dalle-mini para gerar uma imagem semelhante à apresentada.



O programa DALL-E da OpenAI também gera imagens com base nas diretrizes do usuário. O programa dalle-mini é uma versão publicamente disponível deste programa. Está disponível em https://huggingface.co/spaces/dalle-mini/dalle-mini .

Jogar jogos

Uma das primeiras tarefas em que se tentou aplicar a inteligência artificial foi o jogo de xadrez. Com a sua vitória sobre o Grande Mestre Garry Kasparov, esta área de investigação ganhou bastante simpatia e apoio por parte da comunidade ligada à inteligência artificial. Apesar de seguirem um conjunto preciso de instruções e regras, os jogos são caraterizados por um fenómeno conhecido como explosão combinatória — ou seja, um número muito elevado de possíveis escolhas de ações após um determinado número de jogadas. Isto significa que os jogos não permitem encontrar soluções eficazes recorrendo apenas às técnicas de programação tradicionais dentro de um tempo razoável. Para além do xadrez, a inteligência artificial também se destacou no jogo Go, com o programa AlphaGo, e ainda em jogos de vídeo Atari e jogos de estratégia como Dota, Starcraft, entre outros. No final deste curso, vais aprender mais sobre uma área chamada *aprendizagem por reforço (reinforcement learning)*, que é amplamente aplicada neste domínio.



Explosão combinatória no jogo de xadrez

Atividade sugerida:

- Verifique se o seu jogo favorito usa IA em alguma parte.

Robótica

A inteligência artificial é frequentemente necessária para melhorar os comportamentos e as capacidades de objetos físicos, como robôs, máquinas industriais, automóveis, drones, eletrodomésticos ou dispositivos médicos. A informação sobre o mundo chega a estes objetos através de instruções por voz, gravações de câmaras ou medições de sensores, sendo a sua tarefa processar esses dados e transformá-los em decisões. O papel da inteligência artificial nesta área é melhorar as capacidades desses objetos e ajudá-los a adotar comportamentos inteligentes.

1.4 Regulação da Inteligência Artificial, Desafios Legais e Éticos

O rápido avanço das tecnologias de inteligência artificial (IA) tem gerado discussões urgentes em torno da sua regulação. À medida que os sistemas de IA se tornam cada vez mais integrados em diversos sectores — incluindo a saúde, as finanças e os transportes — a necessidade de quadros legais eficazes e de directrizes éticas torna-se cada vez mais importante. Este texto explora as complexidades da regulação da IA, destacando os desafios legais e éticos que surgem neste cenário em constante evolução.

A necessidade de regulamentação

As tecnologias de IA apresentam características únicas que as diferenciam das tecnologias tradicionais. Muitas vezes funcionam como 'caixas negras', em que os processos de tomada de decisão não são transparentes, nem sequer para os seus próprios criadores. Esta opacidade levanta preocupações significativas em relação à responsabilização, ao enviesamento e ao potencial de causar danos. Uma vez que os sistemas de IA podem influenciar áreas críticas como decisões de emprego, resultados de justiça criminal e segurança pública, a necessidade de uma regulação eficaz é elevada.

Principais motivações para a regulação:

- **Segurança e bem-estar**: Garantir que os sistemas de IA não comprometem a segurança pública nem os direitos individuais é uma preocupação fundamental.Incidentes envolvendo veículos autônomos ou algoritmos tendenciosos em processos de contratação ressaltam a necessidade de supervisão regulatória.
- **Responsabilização**: É essencial estabelecer mecanismos claros de responsabilização para determinar quem é responsável quando os sistemas de IA causam danos ou tomam decisões erradas.
- **Uso ético**: Dado que os sistemas de IA podem perpetuar enviesamentos existentes nos dados de treino, a regulamentação deve assegurar padrões éticos que previnam a discriminação e o tratamento injusto.

Desafios legais na regulamentação da IA

O panorama legal em torno da regulação da IA é complexo e frequentemente não acompanha os avanços tecnológicos. Os quadros legais tradicionais podem não abordar adequadamente as especificidades dos sistemas de IA.

1. Definição de Responsabilidade

Um dos maiores desafios é determinar quem é responsável quando um sistema de IA causa danos. Surge a questão de saber se a responsabilidade deve recair sobre os programadores, os utilizadores ou sobre o próprio sistema de IA. Por exemplo: se um veículo autónomo estiver envolvido num acidente, o fabricante deve ser responsabilizado, ou será o proprietário?

2. Questões de Propriedade Intelectual

Com o aumento da criação de conteúdos gerados por IA, surgem questões relativas aos direitos de propriedade intelectual. Quem detém os direitos sobre as obras criadas por IA? As leis atuais podem não cobrir suficientemente estes cenários, originando potenciais conflitos sobre propriedade e direitos de autor.

3. Preocupações com a Privacidade de Dados

Os sistemas de IA dependem fortemente de dados para o seu treino e funcionamento. A recolha e utilização de dados pessoais levantam importantes questões de privacidade. A regulamentação deve equilibrar a necessidade de dados para melhorar as capacidades da IA com o direito dos indivíduos à privacidade e à proteção dos seus dados.

Desafios Éticos na Regulação da IA

Para além das questões legais, os desafios éticos desempenham um papel crucial na definição da regulação da IA.

1. Parcialidade e Equidade

Os sistemas de IA podem inadvertidamente perpetuar enviesamentos presentes nos dados de treino, resultando em resultados injustos. Os reguladores devem estabelecer diretrizes para assegurar a justiça e mitigar o enviesamento nos algoritmos de IA. Isso envolve não apenas soluções técnicas, mas também um compromisso com princípios éticos que priorizam o tratamento equitativo.

2. Transparência e Explicabilidade

A natureza de 'caixa negra' de muitos sistemas de IA dificulta os esforços de transparência. Os utilizadores e as pessoas afectadas muitas vezes não têm conhecimento de como as decisões são tomadas. A regulamentação deve promover a explicabilidade, garantindo que os indivíduos compreendam o raciocínio por trás das decisões de sistemas de IA.

3. Consentimento informado

À medida que as tecnologias de IA influenciam cada vez mais decisões pessoais — como aprovações de crédito ou candidaturas de emprego — torna-se essencial garantir o consentimento informado. Os indivíduos devem ser informados sobre como os seus dados são utilizados e de que forma a IA impacta os processos de tomada de decisão que lhes dizem respeito.

Abordagens da regulamentação

Perante estes desafios, surgiram várias abordagens para a regulação da IA.

1. Quadros Baseados no Risco

A União Europeia propôs uma abordagem baseada no risco para a regulação da IA, categorizando as aplicações em função dos níveis de risco — de riscos mínimos a riscos inaceitáveis. Este quadro permite uma regulamentação adaptada que aborda preocupações específicas associadas a diferentes tipos de aplicações de IA.

2. Supervisão Contínua

A regulação da IA exige uma vigilância contínua devido à sua natureza em rápida evolução. As entidades reguladoras devem adaptar-se aos novos desenvolvimentos tecnológicos e avaliar de forma constante o impacto das regulamentações existentes na sociedade.

3. Governação Colaborativa

Uma regulação eficaz poderá implicar uma colaboração entre governos, actores da indústria e organizações da sociedade civil. O envolvimento de diferentes perspectivas pode conduzir a regulamentações mais abrangentes, que considerem os diversos interesses e questões éticas.

Conclusão

A regulação da inteligência artificial apresenta desafios legais e éticos multifacetados que requerem uma consideração cuidadosa e medidas proativas. À medida que a IA continua a penetrar em várias áreas da vida quotidiana, é essencial estabelecer quadros reguladores robustos que salvaguardem o bem-estar público, assegurem a responsabilização e garantam o cumprimento de padrões éticos. Ao abordar estes desafios através de uma governação colaborativa e de abordagens regulatórias adaptáveis, a sociedade poderá tirar partido dos benefícios da IA, mitigando de forma eficaz os seus riscos.

1.5 Estreita, Geral e Superinteligência

Agora que já conhecemos os campos de aplicação da inteligência artificial e o seu alcance, podemos também introduzir conceitos como inteligência artificial estreita e geral, superinteligência e a singularidade da inteligência artificial.

Vimos que os exemplos de programas de inteligência artificial que mencionámos estão focados na resolução de uma tarefa específica — por exemplo, jogar xadrez, traduzir línguas, segmentar imagens e similares. Dizemos que estes programas possuem **Inteligência Artificial Estreita** (*narrow AI*). Em contraste, temos a **Inteligência Artificial Geral** (*General AI*), que pretende aproximar-se mais da inteligência humana, permitindo a resolução de uma grande variedade de tarefas distintas. De acordo com os principais investigadores da área, programas deste tipo ainda estão numa fase embrionária e não deverão ser desenvolvidos num futuro próximo.

A **superinteligência** refere-se a uma inteligência superior à humana. Tal inteligência poderia ajudar-nos a resolver problemas como o aquecimento global, garantir alimento suficiente para todos ou encontrar a cura para o cancro. Hipoteticamente, poderia também sair do nosso controlo, continuar a evoluir autonomamente e, de alguma forma, pôr em risco a humanidade. **A singularidade da IA** é um conceito teórico que denota o domínio da inteligência artificial sobre a inteligência humana e é um tema frequentemente abordado em filmes e livros de ficção científica.

Observar a inteligência artificial por esta perspetiva é de particular interesse para filósofos, historiadores e investigadores das ciências sociais. O historiador israelita Yuval Noah Harari e o filósofo sueco Nick Bostrom escreveram sobre este tema. Encontram-se também muitos vídeos sobre isto no YouTube.

Tente pensar noutro exemplo de aplicação da superinteligência. Que problemas persistem na sociedade, ciência e tecnologia que, apesar do desenvolvimento, a humanidade ainda não conseguiu resolver?

Explore o significado do termo risco existencial. Qual é a sua perspetiva sobre este conceito?

O campo da inteligência artificial (IA) abrange uma variedade de sistemas e capacidades, que podem ser categorizados em três tipos principais: **Inteligência Estreita, Inteligência Geral e Superinteligência**. Cada categoria representa um nível diferente de complexidade e capacidade nos sistemas de IA, refletindo a evolução tecnológica e o seu potencial impacto na sociedade. Compreender estas distinções é fundamental para entender o panorama atual e as implicações futuras da IA.

Inteligência estreita (IA fraca)

A inteligência estreita, também chamada IA fraca, é concebida para executar tarefas específicas ou resolver problemas concretos. Estes sistemas de IA são muito eficazes nas funções para as quais foram projetados, mas não conseguem operar fora desses parâmetros.

Características da Inteligência Estreita:

- **Funcionalidade específica da tarefa**:Criados para lidar com tarefas como reconhecimento de imagens, tradução de idiomas ou jogar xadrez. Operam eficazmente dentro do seu âmbito limitado, mas não têm capacidade de raciocínio geral.
- **Aprendizagem baseada em dados**: Dependem fortemente de grandes volumes de dados para treino. Utilizam algoritmos de machine learning para analisar padrões e tomar decisões com base nesses dados.

- **Falta de autoconsciência**: IA estreita não possuem consciência ou entendimento do contexto ou das consequências das suas acções. Operam através de algoritmos e regras predefinidas sem compreender o contexto ou as implicações das suas ações.

Exemplos de Inteligência Estreita:

- **Assistentes virtuais**: Aplicações como a Siri ou a Alexa executam tarefas simples, como definir lembretes ou responder a perguntas, mas não conseguem manter conversas fora das suas capacidades programadas.
- **Sistemas de recomendação**: Plataformas como Netflix e Amazon usam IA estreita para sugerir conteúdo com base nas preferências do usuário, analisando o comportamento passado para prever escolhas futuras.
- **Veículos autônomos**: Embora os carros autónomos utilizem algoritmos complexos para navegar nas estradas, eles ainda estão limitados a tarefas de direção e não podem executar outras funções cognitivas semelhantes às humanas.

Inteligência Geral (IA forte)

A Inteligência Geral Artificial (AGI — Artificial General Intelligence) refere-se a uma forma teórica de IA que teria a capacidade de compreender, aprender e aplicar conhecimento em diversas tarefas, de forma semelhante à inteligência humana.

Características da Inteligência Geral:

- **Aprendizagem Versátil**: A AGI é capaz de generalizar conhecimento entre diferentes domínios e adaptar-se a novas situações sem necessidade de reprogramação extensiva.
- **Raciocínio e resolução de problemas**: Ao contrário da IA estreita, a AGI pode raciocinar através de problemas complexos, entender conceitos abstratos e tomar decisões com base em informações incompletas.
- **Interação semelhante à humana**: os sistemas AGI seriam capazes de se envolver em conversas naturais com humanos, demonstrando compreensão emocional e consciência social.

Estado Atual da Inteligência Geral:

Atualmente, a AGI continua a ser um conceito teórico. Apesar dos progressos em machine learning e redes neurais, ainda não existe um sistema que possua a gama completa de capacidades cognitivas humanas. Os investigadores continuam a explorar caminhos para alcançar a AGI, concentrando-se no desenvolvimento de algoritmos que podem aprender de forma mais flexível e autônoma.

Superinteligência

A superinteligência refere-se a um tipo hipotético de IA que ultrapassaria a inteligência humana em praticamente todos os aspetos. Este conceito levanta questões profundas sobre o futuro da humanidade e os dilemas éticos de criar máquinas que poderiam superar os seus próprios criadores.

Características da Superinteligência:

- **Crescimento exponencial**: Sistemas superinteligentes teriam a capacidade de autoaperfeiçoamento rápido, levando a uma "explosão de inteligência" onde as máquinas poderiam melhorar suas próprias capacidades além da compreensão humana.

- **Resolução de problemas globais**: Poderia resolver problemas complexos como as alterações climáticas ou a erradicação de doenças de forma mais eficiente que qualquer ser humano ou grupo.

- **Considerações éticas**: O desenvolvimento da superinteligência coloca dilemas éticos significativos em relação ao controle, à segurança e às possíveis consequências para a sociedade. Garantir que os sistemas superinteligentes se alinhem com os valores humanos é uma preocupação crítica.

Implicações Teóricas:

A discussão em torno da superinteligência muitas vezes inclui cenários sobre a "singularidade", um ponto em que o crescimento tecnológico se torna incontrolável e irreversível. Este conceito suscitou debates entre cientistas, especialistas em ética e futuristas sobre os riscos potenciais associados à criação de entidades que poderiam operar independentemente da supervisão humana.

Desafios para avançar em direção à superinteligência geral e à superinteligência

Embora a busca por IA geral e superinteligente apresente possibilidades empolgantes, vários desafios devem ser enfrentados:

1. **Limitações técnicas**: O desenvolvimento de AGI e da superinteligência levanta questões sobre autonomia, autoridade de decisão e potencial uso indevido, por exemplo em contextos militares ou de vigilância.

2. **Preocupações éticas**: O desenvolvimento de AGI e da superinteligência levanta questões sobre autonomia, autoridade de decisão e potencial uso indevido, por exemplo em contextos militares ou de vigilância

3. **Medidas de segurança**: É crucial garantir que sistemas avançados de IA operem de forma segura. São necessárias estratégias e protocolos robustos para evitar consequências indesejadas.

4. **Perceção pública**: Os mal-entendidos sobre as capacidades da IA podem levar a expectativas ou receios irrealistas relativamente ao seu impacto na sociedade. A educação pública é essencial para promover discussões informadas sobre as tecnologias de IA.

Conclusão

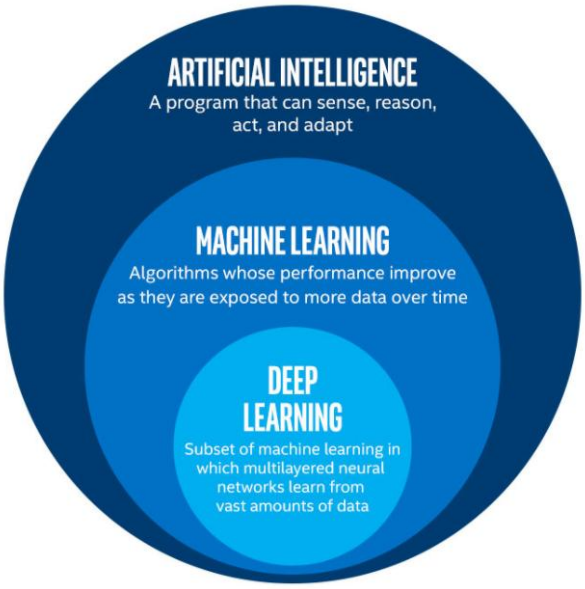
As distinções entre inteligência estreita, inteligência geral e superinteligência mostram a diversidade existente no campo da inteligência artificial. Enquanto a IA estreita já é uma realidade em muitas aplicações, a AGI continua a ser uma meta ambiciosa para pesquisadores em todo o mundo. A perspetiva da superinteligência traz tanto entusiasmo como cautela, à medida que a sociedade enfrenta os potenciais desafios de criar máquinas mais inteligentes que os próprios humanos. À medida que a tecnologia avança, será essencial abordar as questões éticas e garantir um desenvolvimento responsável da IA, para que possamos beneficiar do seu potencial enquanto mitigamos os riscos de forma eficaz.

2. Machine Learning

Bem-vindo ao tópico de **Machine Learning (ML)**! Machine learning é um subconjunto da IA que se concentra no desenvolvimento de algoritmos que permitem aos computadores aprender e fazer previsões com base em dados. Nesta secção, obterá uma compreensão aprofundada dos conceitos básicos de machine learning, nos seus diferentes tipos (supervisionado, não supervisionado, semi supervisionado e por reforço) e da importância dos dados nesse processo. Também abordaremos as etapas envolvidas na construção e validação de modelos de machine learning e como esses modelos podem ser aplicados para resolver problemas do mundo real.

2.1 A Relação entre Inteligência Artificial e Machine Learning

A Inteligência Artificial (IA) e o Machine Learning (ML) são dois campos inter-relacionados que ganharam atenção significativa nos últimos anos devido ao seu impacto transformador na tecnologia e na sociedade. Embora muitas vezes usados de forma intercambiável, eles representam conceitos distintos dentro do cenário mais amplo da inteligência computacional. Entender sua relação é crucial para entender como os sistemas modernos de IA operam e evoluem.



1. **Inteligência Artificial (IA)** engloba uma ampla gama de tecnologias e metodologias destinadas a criar sistemas capazes de executar tarefas que normalmente exigem inteligência humana. Essas tarefas incluem raciocínio, resolução de problemas, percepção, compreensão da linguagem natural e muito mais. O principal objetivo da IA é desenvolver máquinas que possam imitar funções cognitivas associadas aos seres humanos, permitindo-lhes executar tarefas complexas de forma eficiente.

2. **Machine Learning (ML)** é um subconjunto especializado de IA focado no desenvolvimento de algoritmos que permitem que os computadores aprendam com os dados. Em vez de serem explicitamente programados para executar uma tarefa, os algoritmos de ML identificam padrões dentro de conjuntos de dados e fazem previsões ou decisões com base nesses dados. Esta característica permite que os sistemas de ML melhorem o seu desempenho ao longo do tempo, à medida que são expostos a mais dados.

Exemplo:

- **Aplicação de IA:** Um assistente virtual como a Siri ou a Alexa usa Inteligência Artificial (IA) para compreender e responder a comandos de voz.

- **Aplicação de ML:** Um sistema de recomendação da Netflix utiliza Machine Learning - ML para sugerir filmes e séries com base no histórico de visualização do utilizador.

Para visualizar a relação:

- A IA é o conceito geral de máquinas que conseguem realizar tarefas de forma considerada "inteligente".
- A ML é uma abordagem específica para alcançar a IA, onde as máquinas recebem dados e aprendem por si mesmas.

Programação Orientada a Dados

A programação orientada a dados é um paradigma onde o comportamento do programa é controlado pelos dados, em vez de por lógica codificada de forma rígida. Esta abordagem permite criar sistemas mais flexíveis e adaptáveis. Na programação orientada a dados, os dados ditam o fluxo do programa, o que pode ser especialmente útil em aplicações como processamento, transformação e análise de dados.

Exemplos:

- **AWK** e **sed** para processamento de texto: Estas ferramentas processam dados de texto com base em padrões e regras definidos nos próprios dados.
- **XSLT** para transformar documentos XML: O XSLT usa dados em XML para definir como outros documentos XML devem ser transformados.
- **Procmail** para filtragem de emails: O Procmail utiliza regras definidas em ficheiros de configuração para filtrar e processar emails recebidos.

Conceitos Básicos de Machine Learning

O Machine learning envolve vários conceitos-chave::

1. **Aprendizagem Supervisionada:** O algoritmo é treinado com dados rotulados, ou seja, cada exemplo de treino é associado a uma saída conhecida. O objetivo é aprender uma correspondência entre entradas e saídas que permita prever rótulos para novos dados.

   Exemplo: Prever preços de casas com base em características como tamanho, localização e número de quartos. Os dados de treino incluem casas com preços conhecidos.

2. **Aprendizagem Não Supervisionada:** O algoritmo recebe dados sem instruções explícitas sobre o que fazer com eles. O objetivo é descobrir padrões ocultos ou estruturas intrínsecas nos dados.

   Exemplo: Segmentação de clientes no marketing, onde o algoritmo agrupa clientes com base no comportamento de compra, sem rótulos pré-definidos.

3. **Aprendizagem por Reforço:** O algoritmo aprende ao interagir com um ambiente, recebendo recompensas ou penalizações como feedback. O objetivo é aprender uma estratégia que maximize as recompensas acumuladas.

   Exemplo: Treinar um robô para navegar num labirinto, onde recebe recompensas por chegar ao fim e penalizações por bater nas paredes.

4. **Elementos Chave**:

   Tarefa: O problema a resolver.

Experiência: Os dados usados para aprender.

Desempenho: A medida de quão bem o algoritmo resolve a tarefa.

Examplo:

- **Tarefa:** Classificar emails como spam ou não spam.
- **Experiência:** Um conjunto de dados com emails rotulados como spam ou não spam.
- **Desempenho:** A precisão da classificação em novos emails desconhecidos.

A Interconexão entre a IA e ML

A relação entre IA e ML é fundamentalmente sinérgica. Embora todo o ML seja uma forma de IA, nem todos os sistemas de IA utilizam ML. Veja como eles interagem:

1. **Fundamentos da IA:** A IA fornece o quadro teórico para a criação de sistemas inteligentes, enquanto o ML oferece técnicas práticas para implementar esses sistemas. Por exemplo, o processamento de linguagem natural (PNL), uma área significativa da IA, depende fortemente de algoritmos de ML para processar e entender a linguagem humana de forma eficaz.

2. **Mecanismos de aprendizagem**: Os sistemas tradicionais de IA eram baseados principalmente em regras, baseando-se em regras e lógica predefinidas para tomar decisões. No entanto, estes sistemas debateram-se com a incerteza e a variabilidade dos dados. O ML introduziu a capacidade de os sistemas de IA aprenderem com a experiência, permitindo-lhes adaptar-se a novas informações e tomar decisões probabilísticas. Este avanço é particularmente importante em ambientes dinâmicos, como os veículos autónomos, que têm de navegar em condições de estrada complexas.

3. **Aplicações em Robótica**: A sinergia entre IA e ML é evidente na robótica. Os robôs inteligentes utilizam IA para capacidades cognitivas enquanto empregam algoritmos de ML para navegação, prevenção de obstáculos e execução de tarefas. Por exemplo, os robôs de armazém podem navegar autonomamente no seu ambiente usando visão computacional alimentada por técnicas de ML.

4. **Análise preditiva**: em aplicações de negócios, a análise preditiva aproveita a IA e o ML para prever resultados futuros com base em dados históricos. A IA fornece a estrutura fundamental para a criação de modelos preditivos, enquanto o ML melhora a capacidade desses modelos de aprender com dados e gerar previsões precisas.

Técnicas principais em Machine Learning

A Machine Learning engloba várias técnicas que contribuem para a sua eficácia na melhoria das aplicações de IA:

1. **Aprendizagem Supervisionada (Supervised Learning)**: Nesta abordagem, os algoritmos são treinados em conjuntos de dados rotulados onde a saída correta é conhecida. O modelo aprende a mapear entradas para saídas com base nesses dados de treino, permitindo que ele faça previsões sobre dados novos e invisíveis.

2. **Aprendizagem não supervisionada (Unsupervised Learning):** Ao contrário da aprendizagem supervisionada, a aprendizagem não supervisionada envolve algoritmos de treino em dados não rotulados. O modelo identifica padrões ou estruturas inerentes aos dados sem orientação explícita sobre o que procurar.

3. **Aprendizagem por Reforço (Reinforcement Learning)**: Esta técnica envolve treinar um agente através de interações de tentativa e erro com o seu ambiente. O agente recebe feedback na forma de recompensas ou penalidades com base em suas ações, permitindo que ele aprenda estratégias ideais ao longo do tempo.

4. **Aprendizagem profunda (Deep Learning)**: Um subconjunto de ML que utiliza redes neurais artificiais com várias camadas (redes profundas) para analisar conjuntos de dados complexos, como imagens ou sinais de áudio. A aprendizagem profunda impulsionou muitos avanços recentes em áreas como visão computacional e processamento de linguagem natural.

Aplicações do mundo real

A integração de IA e ML levou a avanços significativos em vários setores:

- **Cuidados de saúde**: as ferramentas de diagnóstico alimentadas por IA utilizam algoritmos de ML para analisar imagens médicas para deteção de doenças ou prever os resultados dos pacientes com base em dados históricos de saúde.

- **Finanças**: Em finanças, os modelos de ML são usados para deteção de fraudes, analisando padrões de transações em tempo real, fornecendo aos bancos ferramentas para mitigar o risco de forma eficaz.

- **Veículos autónomos**: Os carros autónomos utilizam uma combinação de técnicas de IA, como visão computacional e tomada de decisão, juntamente com algoritmos de ML, como deep learning ou aprendizagem profunda para interpretar dados sensoriais de câmeras e sistemas LIDAR.

- **Comércio**: As plataformas de comércio eletrónico empregam algoritmos de ML para recomendações personalizadas com base no comportamento e preferências do utilizador, melhorando a experiência do cliente.

Desafios na relação entre IA e ML

Apesar dos seus potenciais benefícios, a relação entre IA e ML também apresenta desafios:

1. **Qualidade de dados**: A eficácia dos algoritmos de ML depende fortemente da qualidade dos dados de treino. Dados de baixa qualidade ou tendenciosos podem levar a previsões imprecisas ou reforçar vieses existentes nos sistemas de IA.

2. **Interpretabilidade**: Muitos modelos avançados de ML, especialmente redes de deep learning, operam como "caixas pretas", tornando difícil para os utilizadores entenderem como as decisões são tomadas. Esta falta de transparência pode prejudicar a confiança nas aplicações de IA.

3. **Preocupações éticas**: À medida que os sistemas de IA influenciam cada vez mais áreas críticas, como contratação ou aplicação da lei, considerações éticas sobre justiça, responsabilidade e transparência tornam-se primordiais.

Conclusão

A relação entre inteligência artificial e machine learning é fundamental para entender os avanços tecnológicos modernos. Embora a machine learning sirva como uma ferramenta poderosa dentro do campo mais amplo da IA, é essencial reconhecer os papéis distintos que cada um desempenha no desenvolvimento de sistemas inteligentes capazes de transformar indústrias e melhorar vidas. À medida que ambos os campos continuam a evoluir juntos, enfrentar os desafios relacionados à qualidade dos dados, interpretabilidade e ética será crucial para aproveitar todo o seu potencial de forma responsável.

2.2 Programação baseada em dados

A importância de grandes quantidades de dados na Machine Learning

No domínio da Machine Learning (ML), os dados são muitas vezes referidos como o "combustível" que alimenta algoritmos e modelos. A eficácia e a precisão dos sistemas de ML dependem fortemente da disponibilidade de grandes quantidades de dados de alta qualidade. Esta secção enfatiza a importância dos dados em ML, introduz o conceito de programação orientada por dados e explica conceitos fundamentais, como conjuntos de dados, instâncias, atributos, variáveis de entrada e modelos.

O papel dos dados em Machine Learning

Os dados servem como base sobre a qual os modelos de ML são construídos. Sem dados suficientes, os algoritmos não podem aprender padrões ou fazer previsões de forma eficaz. A importância de grandes conjuntos de dados pode ser resumida nos seguintes pontos:

1. **Aprender com exemplos**: A ML baseia-se, essencialmente, em aprender a partir de exemplos. Grandes conjuntos de dados oferecem uma variedade diversificada de exemplos que auxiliam os algoritmos a reconhecer padrões e relações presentes na informação.

2. **Precisão aprimorada**: Modelos treinados com conjuntos de dados maiores tendem a generalizar melhor para dados nunca antes vistos, resultando em previsões mais precisas. Isto é especialmente importante em aplicações como diagnóstico na área da saúde ou previsão financeira, onde a fiabilidade das previsões é fundamental.

3. **Robustez**: Um conjunto de dados abrangente, que inclui diversos cenários, permite aos modelos lidar de forma mais eficaz com casos extremos e anomalias. Esta robustez é essencial em aplicações do mundo real, onde os dados podem ser ruidosos ou incompletos.

4. **Representação de características:** Grandes conjuntos de dados permitem extrair características relevantes que podem melhorar consideravelmente o desempenho do modelo. Com mais dados, os algoritmos conseguem aprender representações complexas que captam as tendências subjacentes.

5. **Mitigação do Overfitting:** Ter uma quantidade substancial de dados de treino ajuda a evitar o overfitting, uma situação em que o modelo aprende o ruído dos dados em vez de captar os padrões reais. Um conjunto de dados mais alargado permite uma aproximação mais fidedigna à distribuição real dos dados, promovendo uma melhor capacidade de generalização por parte do modelo.

Programação orientada por dados

Programação orientada por dados é uma abordagem que enfatiza o uso dos dados como principal motor para a tomada de decisões e desenvolvimento de algoritmos. Neste paradigma, os programadores focam-se em recolher, analisar e utilizar dados para orientar as suas práticas de codificação, em vez de depender exclusivamente de regras ou lógica predefinidas.

Por que precisamos de programação orientada por dados

1. **Adaptabilidade**: A programação baseada em dados permite que os sistemas se adaptem às condições em mudança, aprendendo continuamente com novas entradas de dados.

2. **Tomada de decisão aprimorada**: Ao utilizar grandes conjuntos de dados, os programadores conseguem desenvolver algoritmos mais informados, que conduzem a resultados mais eficazes.

3. **Automação**: As abordagens baseadas em dados facilitam a automação, permitindo que as máquinas aprendam com dados históricos e realizem previsões sem necessidade de intervenção humana.

4. **Escalabilidade**: À medida que mais dados se tornam disponíveis, os sistemas conseguem escalar os seus recursos sem necessidade de alterações significativas na arquitetura subjacente.

Conceitos Básicos de Machine Learning

1. **Conjunto de dados**

Um conjunto de dados é uma coleção estruturada de dados usada para treinar modelos de machine learning. Consiste em várias instâncias (pontos de dados) organizadas em linhas e colunas, onde cada coluna representa um atributo ou recurso.

2. **Instância**

Uma instância refere-se a um único registro ou observação dentro de um conjunto de dados. Cada instância contém valores para vários atributos que descrevem suas características.

3. **Atributo**

Os atributos são variáveis ou características individuais que descrevem uma instância num conjunto de dados. Podem ser numéricos (por exemplo, idade, salário) ou categóricos (por exemplo, sexo, cor). Os atributos são fundamentais para definir o espaço de entrada utilizado pelos algoritmos de ML.

4. **Variáveis de input (entrada)**

As variáveis de input são atributos específicos usados como preditores num modelo de ML. Fornecem as informações necessárias para que o modelo identifique padrões e consiga fazer previsões sobre as variáveis de output (também chamadas de variáveis-alvo ou metas).

5. **Variáveis de output (saída)**

As variáveis de output são os valores-alvo que o modelo pretende prever com base nas variáveis de input. Por exemplo, num modelo de previsão do preço de habitação, as variáveis de input podem incluir a área útil e a localização, enquanto a variável de output será o preço estimado da habitação.

O conceito de um modelo

Na ML, um modelo é uma representação matemática que estabelece uma relação entre variáveis de input e variáveis de output, com base em padrões aprendidos a partir dos dados de treino. O processo envolve várias etapas:

1. **Treino:** Durante o treino, o modelo aprende a partir de instâncias rotuladas no conjunto de dados, ajustando os seus parâmetros para minimizar os erros de previsão.

2. **Mapeamento:** Uma vez treinado, o modelo consegue receber novas variáveis de input e gerar previsões correspondentes com base nos padrões aprendidos.

3. **Avaliação:** O desempenho do modelo é avaliado através de métricas como **acurácia**, **precisão**, **recall** e **pontuação F1**, geralmente aplicadas a conjuntos de validação ou teste.

A importância de dispor de grandes quantidades de dados adequados na ML não pode ser subestimada. É essencial para treinar modelos eficazes, capazes de realizar previsões precisas em aplicações do mundo real.

A programação orientada por dados surge como uma metodologia fundamental, que aproveita essa abundância de dados para apoiar o desenvolvimento de algoritmos e os processos de tomada de decisão. A compreensão de conceitos fundamentais como conjuntos de dados, instâncias, atributos, variáveis de entrada, variáveis de saída e modelos estabelece as bases para uma exploração mais aprofundada das técnicas de ML e das suas aplicações em diversos domínios.

À medida que continuamos a explorar o poder dos dados na ML, é imperativo priorizar práticas de recolha e gestão de dados de qualidade, de forma a garantir resultados robustos e fiáveis em sistemas de inteligência artificial.

Vamos começar com um desafio...

Imagine um quebra-cabeças. Tente descobrir, a partir dos números apresentados na coluna da esquerda (vamos chamá-los de inputs), como se relacionam com os números da coluna da direita (os outputs). Que padrão consegue identificar?

Input	Output
1, 9, 5, 4	19
3, 8, 11	22
6, 7, 9, 2, 2	26
2, 3, 6	11

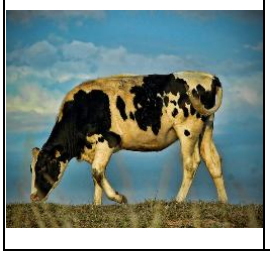
Assumimos que esta tarefa não lhe causou grande dificuldade e que, logo na segunda ou terceira linha da tabela, surgiu a ideia de que os números na segunda coluna — ou seja, a saída — representam a soma dos números que estão na coluna da esquerda, ou seja, a entrada. Para esta tarefa, também já saberá como escrever um algoritmo (mesmo que de olhos semicerrados!) que o conduza à solução.

Por exemplo, na linguagem de programação Python, podemos calcular a soma dos elementos da lista [1, 9, 5, 4] começando por declarar a soma como zero e, em seguida, adicionando um elemento de cada vez até chegarmos ao final da lista:

```
numbers = [1, 9, 5, 4]

sum = 0

for element in numbers:

  sum = sum + element
```

(A função integrada de soma calcula isso rapidamente por nós — e, na verdade, é a que usamos com mais frequência.)

Agora experimente resolver o próximo desafio: neste quebra-cabeças, terá de descobrir como as entradas e as saídas estão relacionadas. Desta vez, os inputs são imagens de animais, e os outputs são os números 0 ou 1.

Input	Output
	1
	0
	0
	1
	0

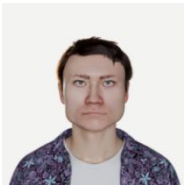
Certamente que lhe ocorreram muitas ideias — e é bem possível que todas sejam válidas! No entanto, como os uns (1) aparecem ao lado das fotos dos gatos, e os zeros (0) ao lado das restantes fotos de animais, pretendíamos que chegasse à conclusão de que esta é uma tarefa em que precisa de reconhecer se há ou não um gato na imagem.

Ou seja: Se houver um gato na imagem de entrada, o valor de saída será 1; Se não houver um gato na imagem de entrada, a saída será 0. Verá, um pouco mais adiante, que este tipo de problema se chama tarefa de classificação binária — e é extremamente comum no campo da ML.

Será possível criar um algoritmo para resolver este problema? Concordará que é importante sermos capazes de identificar o que está nas imagens, pois isso facilita a sua pesquisa e análise. Poderá tentar elaborar uma lista de dezenas de regras para determinar se há um gato na imagem ou não. Não se esqueça de considerar aspetos como o fundo, a iluminação, o ângulo de visão, bem como o facto de existirem mais de 50 tipos diferentes de gatos.

Na verdade, existem muitos problemas como este em que, mesmo que nos esforcemos (imagine nomear 1000 regras!), não conseguimos escrever algoritmos que os resolvam com precisão. Outros exemplos incluem a tradução de um idioma para outro e a deteção de rostos em fotografias. Concordará que estas tarefas também são importantes, pois nos permitem compreender conteúdos escritos numa língua que não falamos ou melhorar a segurança. É por isso que estes problemas são descritos através de conjuntos de dados — pares de entradas e saídas esperadas — e são resolvidos por técnicas de programação orientada por dados. No caso da tarefa de reconhecimento de gatos (e de qualquer outro objeto), um conjunto de dados adequado pode ser organizado de forma semelhante ao nosso exemplo, com imagens como entradas e valores 0 ou 1 como saída. Para uma tarefa de tradução, os pares podem ser frases em ambas as línguas, enquanto numa tarefa de deteção facial, podem ser pares de imagens — uma sem rostos marcados como entrada e outra com os rostos assinalados como saída. Aqui está um exemplo**.**

Input **Output**



Os conjuntos de dados que utilizamos para descrever problemas podem ser criados a partir de atividades do dia a dia. Por exemplo, após examinar um paciente no seu registo digital, um médico regista informações sobre o paciente, como idade, sexo, sintomas e alergias a medicamentos — todos estes valores constituem as entradas — e o código correspondente ao seu diagnóstico, que será a saída. De forma semelhante, nos aeroportos, para cada voo, são conhecidas informações como a hora de partida, a companhia aérea, o tipo de avião, entre outras — todas estas como inputs — e a informação sobre se o voo sofreu ou não atraso, que corresponde ao output.

Os conjuntos de dados também podem ser criados especificamente para resolver uma tarefa específica. Por exemplo, o conjunto de dados que usamos para identificar gatos poderia ter sido criado por uma equipa de voluntários que olharam para as imagens que tínhamos e seguiram nossas diretrizes, por exemplo, se houver um gato na imagem, escreva 1, escreva 0, digite 1 ou 0 na coluna de saída. Para conjuntos de dados de domínio, por exemplo, reconhecendo mudanças em raios-X, especialistas médicos precisariam ser contratados que tenham as habilidades e conhecimentos apropriados para tomar decisões. Um pouco mais tarde, aprenderá mais sobre como os conjuntos de dados são criados.

Provavelmente estará a perguntar-se: como é que aprendemos a ligação entre o input e o output num conjunto de dados? Assim como existe uma área dedicada ao desenvolvimento de algoritmos clássicos de programação e à análise das suas propriedades, também existe uma área focada no desenvolvimento de algoritmos orientados por dados e no estudo das suas características. Esta área chama-se ML (ou machine learning). A ML está no centro de todas as áreas modernas da inteligência artificial, pois está intimamente ligada aos dados e às formas de extrair conhecimento a partir deles. Na próxima secção, a ML responderá à questão que mais lhe interessa.

É importante ressalvar que existem outras áreas que lidam com dados. Entre elas, a mais antiga é certamente a estatística, um ramo da matemática que trata da recolha, descrição e análise de dados, bem como da extração de conclusões a partir desses dados. As técnicas estatísticas estão na base de muitos algoritmos de ML. A Ciência de Dados (**Data Science**) é uma disciplina que surgiu devido à incapacidade das áreas individuais responderem a muitas questões complexas. Por exemplo, todas as empresas enfrentam o desafio de como melhorar os seus serviços. Para isso, podem analisar os comentários dos utilizadores nas redes sociais ou em sites de vendas. Para processar estes comentários, é necessário recolhê-los num único local e armazená-los numa base de dados, organizá-los — por exemplo, separando comentários positivos e negativos — e, depois, analisar cada um desses conjuntos com maior detalhe, para determinar o que os utilizadores consideram negativo ou positivo, como um modelo específico de produto ou alguma funcionalidade. Esta informação deve ser partilhada com a direção da empresa, para que esta possa decidir os próximos passos. A resposta à questão inicial é, como vê, extensa e requer conhecimentos e ferramentas para descarregar conteúdo da web (os chamados scrapers), trabalhar com bases de dados, processamento de linguagem natural, assim como técnicas de visualização de dados que sejam claras e úteis para os especialistas na área. Em todos estes exemplos, a ML é uma parte indispensável do percurso para o conhecimento. Antes de prosseguirmos, vamos resumir como é a resolução de problemas com a programação clássica e a programação orientada por dados. Quando resolvemos um problema usando técnicas clássicas de programação, por exemplo, encontrando o maior elemento em uma sequência de números, primeiro pensamos no problema e nas restrições espaciais e temporais que temos, depois projetamos um algoritmo para resolvê-lo (por exemplo, merge sorting), e então o programamos em uma linguagem de programação e salvamos o código. Verificamos a precisão da implementação em algumas entradas aleatórias até nos certificarmos de que tudo funciona exatamente como esperamos.

Quando precisamos classificar uma nova string, podemos usar o programa que escrevemos, executá-lo e obter a solução apropriada. Quando dependemos da programação orientada por dados, inicialmente temos apenas dados — por exemplo, milhares de pares de entradas e saídas. Mais uma vez, é sensato pensar primeiro no problema. É isso que fazemos agora, ao conhecer o conjunto de dados. Para isso, utilizamos técnicas de análise exploratória de dados, que serão discutidas mais à frente no curso, e que nos podem dar uma ideia de que caminho seguir para encontrar uma solução. Em vez de criar um algoritmo que resolva diretamente o problema, projetamos um algoritmo que aprende a resolver o problema. Isto significa que, se precisarmos de aprender a ligação entre entrada e saída, e alterarmos o conjunto de dados, o algoritmo pode voltar a encontrar a melhor ligação entre eles. A ligação que existe entre entrada e saída depende dos dados (não é algo fixo!), e, por isso, é necessário aprendê-la — não a definimos diretamente. Quando projetamos e programamos um algoritmo deste tipo, precisamos de usar os dados para avaliar o seu funcionamento. Se os resultados não forem satisfatórios, temos de recuar e corrigir o algoritmo, ou até regressar ao início para verificar se há algo mais nos dados que possa ser importante para resolver o problema. Ao contrário da programação clássica, esta iteratividade está muito presente na programação orientada por dados.

Esta questão incorporada está configurada incorretamente

2.3 Conceitos básicos de Machine Learning

Os conceitos que vamos introduzir nesta secção são os fundamentos básicos do machine learning (ML). Ajudarão a compreender os tópicos que se seguem, e ao longo do livro aprenderá mais sobre cada um deles.

Em termos de ML, designamos o conjunto de dados que temos por **data set** (**conjunto de dados**). Pode tratar-se de registos tabulares simples, semelhantes aos que encontramos em bases de dados ou ficheiros Excel, mas também de conjuntos de imagens de satélite ou clips de áudio. Um elemento específico de um conjunto de dados é chamado de **instância**. Assim, uma linha numa tabela de registos ou uma imagem de satélite específica são exemplos de instâncias. O número de instâncias num conjunto de dados pode influenciar a escolha do algoritmo de aprendizagem, pois alguns exigem mais dados do que outros.

Nalguns casos, existem **atributos**, que são propriedades usadas para descrever os dados. Por exemplo, se imaginarmos um registo tabular de ocorrências sísmicas, a data e hora do evento, a latitude, a longitude, a intensidade sísmica, o nível de destruição e outros dados importantes podem ser considerados atributos. Os atributos também são chamados de **features** (ou **características**). Mais à frente aprenderá que tipos de atributos existem e em que devemos ter atenção. Os atributos que usamos para aprender a resolver uma tarefa são chamados variáveis de entrada (inputs), enquanto os que pretendemos prever ou ensinar são as variáveis de saída (outputs). Assim, a data e hora do sismo, as suas coordenadas geográficas e a sua magnitude podem ser variáveis de entrada na tarefa de determinar a destruição causada pelo sismo. O nível de destruição, que também está presente como atributo no conjunto de dados, será a variável de saída.

Por vezes, usaremos termos menos formais, como entrada e saída. É importante notar que é a tarefa em si que define quais os atributos que serão variáveis de entrada e quais serão variáveis de saída.

Agora, a pergunta: quais poderiam ser os atributos de uma imagem de satélite?

Resposta: *Também concorda que, no caso das imagens de satélite, podemos incluir atributos como o local, a data e a hora em que a imagem foi capturada. Podemos ainda adicionar atributos que descrevam o satélite que tirou a imagem. Contudo, nenhum destes atributos descreve diretamente o que a imagem contém. Reflita sobre este tema até chegarmos à secção que o aborda.*

Dissemos que o objetivo dos algoritmos de ML é determinar o mapeamento entre certas entradas e certas saídas. Agora podemos ser mais precisos e afirmar que o objetivo do machine learning é determinar o mapeamento entre determinadas variáveis de entrada e variáveis de saída. Estes mapeamentos são designados por modelos.

O conceito que associamos ao mapeamento é o de função. Nas aulas de matemática, já deve ter ouvido falar frequentemente sobre funções como mapeamentos de entradas para saídas. Por exemplo, a função de uma variável $y = 2x + 4$ mapeia o input $x = 5$ para o valor $y = 14$, enquanto a função de múltiplas variáveis $y = 2x_1 - 3x_2 + x_3 + 5$ mapeia o input $(x_1, x_2, x_3) = (1, -1, 3)$ para o valor $y = 13$. As variáveis que aparecem nas funções estão relacionadas aos valores do atributo. Assim, x na primeira função pode representar a metragem quadrada da propriedade, enquanto x_1, x_2, x_3 na segunda função pode representar os valores de atributos como latitude, longitude e força do terremoto. Na aula de matemática, já ouviu falar que existem diferentes classes de funções (linear, polinomial, trigonométrica, exponencial, logarítmica), e que cada uma delas é caraterizada por algumas propriedades especiais, como continuidade, monotonia ou convexidade. Todo este conhecimento é bem-vindo quando se procura o modelo certo.

A complexidade de uma função é algo que não iremos introduzir formalmente. Irá perceber que algumas funções são mais simples do que outras, ou, por assim dizer, são "mais diretas". Funções simples são mais gratificantes para trabalhar e mais fáceis de compreender, mas limitam a nossa capacidade de descrever algumas das relações mais complexas entre os atributos e os resultados. Por outro lado, funções mais complexas são assim por uma razão, mas pode ser difícil acompanhar alguns dos seus comportamentos matemáticos, o que pode influenciar o processo de aprendizagem. Tentamos encontrar um equilíbrio entre a complexidade do modelo, o que sabemos sobre os dados e aquilo que pretendemos aprender.

Em modelos, como vimos no exemplo introdutório de preço imobiliário de imóveis, **parâmetros** como k e n podem aparecer. Tais modelos são chamados de **modelos paramétricos** e a tarefa de determinar o modelo certo é reduzida à tarefa de determinar os melhores valores dos parâmetros. No modelo linear, apenas dois parâmetros apareceram na tarefa de precificação de imóveis, enquanto os modelos modernos, aqueles que são baseados em redes neurais têm milhões ou bilhões de parâmetros. Veremos que existem também **modelos não paramétricos ligeiramente diferentes**, cujas formas são expressas de forma diferente.

O processo de encontrar o modelo é denominado **treino do modelo**. Quando o modelo contém parâmetros desconhecidos, o objetivo do treino é precisamente determinar os seus valores. Esse é o nosso principal objetivo nesta fase.

Nos conjuntos de dados utilizados para treinar os modelos, é comum encontrarmos valores imprecisos ou contraditórios. Por esse motivo, os modelos nunca são absolutamente perfeitos ou 100% corretos. Isso leva-nos a outro conceito essencial na teoria do machine learning: a função de erro, ou **função de perda**. Esta função indica-nos quão errado está o modelo. Durante o processo de treino, utilizamos ativamente os seus valores, procurando ajustar o modelo de forma a minimizar o erro. No caso dos modelos paramétricos, como o da precificação de imóveis, o objetivo é encontrar os valores dos parâmetros que levam ao menor valor da função de erro.

Uma vez treinado o modelo, é fundamental avaliar quão bom ele é na prática. Para isso, recorremos às chamadas **medidas de qualidade** — cada uma adaptada ao tipo de tarefa e ao domínio específico onde o modelo será aplicado. É importante salientar que, em geral, a função de erro e as medidas de qualidade não são a mesma coisa. Ambas avaliam a performance do modelo, mas em momentos diferentes:

- A função de erro é utilizada durante o treino,
- As medidas de qualidade são usadas após o treino, para avaliação externa.

A função de erro está intimamente ligada à estrutura matemática do modelo, enquanto as medidas de qualidade são pensadas para serem intuitivas e compreensíveis por utilizadores e especialistas do domínio. Se os resultados obtidos através dessas medidas forem insatisfatórios, o modelo deve ser revisto. Mais adiante, veremos o que isto significa e como o podemos fazer. Todo este processo de avaliação é conhecido como **teste do modelo**.

De forma geral, os valores produzidos por um modelo treinado são designados por previsões. Assim, o preço estimado de um novo imóvel ou a avaliação da devastação de um sismo são exemplos de previsões feitas por modelos. É por isso que falamos frequentemente em previsões no contexto da inteligência artificial. Mas fica claro que essas previsões não são aleatórias — são baseadas em dados e modelos bem definidos. A aplicação do modelo para gerar essas previsões é chamada de **inferência**.

Todos os termos destacados neste texto são conceitos-chave em machine learning e surgem com frequência na literatura e nas aplicações práticas. Por isso, é fundamental que fiquem bem compreendidos, tanto no seu significado como no papel que desempenham no desenvolvimento de modelos de IA.

2.4 O processo de Machine Learning

O processo de ML (Machine Learning) é uma abordagem sistemática utilizada para desenvolver algoritmos que permitem aos computadores aprender a partir de dados e realizar previsões ou tomar decisões sem terem sido explicitamente programados para tal. Este processo envolve várias etapas fundamentais, cada uma desempenhando um papel essencial na construção de modelos eficazes de ML. Compreender estas etapas ajuda profissionais e estudantes a lidar com a complexidade da ML e aumenta significativamente a probabilidade de sucesso na resolução de problemas através desta tecnologia.

1. Recolha de dados

O primeiro passo no processo de ML é a recolha de dados, que consiste em reunir dados relevantes que serão utilizados para treinar o modelo. A qualidade e a quantidade dos dados recolhidos influenciam diretamente o desempenho do modelo.

Os dados podem ser obtidos a partir de várias fontes, nomeadamente:

- **Conjuntos de dados públicos**: repositórios como o **Kaggle**, o **UCI Machine Learning Repository** e bases de dados governamentais disponibilizam conjuntos de dados previamente recolhidos para diversas aplicações.
- **Web scraping**: ferramentas automatizadas podem extrair dados de sites, recolhendo informações que não estão imediatamente disponíveis em formatos estruturados.
- **Questionários e experiências**: a recolha personalizada de dados através de **inquéritos** ou **experiências controladas** pode gerar conjuntos de dados específicos e adaptados a um determinado problema.

O resultado desta etapa é um **conjunto de dados coerente** que representa o domínio do problema e que servirá de base para as etapas seguintes do processo de ML.

2. Preparação dos dados

Uma vez recolhidos, os dados devem ser preparados para análise. Esta fase de preparação envolve várias tarefas críticas:

- **Limpeza de dados**: os dados brutos geralmente contêm erros, duplicados ou valores ausentes. A limpeza dos dados é essencial para garantir precisão e confiabilidade. As técnicas incluem a remoção de duplicados, a correção de inconsistências e o preenchimento ou remoção de valores em falta.
- **Transformação de dados**: esta etapa pode envolver a normalização ou padronização de dados para garantir que todos os recursos contribuam igualmente para o desempenho do modelo. Os tipos de dados também podem precisar de conversão (por exemplo, conversão de variáveis categóricas em formatos numéricos).
- **Divisão de dados**: o conjunto de dados é normalmente dividido em conjuntos de treino e teste. Uma proporção de divisão comum é de 80% para treino e 20% para testes, garantindo que o modelo possa ser avaliado em dados não vistos.

A preparação de dados é muitas vezes um dos aspetos mais demorados do processo de ML, mas é crucial para o desenvolvimento de um modelo eficaz.

3. Seleção de recursos e engenharia

A seleção e a engenharia de atributos (também conhecidos como *features*) são etapas fundamentais que determinam quais características dos dados serão utilizadas no treino do modelo:

- **Seleção de atributos**: consiste na identificação das características mais relevantes que contribuem para a previsão da variável-alvo. Técnicas como a análise de correlação, a eliminação recursiva de atributos ou o uso de algoritmos como o LASSO podem ajudar a selecionar os atributos mais importantes e a descartar os irrelevantes.
- **Engenharia de atributos**: nesta fase, podem ser criados novos atributos a partir dos existentes, com o objetivo de melhorar o desempenho do modelo. Isto pode incluir atributos polinomiais, termos de interação ou atributos agregados com base no conhecimento do domínio do problema.

A seleção e engenharia eficazes de recursos podem melhorar significativamente o poder preditivo de um modelo, garantindo que ele se concentre nos aspetos mais informativos dos dados.

4. Seleção do modelo

Com os dados preparados e os atributos selecionados, o passo seguinte consiste na escolha de um algoritmo de ML adequado. A escolha do algoritmo depende de vários fatores:

- **Tipo de problema**: diferentes algoritmos são mais apropriados para diferentes tipos de tarefas —
    - *Classificação* (por exemplo, **regressão logística**, **árvores de decisão**),
    - *Regressão* (por exemplo, **regressão linear**),
    - *Agrupamento* (*clustering*, por exemplo, **k-means**),
    - ou *aprendizagem por reforço*.
- **Características dos dados**: a natureza dos dados (por exemplo, **dimensão**, **volume**, **presença de ruído**) influencia fortemente a escolha do algoritmo. Por exemplo, modelos de **aprendizagem profunda** (*deep learning*) são frequentemente preferidos em conjuntos de dados grandes com padrões complexos e não lineares.

A seleção de um modelo apropriado constitui a base para um treino eficaz e uma avaliação rigorosa.

5. Modelo de Formação

O treino do modelo consiste em alimentar os dados preparados ao algoritmo selecionado, de modo a permitir que este aprenda os padrões existentes no conjunto de dados:

- **Processo de treino**: Durante esta fase, o algoritmo ajusta os seus **parâmetros internos** com base nas **variáveis de entrada** e nas **variáveis de saída** correspondentes (também chamadas de *variáveis alvo*). Este processo é **iterativo** e continua até que seja atingido um critério de paragem, como por exemplo um número definido de *épocas* ou a **convergência** do modelo.
- **Métricas de avaliação**: É essencial definir previamente as **métricas de sucesso** que serão utilizadas para avaliar o desempenho do modelo. Entre as métricas mais comuns destacam-se:
    - a **precisão** (*accuracy*) para tarefas de classificação,
    - o **erro quadrático médio** (*mean squared error*) para tarefas de regressão,
    - e a **pontuação F1** (*F1-score*) para conjuntos de dados **desequilibrados**.

O objetivo do treino é desenvolver um modelo que consiga generalizar bem para dados novos, e não apenas memorizar os exemplos presentes no conjunto de treino (um problema conhecido como **overfitting**).

6. Avaliação do modelo

Após o treino, avaliar o desempenho do modelo em dados de teste invisíveis é crucial:

- **Teste**: O modelo é avaliado usando o conjunto de dados de teste que não esteve envolvido no treino. Essa avaliação fornece insights sobre o quão bem o modelo se generaliza para novas instâncias.
- **Métricas de desempenho**: Dependendo do tipo de problema, várias métricas podem ser empregadas:
    - Para tarefas de classificação: precisão, precisão, recordação, pontuação F1.

- Para tarefas de regressão: R-quadrado, erro absoluto médio (MAE), erro quadrado médio (MSE).

Esta avaliação ajuda a identificar áreas onde o modelo se destaca ou precisa de melhorias.

7. Ajuste de hiperparâmetros

O ajuste de hiperparâmetros consiste na optimização de parâmetros que controlam o processo de treino, mas que não são aprendidos directamente a partir dos dados:

- **Pesquisa em grelha** (*grid search*) e **pesquisa aleatória** (*random search*): Estas técnicas exploram de forma sistemática ou aleatória diferentes combinações de hiperparâmetros, com o objectivo de encontrar as configurações que maximizam o desempenho do modelo.
- **Validação cruzada**: A utilização da **validação cruzada** durante o ajuste de hiperparâmetros permite avaliar a estabilidade e a fiabilidade do modelo, garantindo que as melhorias observadas não dependem apenas de um subconjunto específico de dados.

Um bom ajuste fino dos hiperparâmetros pode conduzir a melhorias significativas na precisão, generalização e robustez do modelo.

8. Implantação

Uma vez que um modelo satisfatório tenha sido treinado e avaliado, pode ser colocado em produção:

- **Integração**: O modelo final deve ser integrado nos sistemas existentes, onde passará a fornecer previsões ou informações com base em dados em tempo real.
- **Monitorização**: A monitorização contínua após a implantação é essencial para garantir que o modelo mantenha um bom desempenho à medida que novos dados se tornam disponíveis e as condições mudam.

A implantação marca a transição do desenvolvimento para a aplicação prática, destacando a sua utilidade no mundo real.

Conclusão

O processo de ML engloba um conjunto de etapas estruturadas que orientam os profissionais desde a recolha inicial de dados até à implantação de modelos preditivos. Cada etapa — recolha e preparação dos dados, seleção e engenharia de atributos, escolha e treino de modelos, avaliação, ajuste de hiperparâmetros e implantação — desempenha um papel crucial no desenvolvimento de soluções eficazes baseadas em dados. Ao compreender este processo de forma integrada, os profissionais e estudantes aumentam significativamente a sua capacidade de desenvolver modelos robustos que fornecem informações úteis e acionáveis em múltiplos domínios no mundo atual orientado por dados.

2.5 Tipos de Machine Learning

Nesta secção, iremos apresentar os tipos básicos de machine learning: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem por reforço. Todos estes tipos estão por detrás de tarefas como prever a precipitação, recomendar filmes para ver ou jogar. Cada uma destas áreas será abordada com mais detalhe mais à frente no curso. Também falaremos sobre algumas áreas de investigação atuais, como a aprendizagem por transferência de conhecimento.

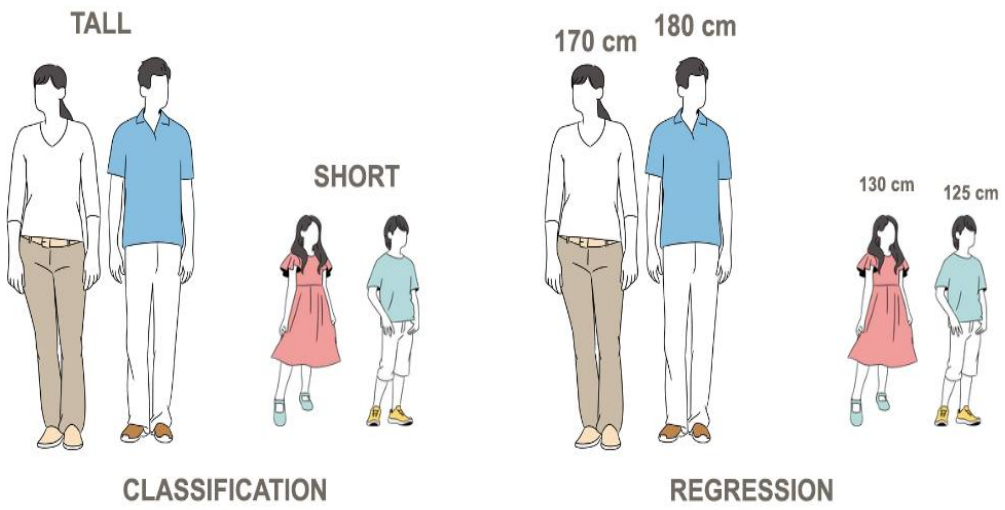
Machine Learning Supervisionada

A maioria dos exemplos que vimos até agora são, na realidade, exemplos de **ML supervisionada**. A aprendizagem supervisionada envolve algoritmos que se alinham perfeitamente com o que temos vindo a discutir: ajudam-nos a aprender a associar um conjunto de valores a outro. Por isso, é necessário que, no conjunto de dados onde aplicamos estes algoritmos, além dos valores dos atributos, conheçamos também os valores da variável alvo.

As duas tarefas principais da aprendizagem supervisionada são a **regressão** e a **classificação**. Em ambas as tarefas queremos aprender a prever valores, mas nas tarefas de regressão esses valores podem ser arbitrários, enquanto na classificação são escolhidos a partir de um conjunto finito e predefinido de valores.

Por exemplo, tarefas de regressão são adequadas para prever a temperatura, o preço de um produto (como na tarefa de determinar o preço dos imóveis que vimos na introdução), a devastação provocada por um sismo, entre outros. Por outro lado, decidir se um e-mail é indesejado ou desejado, ou determinar o género de um filme, são exemplos de tarefas de classificação, porque o conjunto de valores possíveis é limitado — o e-mail pode ser "indesejado" ou "desejado" (dois valores), e um género pode ser, por exemplo, comédia, drama, ação ou thriller (quatro valores).

Mais adiante, iremos apresentar definições mais precisas para cada uma destas tarefas.

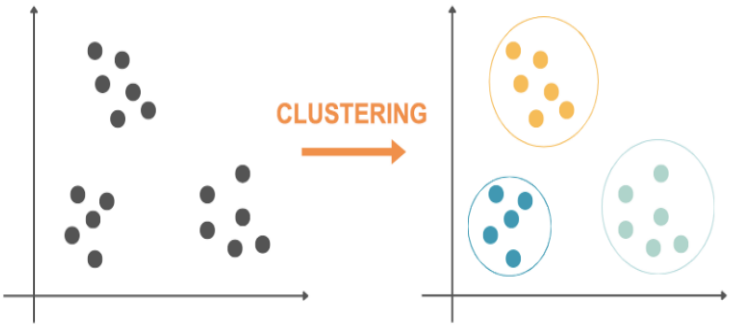


Classificação e regressão. Determinar se alguém é alto ou baixo é a tarefa da classificação. Determinar a altura exata é a tarefa da regressão.

ML Não Supervisionada

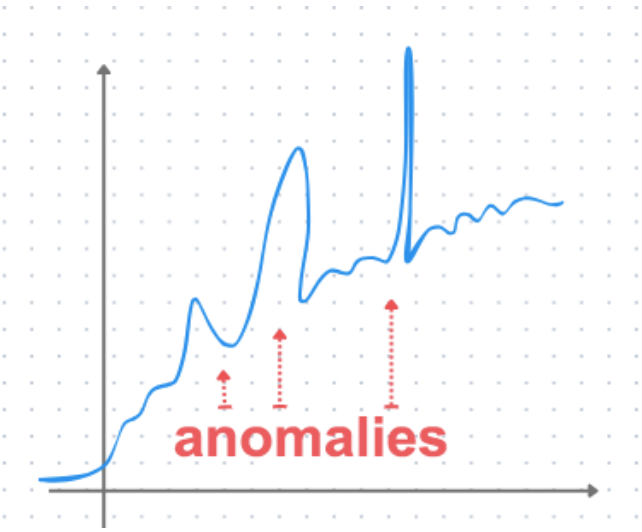
Utilizamos ML não supervisionada em tarefas que requerem a análise da estrutura de um conjunto de dados. Por exemplo, ao analisar as compras dos clientes numa loja, pode ser interessante identificar produtos que são frequentemente comprados em conjunto, para assim organizá-los melhor no espaço da loja, melhorar a oferta e aumentar os lucros. De forma semelhante, os comentários dos utilizadores podem ser analisados e agrupados, permitindo extrair informações sobre os serviços ou funcionalidades de que falam os clientes. Tarefas deste género, em que queremos identificar grupos ou padrões dentro dos dados, são chamadas de **clustering** (agrupamento).

Mais adiante no curso, aprenderá sobre o algoritmo k-means, um dos algoritmos de clustering mais conhecidos.



Agrupamento

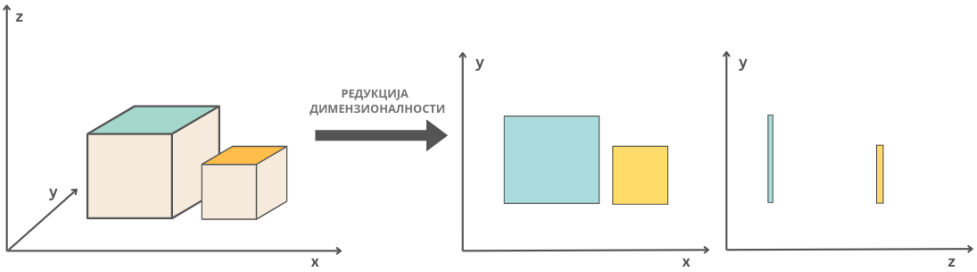
Detectar instâncias de dados que são, de alguma forma, diferentes das restantes também faz parte das tarefas de ML não supervisionada. Por exemplo, detetar medições atípicas dos sensores numa fábrica pode ser um sinal para iniciar procedimentos de segurança adicionais. Da mesma forma, identificar transações bancárias invulgares, como aquelas realizadas a partir de locais remotos ou com valores incomuns, pode indicar possíveis casos de fraude. Esta tarefa de ML não supervisionada é conhecida como **detecção de anomalias**.



Detecção de anomalias

A ML não supervisionada também abrange tarefas de redução da dimensionalidade. Muitas vezes, para efeitos de representação gráfica dos dados, é necessário reduzir o número de atributos para valores menores, como dois ou três. Claro que, durante esta transformação, alguma informação do conjunto de dados original se perde, mas, por outro lado, ganha-se a capacidade de visualizar os dados e, por vezes, de perceber melhor algumas regularidades.

Ter uma menor dimensionalidade dos dados (menos atributos) é desejável, pois permite uma execução mais rápida dos algoritmos e reduz a complexidade de memória — o que pode ser especialmente importante quando dispomos de recursos limitados para trabalhar. Alguns dos algoritmos mais utilizados para a redução da dimensionalidade são a Análise de Componentes Principais (ACP) e o t-SNE.



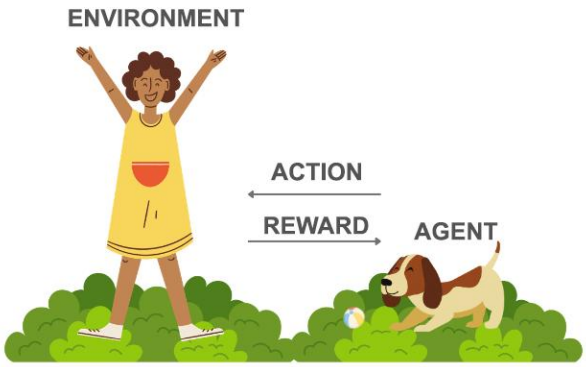
O Significado da Redução da Dimensionalidade: Dois Cuboides e Suas Projeções do Espaço Tridimensional ao Bidimensional

Curiosamente, em tarefas de machine learning não supervisionadas, não é necessário conhecer os valores da variável alvo. O agrupamento, a detecção de anomalias e a redução da dimensionalidade são realizados apenas com base nos valores dos atributos.

Aprendizagem por reforço

Certamente já viu várias vezes como se treina um cão. Quando lhe é dada uma tarefa, por exemplo, trazer uma bola do outro lado do quintal, a recompensa na forma de um biscoito quando ele a traz motiva o cão a realizar a tarefa com ainda mais sucesso e entusiasmo na próxima vez. Esta ideia está também na base da aprendizagem por reforço.

A aprendizagem por reforço é uma área da ML utilizada em tarefas como jogar ou conduzir de forma autónoma. Carateriza-se pela existência de um ambiente que possui estados próprios, um agente que pode executar um determinado conjunto de ações e o conceito de recompensa. O objetivo é que o agente, num determinado ambiente cujos estados vão mudando, escolha (aprenda) a sequência de ações que lhe permita obter a maior recompensa possível. No contexto do exemplo introdutório, o estaleiro é o ambiente. Os seus estados podem ser, por exemplo, uma bola no fim do quintal ou o gato do vizinho numa árvore. O cão é o agente, e o conjunto de ações que pode executar inclui correr, sentar-se ou dormir. A recompensa pode ser uma série de biscoitos ou nada. Se o cão escolher a sequência certa de ações (correr, encontrar a bola e trazê-la) perante uma mudança no ambiente, como a aparição da bola, será capaz de ganhar a maior recompensa.



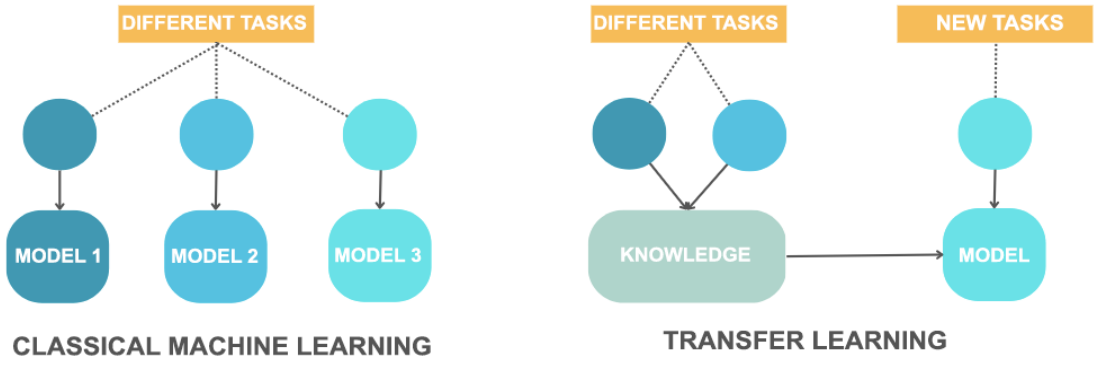
Aprenderá mais sobre este tipo de aprendizagem no final do curso.

Novos rumos na aprendizagem

Quando precisamos dominar uma nova tarefa, por exemplo, aprender a andar de trotinete, não começamos do zero. Todos os conhecimentos e competências que adquirimos noutras tarefas, como jogar basquetebol,

andar de bicicleta, e até a perseverança e paciência em tarefas que nem sempre eram as nossas favoritas — como arrumar o sótão — ajudam-nos a dominar melhor a nova tarefa. Esta ideia está na base da aprendizagem por transferência. Por isso, muitas vezes, pode ouvir falar de modelos que foram usados como ponto de partida para o desenvolvimento de outros modelos. Estes modelos são primeiramente treinados em conjuntos de dados e tarefas gerais e, depois, retreinados para resolver tarefas específicas.

Por exemplo, o modelo de linguagem GPT foi usado como base para desenvolver o ChatGPT, que já tinha demonstrado bom desempenho em tarefas como geração de resumos, versões reduzidas de texto e respostas a perguntas.



A ideia de aprendizagem através da transferência de conhecimento

As técnicas de transferência de conhecimento podem ser combinadas com todos os tipos de aprendizagem mencionados anteriormente. São especialmente importantes quando os conjuntos de dados de treino para uma tarefa específica não são suficientemente grandes, ou quando estamos a desenvolver um modelo para um domínio específico.

2.6 Dados em Machine Learning

Na comunidade de IA, é comum ouvir dois ditados: "os dados são o novo ouro" e "entra lixo, sai lixo". Estes lembram-nos do valor dos dados para a compreensão e modelação de fenómenos, assim como da importância de criar conjuntos de dados de alta qualidade. Vamos explorar estes tópicos.

Hoje em dia, quase todos os domínios de atividade geram grandes quantidades de dados: informações sobre vídeos que vemos online, produtos que compramos, amigos com quem nos conectamos nas redes sociais, bem como informações sobre consultas médicas, condições climáticas na nossa cidade ou condições de trânsito registadas por instituições relevantes. Todos estes dados podem ser usados para entender melhor o ambiente onde são gerados.

Tal como na história dos bancos de dados que estudou no ano passado, no machine learning descrevemos entidades e eventos importantes, cujos comportamentos queremos modelar, usando atributos (também chamados de características). Por exemplo, um filme pode ser descrito pelo título, género, ano de lançamento, produtora, orçamento, lucro, sinopse, nome do realizador e nomes dos atores principais. Escolher os atributos certos para registar durante a recolha de dados não é tarefa fácil, porque não sabemos antecipadamente quais atributos serão mais úteis para a tarefa que queremos resolver no futuro. Por exemplo, se quisermos usar dados para prever o lucro de um filme (uma tarefa de regressão), informações sobre os atores e a produtora podem ser mais úteis, enquanto para determinar o género do filme (uma

tarefa de classificação), a sinopse pode ser mais importante. Em domínios mais complexos, estas escolhas trazem ainda mais dilemas e desafios.

Devido à necessidade de usar dados para uma vasta gama de aplicações, podemos pensar em recolher o maior número possível de atributos. Embora esta ideia seja válida em algumas situações, normalmente devemos lembrar que grandes volumes de dados exigem armazenamento adequado, hardware capaz de suportar o seu processamento e uma equipa de especialistas com as competências e conhecimentos necessários para realizar estas tarefas. Portanto, estas escolhas podem ser dispendiosas e exigir planeamento cuidadoso. Também é importante notar que analisar e interpretar grandes quantidades de dados é um desafio e requer competências técnicas específicas, como técnicas de visualização de dados. Além disso, muitos domínios que envolvem dados privados e sensíveis devem cumprir regulamentos e diretrizes éticas rigorosas sobre a recolha e utilização de dados, o que impõe restrições adicionais na seleção de atributos e no armazenamento dos dados. Assim, a tarefa de recolher dados e criar conjuntos de dados de alta qualidade é exigente e requer organização cuidadosa.

Nas próximas lições, veremos que cada atributo é definido pelo seu tipo e conjunto de valores, e estas propriedades influenciam a forma como preparamos os dados. Em última análise, os algoritmos de ML (Machine Learning) só podem ser aplicados a valores numéricos. O número de atributos e as suas propriedades também influenciam a escolha do algoritmo de ML.

Algoritmos avançados de ML, como as redes neurais, conseguem identificar por si próprios os atributos importantes para resolver uma tarefa. Isto alivia-nos da necessidade de pensar tanto na seleção e combinação dos atributos. Este aspeto é particularmente útil quando trabalhamos com dados complexos, como imagens ou texto, em que definir e extrair atributos nem sempre é intuitivo. Estes algoritmos conseguem trabalhar diretamente com dados brutos.

- O que considera desafiante na recolha de dados no domínio que lhe interessa? Pode ser no desporto, numa disciplina científica, num fenómeno social ou qualquer outro campo.
- Tem alguma preocupação ou reserva relativamente à recolha e ao tratamento dos dados?
- O que é mais importante para si pessoalmente no processo de recolha de dados?

Conjuntos de dados populares

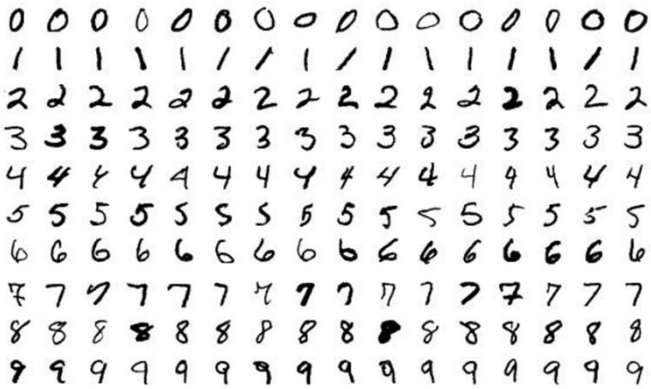
Pode parecer surpreendente, mas os conjuntos de dados também podem ser populares! Alguns são conhecidos por terem sido usados nas primeiras tarefas de ML, enquanto outros ganharam popularidade devido ao envolvimento contínuo da comunidade, que os expandiu e complementou.

Como diferentes conjuntos de dados representam diferentes domínios da inteligência artificial, vamos agrupá-los consoante o tipo de dados que contêm: imagens, texto, áudio e vídeo. Muitas bibliotecas usadas em ML facilitam o carregamento rápido e simples destes conjuntos de dados.

Visão computacional

MNIST

Um dos conjuntos de dados mais populares na área da visão computacional é o MNIST, um conjunto de imagens de dígitos manuscritos. O seu desenvolvimento começou em 1998 pelo Instituto Nacional de Normas e Tecnologia dos EUA (National Institute of Standards and Technology, NIST). Todas as imagens têm 28x28 pixels, são a preto e branco, e o conjunto total conta com 70.000 imagens: 60.000 para treino e 10.000 para teste. Na imagem pode ver alguns dos dígitos deste conjunto de dados.



Alguns dos números da conferência MNIST

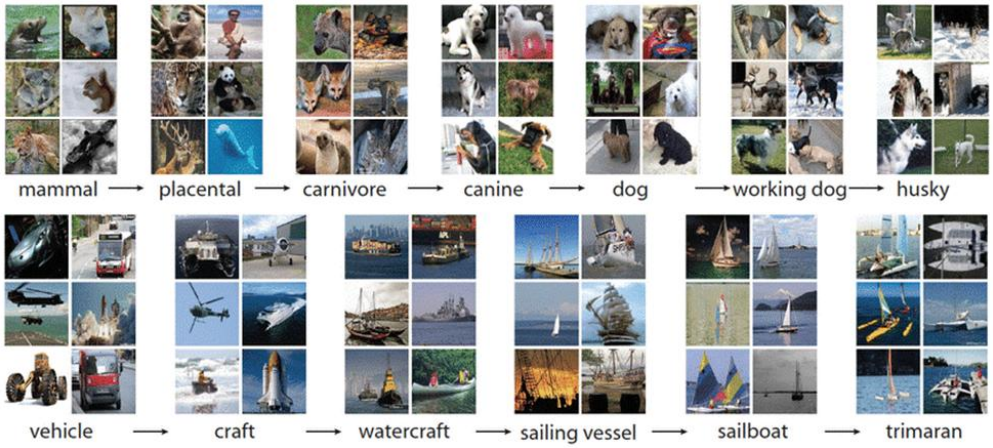
O conjunto MNIST é utilizado para treinar classificadores multiclasse, frequentemente em combinação com redes neurais convolucionais, sobre as quais irá ouvir mais adiante no curso.

Para cada dígito do conjunto MNIST, existe uma classe atribuída. Pense nos dígitos que podem ser potencialmente difíceis de distinguir (por exemplo, os dígitos 1 e 7 podem parecer-se bastante) e, em seguida, tente encontrar alguns exemplos na internet.

ImageNet

As imagens do conjunto **ImageNet** representam objetos variados, como computadores, janelas, aviões, plantas, animais tropicais e muitas outras categorias. Curiosamente, estas imagens estão organizadas em grupos relacionados, chamados *synsets*, que têm uma estrutura hierárquica do tipo pai-filho. Por exemplo, todos os veleiros pertencem a um *synset*; na hierarquia abaixo deste existem grupos como planadores e trimarãs, enquanto acima há categorias mais gerais como embarcações e veículos.

Na imagem, na linha inferior, pode observar essa hierarquia: na base encontram-se os trimarãs e no topo a categoria "veículo". Na linha superior, pode ver *synsets* relacionados com cães e algumas das suas classificações.



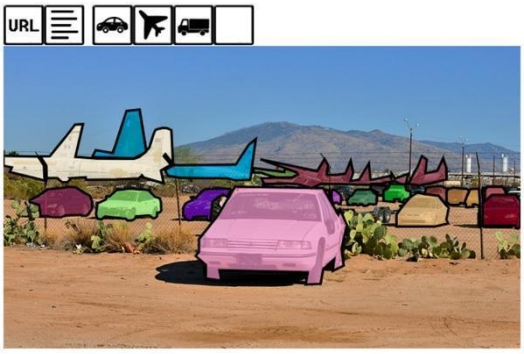
Exemplo de uma imagem do ImageNet

A coleção contém atualmente cerca de 14 milhões de imagens e mais de 21.000 sinsets. Ele é usado em uma variedade de tarefas de classificação de imagem e deteção de objetos em imagens.

O site oficial da conferência ImageNet é https://www.image-net.org/index.php. Pesquisadores das universidades de Stanford e Princeton estão trabalhando ativamente no seu desenvolvimento. Tente descobrir a qual grupo um computador pertence no conjunto ImageNet e quais grupos estão na hierarquia abaixo e acima

COCO

O conjunto de dados **COCO** (acrônimo para *Common Objects in Context*) é usado em tarefas de deteção de objetos, segmentação de imagens e associação automática de títulos a imagens. Foi criado pela Microsoft e partilhado com a comunidade em 2015.



Uma imagem do conjunto COCO com objetos reconhecidos marcados: aviões, caminhões e carros

O conjunto pode ser visto interativamente no site oficial: para cada imagem há um URL a partir do qual a imagem foi tirada, vários títulos associados à imagem e, em seguida, uma série de ícones correspondentes aos objetos reconhecidos. O número de imagens no conjunto de dados é de 330.000 e contém 80 categorias de objetos com mais de 1,5 milhão de instâncias. O link para a secção de pesquisa no site é https://cocodataset.org/#explore .

Processamento de Linguagem Natural

IMDB

Se gosta de assistir filmes e programas de TV, estará interessado no **conjunto de dados IMDB** , que contém comentários de usuários da popular plataforma IMDB. Para cada ponto de vista neste conjunto de dados, sabe-se também se é positivo ou negativo, ou seja, se contém principalmente algo louvável e bom sobre o filme ou alguma crítica e objeção. Quando se trata de conjuntos de dados que contêm conteúdo textual, é sempre importante enfatizar em qual idioma eles são escritos. O conjunto de dados do IMDB contém visualizações em inglês com um total de 50.000 visualizações, 25.000 visualizações positivas e 25.000 negativas. Abaixo pode ver um input positivo e um negativo neste conjunto de dados.

Review	Sentiment 0-negative 1-positive
Don't even ask me why I watched this! The only excuse I can come up with that I was sick with Bronchitis and too weak to change the channel. :) It's too terrible for words, the movie that is, not the Bronchitis.	0
this movie is the best movie ever it has a lot of live action It's just great everyone should watch it and the actor are great the location is Rome Italy thats the best place ever the actors are great	1

Exemplos de avaliações positivas e negativas do conjunto IMDB

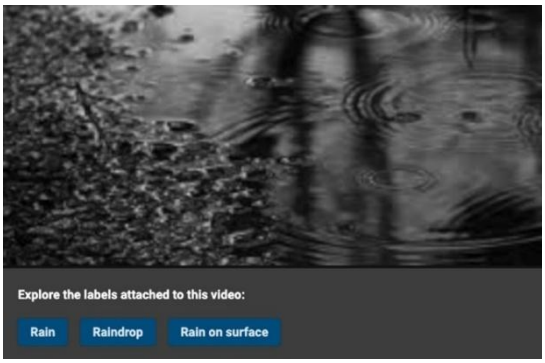
O conjunto de dados IMDB é usado em tarefas de análise de sentimento - lembre-se que estas são tarefas em que é necessário reconhecer uma emoção ou atitude presente no texto. Uma vez que o conjunto contém apenas informações se a revisão é positiva ou negativa, a tarefa de análise de sentimento no conjunto IMDB é abordada como um problema de classificação binária. Em geral, a escala de sentimento pode ser mais fina e inclui classificações como muito positivas, positivas, neutras, negativas ou muito negativas.

Sound processing

Processamento de som

Conjunto de áudio

Um AudioSet é um conjunto de dados que contém trechos de 10 segundos de vídeo do YouTube. Cada um desses trechos está associado às características dos sons ouvidos neles. O conjunto foi criado pelo Google e contém mais de 2 milhões de clipes com duração total de 5,8 mil horas.



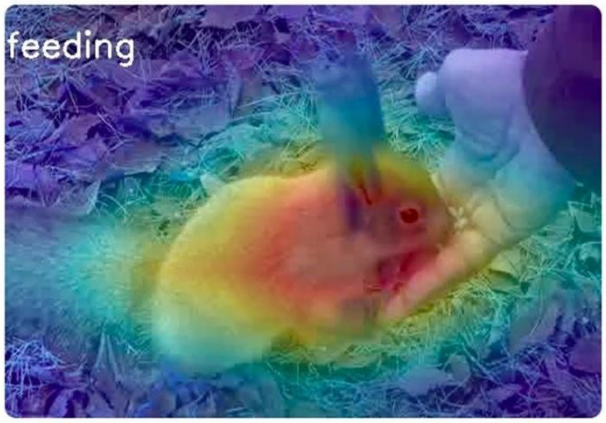
Um exemplo de um clipe de vídeo com as anotações de áudio associadas que contém

O site oficial da conferência fornece uma visão geral de exemplos e informações sobre a organização da conferência. São utilizadas 632 categorias diferentes, tais como os sons dos instrumentos musicais, o som do vento, o som do homem, o ruído, etc. Pode visitar a morada https://research.google.com/audioset/index.html e ouvir mais alguns exemplos. A conferência em si foi criada com a ideia de apoiar o desenvolvimento de algoritmos de reconhecimento de som.

Momentos no Tempo

Moments in Time é um conjunto de dados que está a ser desenvolvido com a ideia de ajudar os sistemas de inteligência artificial a aprender a reconhecer ações e eventos. Este conjunto contém atualmente um milhão de vídeos de 3 segundos de duração em que as atividades são marcadas. Os vídeos contêm pessoas, animais, objetos e fenômenos naturais. Apenas alguns dos eventos que são cobertos são dança, exercício, subir em uma árvore, pular na água e dormir.

O encontro Moments in Time está a ser desenvolvido por uma equipa do Massachusetts Institute of Technology (MIT) e no site oficial do projeto pode ver mais alguns exemplos de vídeos e ações reconhecidas. O link para o site oficial é http://moments.csail.mit.edu/ .

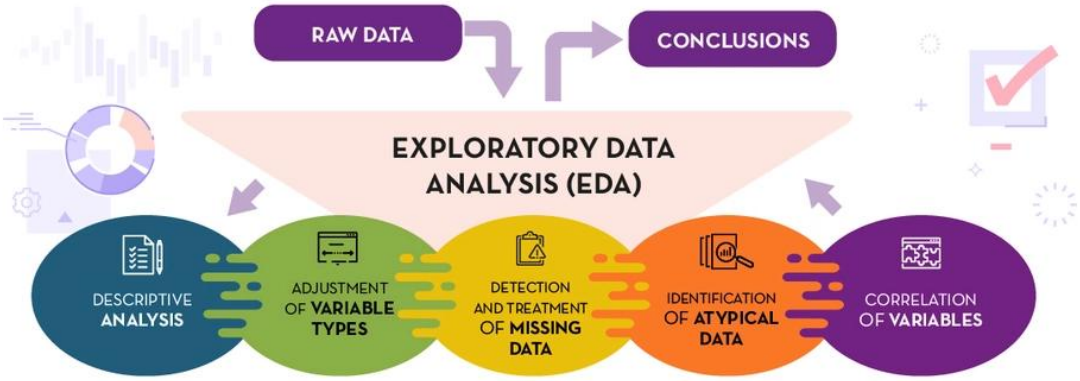


Um vídeo em que se reconhece que um homem alimenta um coelho

2.7 Análise exploratória de dados (EDA)

Encontrar um novo conjunto de dados é como uma viagem a um lugar desconhecido. É preciso explorá-lo com atenção, descobrir onde está cada coisa e quais as ligações entre as diferentes partes. Nesta secção, aprenderá algumas técnicas que o ajudarão nesta aventura com dados.

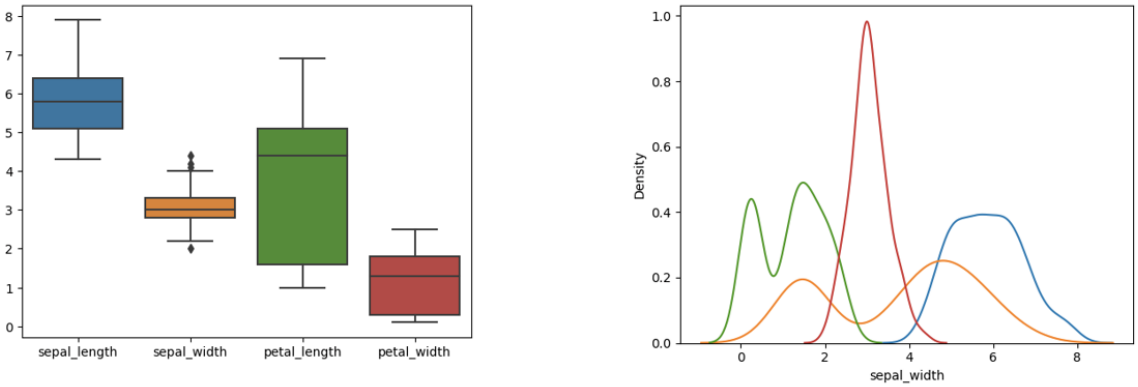
Começamos cada tarefa de ML por conhecer o conjunto de dados. Se utilizarmos dados tabulares, interessa-nos quais os atributos presentes, que valores possuem e se alguns poderão estar relacionados. Quando trabalhamos com outros tipos de dados, como texto, geralmente queremos saber se todos os textos estão escritos na mesma língua e qual a sua extensão. Como nenhum conjunto de dados é perfeito, na análise tentamos identificar possíveis duplicados e algumas entradas atípicas. Todas estas tarefas designam-se por análise exploratória de conjuntos de dados — Análise Exploratória de Dados (AED). O seu objetivo é ajudar-nos, através de um conjunto diversificado de tarefas, a conhecer melhor o conjunto de dados e a tomar decisões mais informadas para a preparação dos dados. Dada a importância dos dados nas etapas seguintes (lembre-se do ditado "lixo na entrada, lixo na saída"), procuramos dedicar tempo suficiente à análise exploratória do conjunto de dados, avançando para a etapa seguinte apenas quando tivermos a certeza de que compreendemos bem os dados.



Tarefas exploratórias de análise de dados

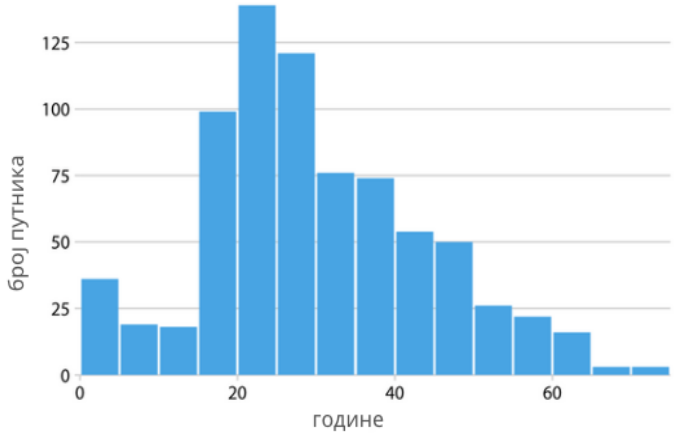
Análise de atributos

Como os atributos são usados para descrever uma grande variedade de propriedades, seus tipos e intervalos de valores variam. Os dois grandes grupos de atributos que encontramos são **atributos numéricos** (quantitativos) e **categóricos** (qualitativos). Os atributos numéricos têm, como o nome indica, valores numéricos. Tais são, por exemplo, a altura do jogador, a distância do aeroporto, o número de animais de estimação, a temperatura exterior, o número de gelados vendidos, a concentração de glicose no sangue e muitos outros. Para esses atributos, geralmente olhamos para os intervalos de valores, os valores mais altos e mais baixos, o valor médio, a mediana, bem como a própria distribuição. Chamamos todas essas **análises de análises descritivas,** porque elas nos ajudam a descrever a quantidade à qual um atributo está associado.



Exemplos de algumas análises descritivas dos atributos do conjunto de dados Iris

Atributos categóricos são um tipo de atributo que pode ter um conjunto finito de valores. Tais atributos são, por exemplo, a cor do carro, o tipo de roupa, o sexo do paciente, a estação atual, entre outros. Esses atributos são geralmente representados por cadeias de caracteres ou códigos numéricos equivalentes. Por exemplo, o mês do ano pode ser listado como o nome "fevereiro" ou como o número "dois" (porque fevereiro é o segundo mês do ano). É importante notar que, mesmo que usemos códigos numéricos para representar esses atributos, não faz sentido calcular valores como média ou máximo, porque esses valores não são inerentemente numéricos. Para eles, geralmente analisamos quais valores eles podem tomar e com que frequência eles aparecem, e mostramos essas conclusões usando gráficos.



Um exemplo da análise do atributo "ano" no conjunto Titanic

A Unificação de Valores

Durante a análise dos dados, podemos descobrir que os valores dos atributos não são uniformemente definidos. Por exemplo, nomes de cores podem ser escritos de forma inconsistente, às vezes em letras minúsculas e às vezes maiúsculas, ou as datas podem ser dadas em formatos diferentes, como dia-mês-ano e ano/mês/dia. Para poder realizar corretamente a tarefa de análise, é desejável unificar esses valores, ou seja, vamos reduzi-los à mesma forma de representação. Normalmente há um caminho que é mais desejável ou útil, mas também acontece que as eleições são completamente iguais.

Valores em falta

Ao analisar um conjunto de dados, podemos notar que os valores de alguns atributos estão faltando. Isso pode ser devido a descuido na entrada de dados ou simplesmente indisponibilidade de informações. Esses valores no conjunto de dados são chamados de valores ausentes.

name and surname	year	work experience	revenues
Marco	48	22	83
Ivan	67	30	110
Ana	34	10	
Peter		4	
Milan	21		85

Um exemplo de um conjunto com valores ausentes

O passo mais simples que podemos tomar quando notamos valores ausentes é excluir os atributos (colunas do conjunto de dados) ou as instâncias (tipos de conjunto de dados) em que eles aparecem. Por exemplo, se não soubermos o valor de um atributo para mais de 50% das instâncias de um conjunto de dados, faz sentido excluí-lo. Se, por outro lado, tivermos apenas algumas instâncias em que o valor do atributo está ausente, é melhor excluir as instâncias e manter o atributo. No entanto, estas decisões nem sempre são fáceis. Por exemplo, pode acontecer que diferentes valores de atributo estejam faltando em instâncias diferentes, então excluímos e ignoramos um número significativo de instâncias dessa maneira, o que pode ser problemático se não tivermos um grande conjunto de dados. É por isso que faz sentido considerar mais algumas opções para trabalhar com valores em falta.

Se o atributo em falta for numérico, por exemplo, a distância ao aeroporto ou a altura do jogador, podemos substituir os valores em falta pelo valor médio dos valores conhecidos. O argumento que temos para essa escolha é que usaremos as informações que já existem no conjunto de dados e que não mudaremos muito sobre algumas outras propriedades de atributo. Por outro lado, se estivermos a falar de atributos categóricos, como a cor do carro ou o país de fabrico, que podem ter um conjunto finito de valores, podemos substituir o valor em falta pelo valor mais comum. Outra opção que é válida para atributos numéricos e categóricos é o uso de valores aleatórios - para que possamos substituir a cor ausente por uma cor aleatória de um possível conjunto de cores, e a altura ausente de um jogador com um valor da faixa da menor e maior altura no conjunto. Em todos os casos, devemos ter cuidado porque as mudanças nos dados podem afetar o sucesso do modelo e os resultados que obtemos. Também é muito importante em que momento fazemos esses reparos. Falaremos sobre isso mais adiante.

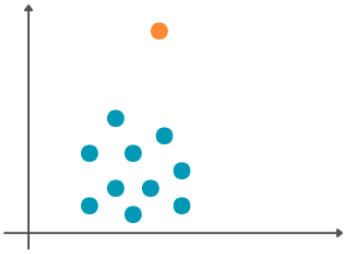
Duplicados

A presença de duplicatas no conjunto de dados pode afetar o poder de generalização do modelo. É por isso que é sempre conveniente verificar se existem dados repetidos ou muito semelhantes. Quando se trata de

dados tabulares, duplicatas podem ser encontradas comparando diretamente os valores dos atributos. Ao trabalhar com diferentes tipos de dados, geralmente precisamos de técnicas mais avançadas. Por exemplo, imagens duplicadas podem ser simétricas, como em um espelho, horizontal ou verticalmente. O mesmo acontece com os dados textuais. Duas notícias podem conter o mesmo anúncio (relatado por algumas agências de notícias) com manchetes ligeiramente diferentes, portanto, em termos de comparação direta de caracteres, elas são diferentes, mas iguais.

Detectar exceções

Perceber dados que são de alguma forma diferentes dos demais nos permite detectar erros nos dados ou descobrir novos comportamentos atípicos. Esses dados são chamados de exceções ou *valores atípicos*. A distância do aeroporto, que é de -1,2 km, seria um número de discrepância, porque esperamos que a distância seja um valor positivo. Dessa forma, poderíamos detectar o erro e corrigi-lo. Por outro lado, uma temperatura de 45 °C é também um valor invulgar, mas real devido às alterações climáticas e talvez muito útil como informação para tomar certas medidas e ações.



Representação gráfica da discrepância

As discrepâncias também podem afetar o resultado dos algoritmos de machine learning. É por isso que, uma vez detectados e processados, é importante decidir se devem ser mantidos ou eliminados.

Correlação de atributos

Os atributos podem estar relacionados entre si. Podemos observar essa relação se desenharmos um gráfico com o valor de um atributo no eixo x e o valor de outro atributo no eixo y. Por exemplo, podemos analisar pares de atributos como a temperatura exterior e o número de gelados vendidos, a temperatura exterior e o consumo de eletricidade, e a temperatura exterior e o número de livros na biblioteca. Cada um destes pares corresponde a um gráfico, como na figura abaixo.

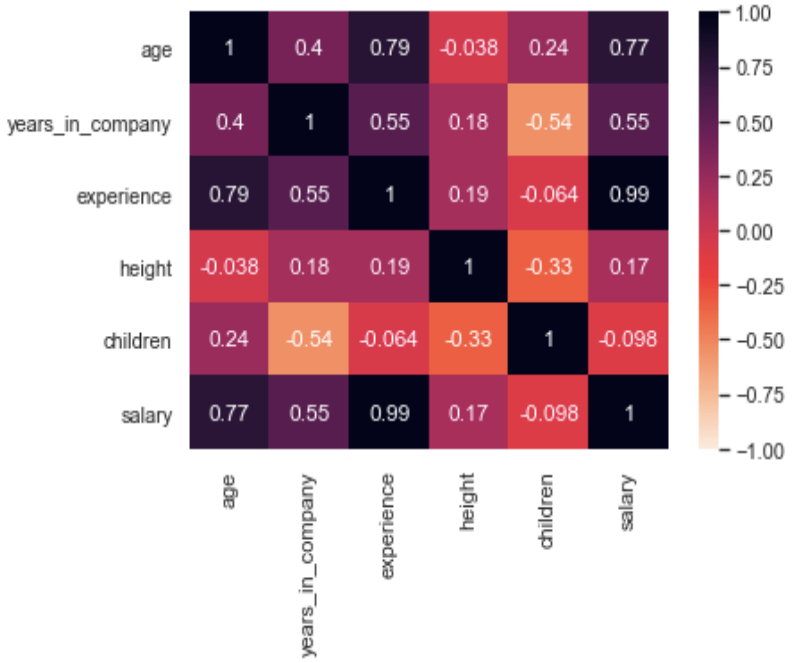
Podemos notar que o aumento da temperatura está acompanhado por um aumento no número de gelados vendidos. Quando o aumento no valor de um atributo é acompanhado pelo aumento no valor do outro, dizemos que estes estão positivamente correlacionados. No gráfico, podemos ainda observar que esta dependência é linear, ou seja, segue uma linha imaginária que atravessa o conjunto de pontos.

Por outro lado, a relação entre a temperatura exterior e o consumo de eletricidade é diferente: a diminuição da temperatura está associada a um aumento do consumo de eletricidade, provavelmente devido ao uso de aquecedores. Atributos em que o aumento de um valor corresponde à diminuição do outro são ditos estar negativamente correlacionados. A partir do gráfico, concluímos novamente que este tipo de correlação é linear. O terceiro gráfico, que relaciona a temperatura exterior com o número de livros na biblioteca, não indica qualquer regularidade óbvia entre os atributos. Podemos concluir que estes atributos não estão linearmente correlacionados.

Gráficos de associação de atributos

Para medir a relação linear de atributos, também podemos usar diferentes tipos de coeficientes que são estabelecidos no domínio da estatística matemática. Um desses coeficientes é o coeficiente de correlação de Pearson. Seus valores variam de -1 a 1 e indicam a direção e a força da conexão. Valores de coeficiente mais próximos de -1 indicam correlação negativa, valores de coeficiente mais próximos de 1 indicam correlação positiva e valores em torno de zero indicam ausência de correlação linear.

É comum que os valores dos coeficientes de correlação entre atributos sejam exibidos graficamente na forma do chamado mapa de calor. Cada quadrado neste mapa corresponde a um par de atributos e sua cor é ajustada ao valor do coeficiente de correlação. A coluna localizada na lateral deste mapa conecta os valores e tons de cores. Ao observar este mapa, podemos facilmente ver as correlações nos dados. A figura abaixo mostra os pares de atributos de um conjunto de dados que combina informações sobre os funcionários. Apesar de sabermos pouco sobre este conjunto, podemos concluir que a experiência e o número de anos (atributo idade) acompanham melhor *os valores salariais*. Também podemos ver que existe uma correlação entre o número de anos (o atributo idade) e o atributo experiência.



Mapa de calor com valores do coeficiente de correlação

Identificar os atributos que estão relacionados permite-nos, em primeiro lugar, compreender melhor o domínio a que os dados se referem. Algumas conexões podem ser esperadas, enquanto outras podem nos trazer novos conhecimentos. Ao excluir atributos vinculados, podemos reduzir a dimensionalidade do

conjunto de dados. Desta forma, podemos acelerar o trabalho de alguns algoritmos e entender os resultados com mais facilidade. Há também algoritmos de machine learning que não se comportam bem se houver associações no conjunto de dados - excluir atributos aos quais isso se aplica pode melhorar o sucesso do algoritmo.

Esta secção é emparelhada com o caderno Jupyter 03-exploratory-data-analysis.ipynb . Se quiser praticar as tarefas que descrevemos, clique no link e, em seguida, no botão 

para abrir o conteúdo no *ambiente do Google Colab*. Se visualizar os blocos de notas na sua máquina local, localize o bloco de notas com o mesmo nome entre os conteúdos e execute-o. Para obter instruções mais detalhadas, consulte a *secção Zona prática* e os cadernos de exercícios Jupyter.

No caderno Jupyter, utilizando as funções da *biblioteca Pandas*, foram analisados os dados do conjunto Titanic. Este conjunto contém informações sobre os passageiros que estavam no famoso navio Titanic quando, em 1912, navegando no Oceano Atlântico, atingiu um iceberg e naufragou.

2.8 Criando uma representação de um conjunto de dados

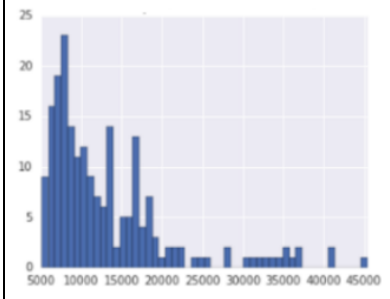
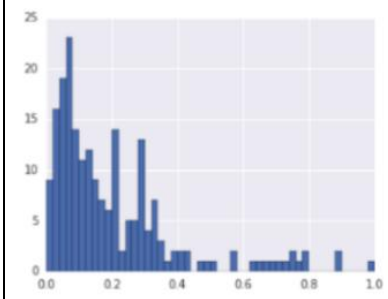
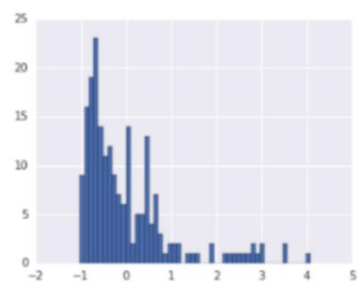
Após a análise exploratória do conjunto de dados, podemos tomar decisões sobre quais atributos e instâncias devem ser descartados. O conjunto de dados restante precisa então de ser preparado, de forma a podermos aplicar alguns dos algoritmos de ML. Dependendo do tipo de atributo e do conjunto de valores que este contém, existem diferentes técnicas de preparação de dados. Abaixo poderá aprender mais sobre algumas dessas técnicas. Na próxima secção, irá descobrir qual o momento ideal para as aplicar.

Preparação de atributos numéricos

Ao trabalhar com atributos numéricos, encontramos quantidades que são expressas em diferentes escalas de valores. Por exemplo, em um único conjunto de dados médicos, pode haver análises laboratoriais com valores que variam de 0 a 1 e informações sobre a altura do paciente expressas em centímetros, de 100 a 250, que é significativamente maior. Muitos modelos de machine learning são sensíveis à presença desses atributos e, portanto, levam mais tempo para encontrar uma solução. Não é fácil entender os resultados que vêm com esse tipo de trabalho. É por isso que, ao trabalhar com dados numéricos, usamos técnicas de normalização que nos ajudam a trazer um conjunto de valores de atributos para os mesmos intervalos de valores.

Uma dessas normalização é *a normalização min-max*. Para esclarecer como ela é realizada, suponha que precisamos normalizar o valor do atributo X para expressar a altura dos pacientes. Seja Xmax = 180 representar a altura máxima dos pacientes e Xmin = 110 o menor. A normalização de *min-max* é realizada aplicando a fórmula $(x - Xmin)/(Xmax - Xmin)$ a cada valor do atributo x. Se x = 165, o novo valor normalizado será $x' = (165 - 110)/(180 - 110) = 0.786$. Desta forma, o valor do atributo é reduzido para um intervalo de 0 a 1.

Um dos aspetos mais importantes da normalização é a padronização. Envolve centralizar o valor do atributo em torno de zero e dimensionar para a variância da unidade. Para esclarecer como ela é realizada, podemos novamente supor que precisamos normalizar o valor do atributo X, que expressa a altura dos pacientes. Vejamos agora Xmédia = 153,2 a altura média no conjunto de dados e $\sigma = 40,23$ o desvio padrão. A padronização é realizada aplicando a fórmula $(x - Xmean)/\sigma$ a cada valor do atributo x. O novo valor padronizado para um paciente cuja altura é x = 165 é agora $x' = (165 - 153,2)/40.23 = 0.293$.

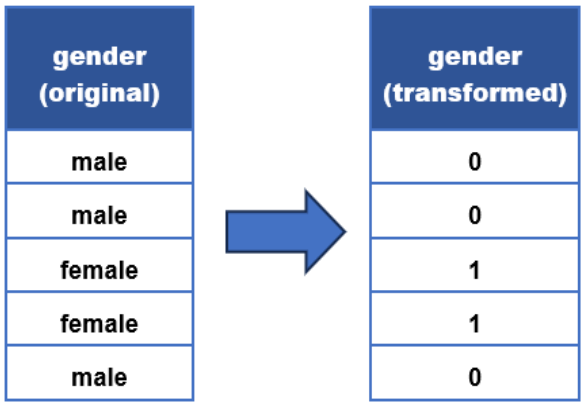
		
Conjunto original	min-max normalização	Normalzação

O efeito da normalização e padronização em um conjunto de dados

Preparação de atributos categóricos

Como os algoritmos de ML só podem ser aplicados a números, os atributos categóricos exigem uma preparação especial. Já referimos que representam quantidades com um número finito de valores e que muitas vezes surgem sob a forma de texto (strings). Alguns dos exemplos que mencionámos são o nome da cor, o sexo do paciente e o mês do ano.

Se um atributo tiver apenas dois valores — por exemplo, representando o sexo de um paciente — os seus valores são frequentemente convertidos para os números 0 e 1. Por exemplo, o valor "feminino" pode ser representado pelo número 1 e o valor "masculino" pelo número 0. Estes atributos são designados por atributos binários.



Exemplo de mapeamento de valor

Para atributos que podem ter vários valores, usamos *codificação one-hot*. Para esclarecer seu significado, podemos olhar para um atributo que representa uma cor, que pode ter três valores: vermelho, amarelo e verde. A ideia é representar o atributo de cor padrão usando três novos atributos, cada um dos quais corresponderá a um dos valores que a cor pode ter: vermelho, amarelo e verde (olhe para a imagem, esta foi uma frase complicada). Isto significa ainda que vamos transformar cada um dos valores do atributo inicial num triplo de valores, ou seja, o valor *do vermelho* num triplo de *1* , *0*, 0, *o* valor *do amarelo* num triplo de *0* , *1* , *0, e* o valor *do verde* num triplo *de 0* , *0* , *1* . Os trigêmeos, como podemos ver, consistem em zeros e exatamente uma unidade na coluna que corresponde ao valor do atributo.

Exemplo de codificação one-hot

Representação do conjunto de dados

Após a etapa de transformação de atributos, chegamos à forma final de dados que podemos usar para executar algoritmos de aprendizagem. Este formulário final é chamado **de representação do conjunto de dados** . Na história até agora, abordamos, em primeiro lugar, como chegar à representação de dados tabulares. E para todos os outros tipos de dados, como imagens, áudios, texto, vídeo-conteúdo, mas também estruturas complexas, como gráficos, precisamos criar representações adequadas. Na secção sobre redes neurais, vamos aprender sobre algumas outras maneiras de criar representações.

2.9 Conjuntos de formação, validação e ensaio

Nesta secção, serão abordados os conjuntos de treino, validação e teste. O conjunto de treino pode ser entendido como o "manual" a partir do qual um modelo de ML adquire conhecimento — é neste conjunto que o modelo encontra exemplos e padrões que sustentam o processo de aprendizagem.O conjunto de teste, por sua vez, funciona como uma avaliação final, destinada a verificar a eficácia do modelo após o término da fase de treino. A sua função é medir o desempenho do modelo e determinar o grau de generalização alcançado.O conjunto de validação é utilizado durante o processo de treino, funcionando como uma ferramenta de monitorização contínua. Pode ser comparado a testes intermédios, cuja finalidade é indicar se o modelo está a evoluir de forma adequada. Com base nos resultados obtidos neste conjunto, é possível realizar ajustes aos parâmetros ou à arquitetura do modelo, com o objetivo de otimizar a aprendizagem antes da avaliação final com o conjunto de teste.

Depois que os dados são analisados e as instâncias e atributos apropriados são selecionados, o conjunto de todos os dados é dividido em **um conjunto de treino** (treinável) e **um conjunto de testes**. Como pode adivinhar pelo nome do conjunto, o conjunto de treino é usado para treinar o próprio modelo de ML. Um algoritmo selecionado é aplicado a ele e cria o próprio modelo. Um conjunto de testes é usado para testar o modelo, ou seja, o cálculo de medidas adequadas de qualidade do modelo. Graças a ele, pode-se avaliar objetivamente o quão bem o modelo aprendeu a tarefa necessária. Normalmente, também usamos uma parte dos dados do conjunto de dados de linha de base para criar **um conjunto de validação**. Um kit de validação é usado para acompanhar o processo de treino de um modelo e determinar algumas configurações de modelo que levam a melhores medidas de qualidade. Estes tópicos serão discutidos mais adiante no curso.

Dividindo um conjunto de dados num conjunto de treino, um conjunto de validação e um conjunto de testes

Os conjuntos de treino, validação e teste geralmente são criados dividindo aleatoriamente um conjunto de dados subjacente. Primeiro, definimos o tamanho desses conjuntos e, em seguida, selecionamos aleatoriamente as instâncias que serão encontradas em cada um deles. Normalmente, o conjunto de treino é o maior, enquanto os conjuntos de teste e validação são menores porque queremos ter dados suficientes para treinar o modelo, mas também dados suficientes para avaliar adequadamente seu desempenho. A prática é que as proporções de tamanho desses conjuntos sejam expressas em proporção. Por exemplo, muitas vezes encontrará a proporção do conjunto de treino para o conjunto de teste expressa como 2:1, o que significaria que dois terços do conjunto inicial é um conjunto de treino e um terço é um conjunto de teste. Da mesma forma, uma proporção de 2:1:1 significaria que dois quartos (ou seja, metade) do conjunto de base seriam usados como um kit de treino e um quarto cada como um conjunto de validação e teste.

Embora seja conveniente que os conjuntos de treino, validação e teste sejam criados aleatoriamente, isso ainda significaria alguma visão sobre como essa divisão foi feita. Por exemplo, quando queremos replicar um experimento ou permitir que outros o executem por conta própria (esta é uma propriedade importante dos experimentos e é chamada de reprodutibilidade), é preferível usar os mesmos conjuntos de treino, validação e teste. Da mesma forma, quando resolvemos um problema, não temos certeza imediata do que é melhor fazer, então tentamos um número maior de algoritmos e criamos um número maior de modelos. Por uma questão de equidade de comparação, isso significaria que criamos todos os modelos sobre o mesmo conjunto de treino e avaliamos sobre o mesmo conjunto de testes. É por isso que é uma boa ideia definir um parâmetro no nível da biblioteca que afeta a aleatoriedade da divisão (geralmente chamada de *semente aleatória* e tem o mesmo propósito de definir sementes em um gerador de números aleatórios) ou simplesmente dividir os dados do zero e continuar a usá-los de forma consistente. Alguns conjuntos de dados comumente usados têm essas divisões predefinidas em um conjunto para treino, validação e teste (por exemplo, pode examinar o *conjunto MNIST*).

Uma propriedade importante que os conjuntos de treino, validação e teste precisam atender é que eles sejam desarticulados. Isso significa que cada instância do conjunto de dados de origem ao criar conjuntos de treino, validação e teste deve pertencer exatamente a um desses conjuntos e não deve haver interseções e instâncias compartilhadas. Deve-se lembrar que espera-se que os modelos de machine learning se generalizem bem, ou seja, que se comportem bem para novas instâncias que o modelo não teve a oportunidade de conhecer no conjunto de treino. Se os conjuntos de treino e os conjuntos de teste se sobrepõem, não seremos capazes de avaliar objetivamente se o modelo realmente aprende ou memoriza. Recorda informações do conjunto de treino. O mesmo se aplica à relação entre o conjunto de treino e o conjunto de validação: a função do conjunto de validação é ajudar a selecionar as configurações que tornarão a aprendizagem tão bem-sucedida quanto possível. Se esses conjuntos se sobrepõem, não seremos capazes de avaliar objetiva e imparcialmente o comportamento do modelo e selecionar as configurações adequadas.

O requisito de que os conjuntos de treino, validação e teste sejam separados significa também que as informações de um conjunto não devem transbordar para os outros. Isto é especialmente importante quando se aplicam técnicas definidas de pré-processamento e preparação dos dados.

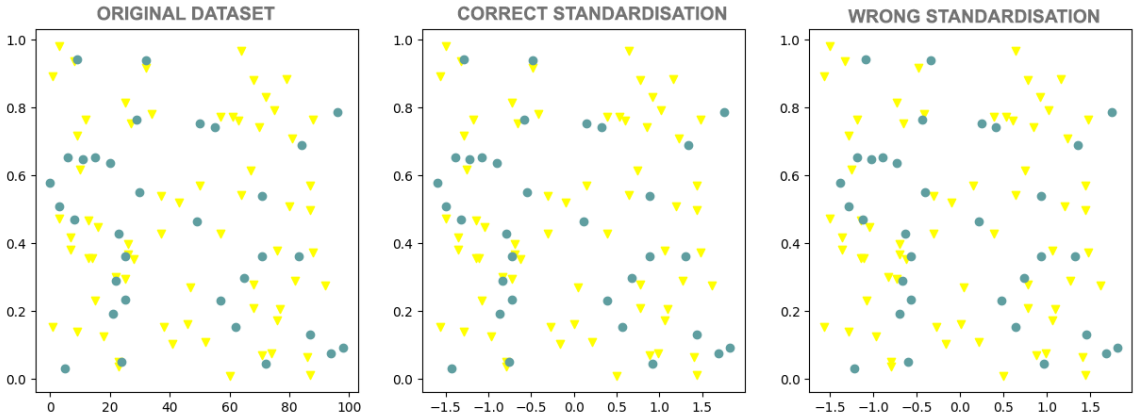
Vamos considerar o exemplo seguinte. Mencionámos que, devido à sensibilidade do modelo de ML aos valores dos atributos, a padronização dos atributos numéricos é frequentemente realizada. Poderia pensar-se em calcular a média e o desvio padrão com base em todo o conjunto de dados e, em seguida, utilizar esses valores para padronizar os conjuntos de treino e teste.

Para evitar o transbordamento de informação, a sequência correta dessas etapas é, na verdade, a seguinte:

- dividir o conjunto de dados num conjunto de treino e testes,
- cálculo da média e do desvio-padrão apenas no conjunto de treino,
- transformação do conjunto de treino utilizando valores calculados,
- Transforme o conjunto de testes usando os valores calculados ao longo do conjunto de treino.

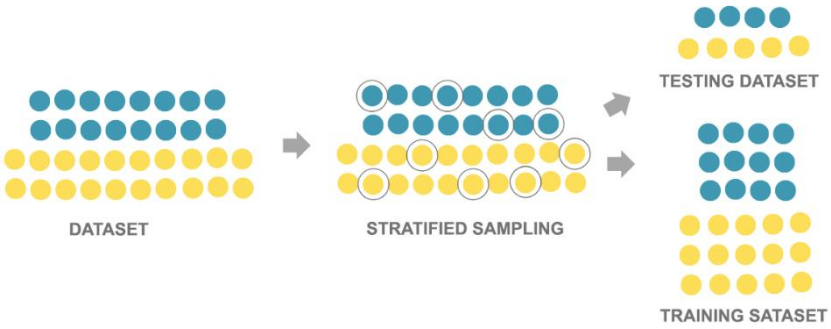
Devido ao desejo de evitar erros, pode parecer razoável fazer o seguinte: depois de dividir o conjunto de dados inicial em conjunto de treino e conjunto de teste, realizar uma padronização separada para cada um desses conjuntos. No entanto, esta abordagem, embora mais cautelosa, não está correta, pois implica modificar o conjunto de teste de forma indevida.

Na imagem inferior esquerda, os triângulos amarelos representam as ocorrências do conjunto de treino e os círculos azuis as do conjunto de teste. A imagem do meio mostra essas instâncias após a padronização correta (podes comparar cuidadosamente as imagens e a disposição dos pontos — a escala no eixo x mudou devido à padronização, mas o resto manteve-se igual). Na imagem à direita, podes ver as instâncias depois de uma padronização feita separadamente para o conjunto de treino e para o conjunto de teste — agora, a disposição espacial mudou bastante.



Exemplos de padronização certa e errada

Ao dividir o conjunto de dados da linha de base, seria ideal preservar as proporções relativas aos valores dos atributos e ao valor da variável de destino. Por exemplo, se a proporção de pacientes do sexo masculino para o feminino no conjunto de dados médicos for de 4:5, seria ideal que, após a divisão, a proporção de pacientes no conjunto de treino e no conjunto de testes fosse de aproximadamente 4:5. As técnicas que permitem esse tipo de divisão são chamadas de técnicas de **estratificação** . No entanto, devido ao número de atributos e suas combinações, na prática, este não é muitas vezes um requisito realista, por isso, na maioria das vezes, insiste-se na proporcionalidade em relação aos valores da variável alvo. Discutiremos este tópico separadamente no contexto da tarefa de classificação.



Montagens estratificadas de treino e teste

3. Modelos de Aprendizagem

Bem-vindo ao tema dos **Modelos de Aprendizagem**! Esta secção mergulha nos vários modelos e técnicas usadas em *machine learning* para resolver diferentes tipos de problemas. Aprenderá sobre regressão linear, classificação, árvores de decisão e o algoritmo dos K vizinhos mais próximos, entre outros. Discutiremos as características, vantagens e desvantagens de cada modelo e a importância da validação do modelo para garantir precisão e confiabilidade. Através de exercícios práticos, irá aplicar estes modelos a conjuntos de dados reais e interpretar os resultados.

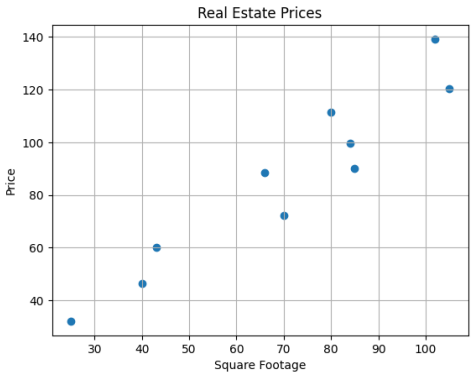
3.1 Regressão Linear

Vamos voltar ao exemplo que usamos para introduzir o paradigma de programação orientada por dados. No conjunto de dados, tínhamos informações acerca da área de imóveis e dos seus preços, e a nossa tarefa era estabelecer a conexão entre esses valores para que pudéssemos estimar preços para novos imóveis.

Para simplificar, vamos ter um conjunto de treino que contém 10 instâncias, que estão apresentadas na tabela abaixo:

Área (m^2)	Preços imobiliários (em milhares de €)
43	60
25	32.1
66	88.4
80	111.4
105	120.32
70	72.1
40	46.3
85	90.1
84	99.6
102	139.2

Também podemos exibir estes valores graficamente. Ao longo do eixo x, definiremos os valores da área, ao longo do eixo y os valores dos preços dos imóveis e marcamos estes pares de valores com círculos azuis.



Vamos escolher para o modelo uma função que relacione a área dos imóveis x com os seus preços y com a equação $y = \beta_0 + \beta_1 x$, onde β_0 e β_1 parâmetros desconhecidos. Este é o chamado **modelo linear** , e como o usamos para resolver o problema da regressão, também o chamamos de **modelo de regressão linear** . Note que esta é na verdade a equação de uma reta $y = kx + n$, onde o declive é denotado por β_1 e a ordenada na origem é denotada por β_0. A motivação para a introdução deste modelo reside no facto dos pontos seguirem a bissetriz dos quadrantes ímpares, talvez ligeiramente abaixo.

Esta secção é emparelhada com *Jupyter Volume* 05-1-linear_regression.ipynb. Para acompanhar o conteúdo, clique no link e, em seguida, no botão Open in Colab para abrir o conteúdo no *Google Colab.* Se estiver visualizando os blocos de anotações no seu dispositivo, localize o bloco de anotações com o mesmo nome entre os conteúdos e execute-o. Para mais instruções detalhadas veja a secção *Hands-on Zone* e a aula *Jupyter Exercise Notebook*.

A sua tarefa é carregar o conjunto de dados imobiliários no bloco de anotações complementar e selecionar os valores dos parâmetros β_0 e β_1 que considera que melhor correspondem a esses dados. Pode ajustá-los movendo os controles deslizantes para a esquerda e para a direita. Lembre-se dos valores que escolheu e das ideias que o guiaram ao determinar os parâmetros.

Provavelmente já tentou obter o valor mais perto possível dos pontos dados e que faz o menor número de desvios possível. Certamente ficou igualmente satisfeito com algumas escolhas de parâmetros, enquanto outras foram menos boas. E do ponto de vista da *machine learning*, tentamos encontrar os valores dos parâmetros β_0 e β_1 para os quais cometemos o menor erro, mas temos que definir exatamente qual é o erro realmente. Veja como vamos fazê-lo.

Suponha que os valores selecionados sejam $\beta_0 = 6.3$ e $\beta_1 = 1.02$. Agora vamos expandir a tabela de dados com uma coluna com os valores calculados por este modelo de regressão linear para os valores de quadratura que temos. Do ângulo do modelo, representamo-los com o tamanho x.

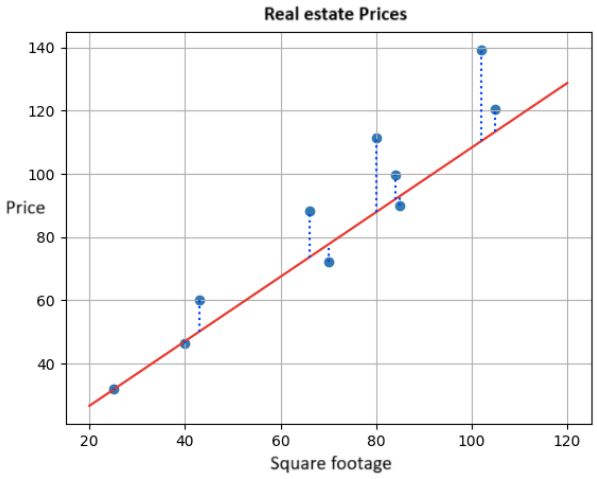
Área (m^2)	Preços imobiliários (em milhares de €)	Preço do modelo y=6.3+1.02x (em milhares de €)
43	60	50.16
25	32.1	31.8
66	88.4	73.62
80	111.4	87.9

105	120.32	113.4
70	72.1	77.7
40	46.3	47.1
85	90.1	93
84	99.6	91.98
102	139.2	110.34

A diferença entre os valores esperados (conhecidos no conjunto de dados) e os valores que calculamos (designados por previsões) é um erro. Agora vamos calcular todos os erros e registá-los na tabela.

Área (m^2)	Preços imobiliários (em milhares de €)	Preço do modelo y=6.3+1.02x (em milhares de €)	Erro do modelo
43	60	50.16	9.84
25	32.1	31.8	0.3
66	88.4	73.62	14.78
80	111.4	87.9	2.53
105	120.32	113.4	6.92
70	72.1	77.7	-5.6
40	46.3	47.1	-0.8
85	90.1	93	-2.9
84	99.6	91.98	7.62
102	139.2	110.34	28.86

Para facilitar o acompanhamento do comportamento dos erros, na imagem abaixo, seus valores são mostrados por linhas pontilhadas azuis.



Para se ter uma ideia do erro total do modelo, é imprudente adicionar os erros individuais, uma vez que alguns valores de erro são positivos e alguns valores são negativos. Portanto, podemos elevá-los ao quadrado e somá-los - isso vai dar-nos informações mais fortes sobre o tamanho do erro, independentemente de ser positivo ou negativo. Se dividirmos a soma resultante pelo número de instâncias no conjunto, vamos ter uma ideia do erro médio do modelo. No nosso caso, é:

$$(9.84^2 + 0.32^2 + 14.782^2 + 23.52^2 + 6.92^2 + (-5.6)^2 + (-0.8)^2 + (-2.9)^2 + 7.62^2 + 28.86^2)/10 = 184.687$$

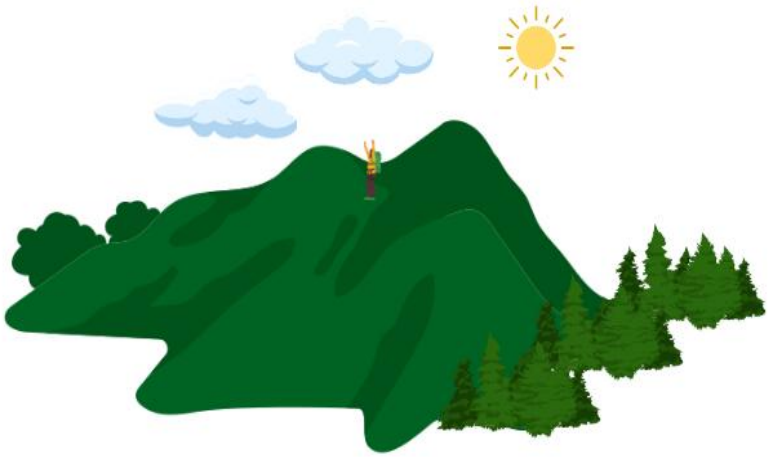
O erro do modelo de regressão linear calculado desta forma é chamado de **erro quadrado médio** (*MSE).* Para valores fixos dos parâmetros β_0 e β_1, o procedimento de cálculo que descrevemos pode ser abreviado pela fórmula $\frac{1}{N}\sum_{i=1}^{N} \left(y_i - (\beta_0 + \beta_1 x_i)\right)^2$. Nele, os pares x_i, y_i correspondem às instâncias individuais, à área das propriedades x_i e aos seus preços y_i, e o número n indica o número total de instâncias. São 10 no nosso caso. A expressão figurada na soma representa a diferença entre o yi esperado e os valores calculados $\beta_0 + \beta_1 x_i$.

O quadrado do erro é o erro que se emparelha sempre com o modelo de regressão linear, e que queremos minimizar o máximo possível, escolhendo os parâmetros corretos β_0 e β_1. A partir da experiência de definir os parâmetros, verifica-se que esta não é uma tarefa muito fácil. Felizmente, existem técnicas matemáticas que podem nos ajudar nisso. Para descobrir como solucionar , vamos avançar para a próxima secção sobre descida de gradiente.

3.2 Descida de Gradiente

Nesta secção, abordaremos a descida de gradiente, uma técnica que nos ajuda a encontrar os parâmetros para os quais a função de erro quadrado médio tem o menor valor. Esta técnica também pode ser aplicada a outras funções sob certas condições.

Terá que imaginar algo novamente - desta vez imagine que está no topo de uma bela montanha. Isso não é assim tão difícil! O problema é que agora a tarefa é: "descer ao pé" rapidamente! Uma maneira de fazer isso é primeiro olhar ao redor e ver em qual direção em sua área a montanha é mais íngreme - tem de ter em mente que precisa de descer muito rapidamente! Então pode dar um passo cuidadoso nessa direção e, em seguida, parar e olhar ao redor novamente. Novamente, pode ver em que direção a montanha é mais íngreme em sua área, dê um passo nessa direção e pare. É claro que pode continuar a repetir esta ordem de observação, escolha de direção e "mergulhe" até chegar ao pé. Há um refresco esperando por você para uma tarefa concluída com sucesso!



Um pouco de atividade física no meio de uma história de regressão linear não faz mal, mas sente-se que há mais alguma coisa. A função de erro quadrado médio depende da escolha dos parâmetros of β_0 e β_1- para diferentes combinações de valores de β_0 e β_1, obtemos valores de erro diferentes. Se traçarmos um gráfico desta função, por exemplo, ao longo do eixo x registamos os valores de β_0, o longo do eixo y registamos os valores de β_1, e ao longo do eixo z registamos os valores de erro, obtemos um gráfico que se parece com o da figura abaixo. Se marcarmos uma seleção aleatória dos parâmetros β_0 e β_1 com um ponto vermelho, para chegar ao ponto em que o valor de erro é menor, realmente temos que descer até o pé dessa superfície. É por isso que a "técnica" que desenvolvemos no exemplo anterior é muito relevante. Só precisamos descobrir como encontrar as direções mais íngremes de descida. As funções ajudar-nos-ão a fazer isso.

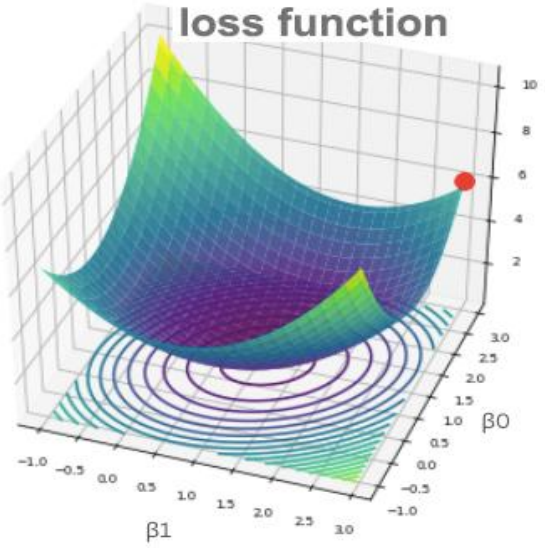
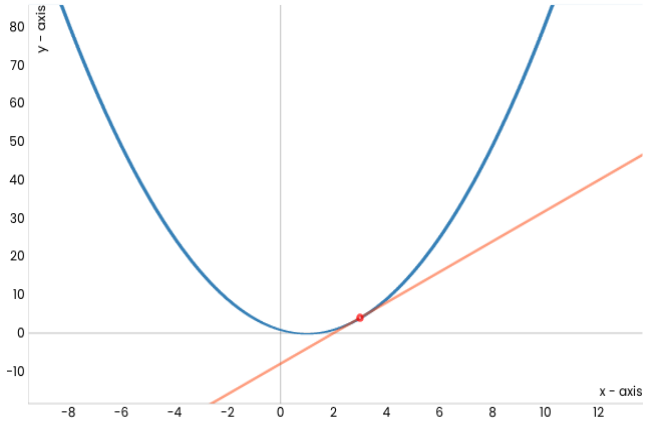


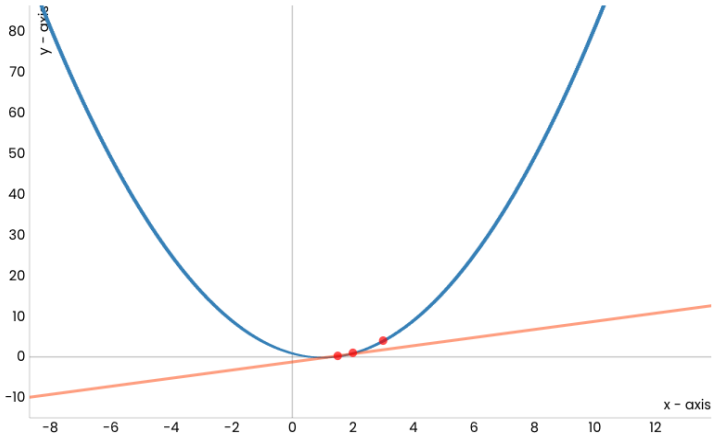
Gráfico da função de erro quadrado médio

O menor valor de uma imagem de uma função é chamado de mínimo.

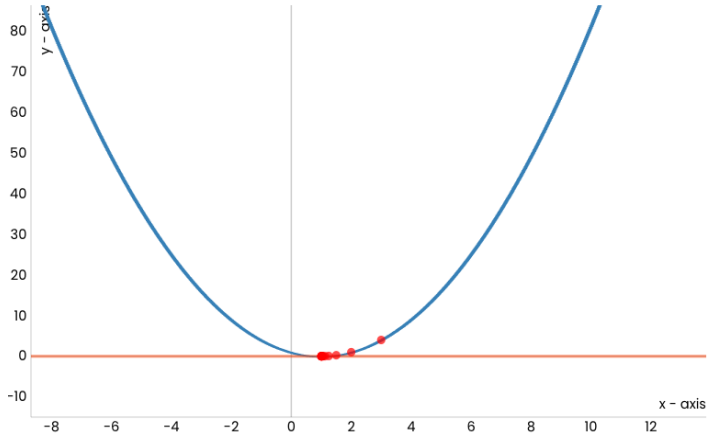
Agora vamos olhar para a função quadrática $f(x) = (x-1)^2$ cujo gráfico é mostrado na figura abaixo, e tentar atingir o seu mínimo com a técnica de descida - está no ponto x=1 e é 0.



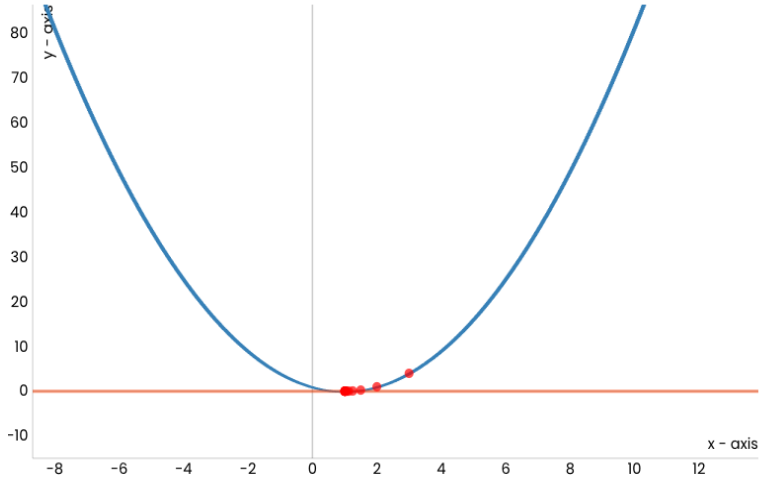
Observe o ponto vermelho correspondente a x=3 (escolhido aleatoriamente) que marca a posição inicial do movimento em direção ao mínimo desta função. Parece que a reta laranja marca a direção mais íngreme ao longo da qual podemos começar a descida. Curiosamente, esta reta na verdade representa a tangente da nossa função no ponto x=3. Se dermos "um passo" nesta reta, encontramo-nos num novo ponto. Vamos marcar seu valor em vermelho e exibi-lo no gráfico. Está um pouco mais perto do mínimo esperado.



Agora podemos repetir o processo: vamos desenhar uma tangente num novo ponto e, em seguida, dar um passo nessa reta.



Após um certo número de etapas, este procedimento levar-nos-á ao mínimo da função, i.e., ao ponto x = 1.



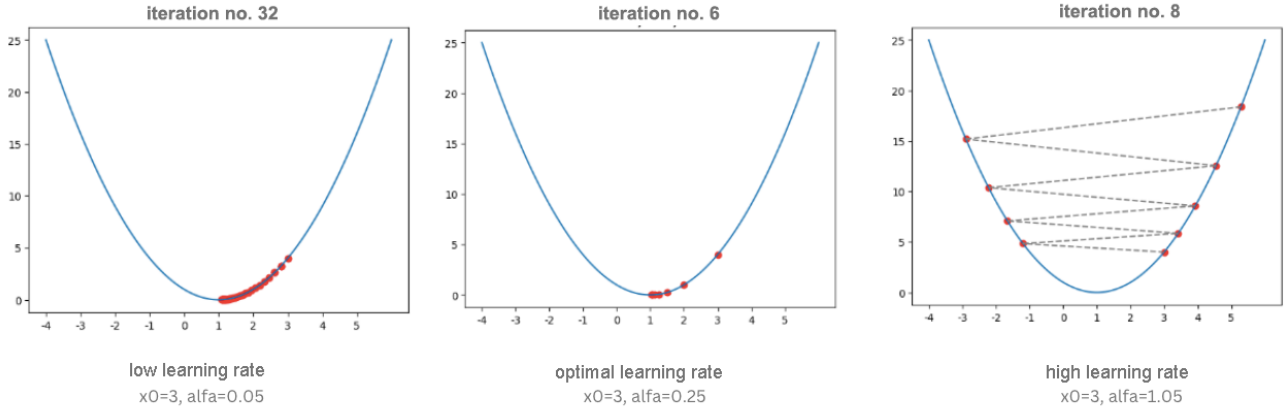
Esta secção está emparelhada com Jupyter Notebook 05-2-gradient_descent.ipynb. Para acompanhar o conteúdo, clique no botão Open in Colab para abrir o conteúdo no *Google Colab.* Se estiver visualizando os blocos de anotações no seu dispositivo, localize o bloco de anotações com o mesmo nome entre os conteúdos e execute-o. Para instruções mais detalhadas, veja a secção *Hands-on Zone* e a aula *Jupyter Exercise Notebook*.

No caderno que acompanha este material, pode iniciar a animação e certificar-se de que é assim.

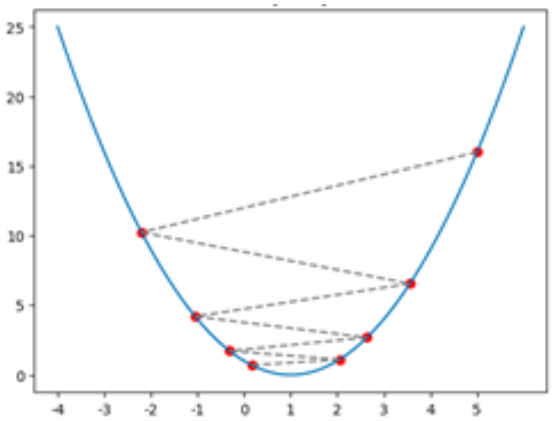
Antes de entrarmos em mais detalhes sobre o procedimento que descrevemos, vamos relembrar o que são essas tangentes na realidade. Para um ponto fixo x, o coeficiente da direção da tangente em x é igual ao valor da primeira derivada da função em x. A primeira derivada da nossa função é a função $f'(x) = 2x - 2$ e no ponto inicial x = 3 o valor da derivada é $f'(x) = 4$. Isto significa que a reta tangente tem a equação $y = 4x - 8$ (o número -8 é obtido a partir da condição de que esta reta deve conter um ponto (3, 4)). É por isso que também podemos dizer que a tangente tem uma direção correspondente à derivada de uma função em um determinado ponto, e para o próprio movimento nessa direção que o movimento ao longo da direção da derivada está nesse ponto. Agora, o dilema é se avançamos ou descemos, ou seja, estamos indo na direção oposta, ou estamos indo na direção correta? Bem, já que queremos descer ao mínimo, precisamos seguir a direção oposta à direção da derivada da função.

Se agora denotarmos o ponto de partida com x_0, obtemos um novo ponto x_1 dando um passo ao longo da direção da derivada da função no ponto x_0. Se denotarmos o comprimento da etapa com α ,calculamos o valor do novo ponto $x_1 = x_0 - \alpha f'(x_0)$. Ao repetirmos o procedimento, calculamos o valor do ponto x_2 como $x_2 = x_1 - \alpha f'(x_1)$ e continuamos com os cálculos $x_3 = x_2 - \alpha f'(x_2), x4 = x_3 - \alpha f'(x_3)$, … Repetimos o procedimento até que, para dois valores consecutivos, digamos x_{34} e x_{35}, s valores da função estejam próximos o suficiente, ou seja, enquanto o valor absoluto da diferença $f(x_{35}) - f(x_{34})$ não seja inferior a alguma precisão predeterminada, digamos 0.001. Assim, computacionalmente, podemos abordar o conceito de convergência em matemática.

O valor α que foi introduzido chama-se **taxa de aprendizagem** e representa um parâmetro muito importante do algoritmo que descrevemos. Se os valores para α forem muito pequenos, demoraremos muito tempo a atingir o mínimo. Por outro lado, se os valores para α forem muito altos, pode acontecer de pularmos o mínimo ou cairmos em uma armadilha de ziguezague pulando constantemente em torno dele! Veja a imagem abaixo!



O Impacto das Escolhas nas Etapas de Aprendizagem



Bloqueio Zigzag

Certifique-se de ambos os comportamentos no bloco de anotações complementar usando as diferentes configurações para a etapa de aprendizagem na animação.

O algoritmo que descrevemos chama-se **Descida de Gradiente** e, apesar da sua simplicidade, é um dos algoritmos mais importantes em *machine learning* porque torna possível encontrar o menor valor de uma função de erro. Há muitos detalhes sobre este algoritmo que não entraremos nas propriedades das funções às quais este algoritmo pode ser aplicado com sucesso, o cálculo numérico da derivada e a seleção de etapas de aprendizagem. Todos eles devem ser considerados durante a aplicação prática do algoritmo.

O algoritmo em si não é desagradável de programar, então vamos embarcar numa aventura. Precisamos de uma função f, que calculará o valor de uma determinada função dada, e de uma função **f_derivative**, que calculará o valor da derivada de uma determinada função. Precisamos definir tanto a etapa de aprendizagem alfa quanto os critérios de paragem: suspenderemos o procedimento quando os valores da função em duas iterações sucessivas estiverem próximos o suficiente (a diferença entre seus valores é menor do que alguma precisão épsilon predeterminada) ou quando atingirmos um número finito de iterações max_iterations (também precisamos ter certeza em casos de escolhas inadequadas de etapas de aprendizagem).

Siga o seguinte código. O algoritmo começa por definir um ponto de partida. Como o ponto em que nos movemos pelo algoritmo de descida de gradiente é o ponto de partida da próxima etapa, usamos os marcadores **x_old** e **x_new** para marcá-los em etapas sucessivas. O relatório que criamos no final da

função contém informações sobre se o algoritmo parou, quantas etapas ele executa, ou seja, iteração necessária e que valor encontrou.

```
def gradient_descent(f, f_derivative, x, alpha, epsilon, max_iterations):
    # set the initial value for x
    x_old = x
    # in each iteration...
    for i in range(0, max_iterations):
        # calculate the current value for x
        x_new = x_old - alpha * f_derivative(x_old)
        # and then check if the stopping criterion is met
        if np.abs(f(x_new) - f(x_old)) < epsilon:
            break
        # if the criterion is not met, prepare x for the next iteration
        x_old = x_new
    # at the end of the whole process, prepare a report with information:
    # whether the algorithm stops,
    # how many iterations it lasted,
    # and what value of x was found
    report = {}
    report['stops'] = i != max_iterations
    report['number_of_iterations'] = i
    report['x_min'] = x_old

    return report
```

A função e sua derivada podem ser definidas pelos seguintes blocos Python:

```
def f(x):
   return (x-1)**2
def f_izvod(x):
   return 2*x-2
```

Depois de executar a função gradient_descent para os valores dos argumentos x0 = 3, alpha = 0.1, epsilon = 0.001, e max_number_of_iterations = 100, obtemos que o mínimo da função é 1.0048, o que podemos confirmar. Também pode executar o código por conta própria e certificar-se de obter o resultado. Não se esqueça de examinar como os resultados mudam se outros valores de argumento forem selecionados.

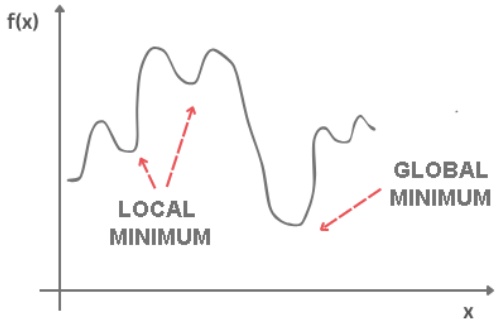
Agora podemos voltar ao problema de encontrar os parâmetros da regressão linear β_0 e β_1 para os quais o valor do erro quadrado médio deve ter o menor valor. A função de erro quadrado médio é uma função de duas variáveis - depende do valor do parâmetro β_0 e do valor do parâmetro β_1. Ao trabalhar com funções de múltiplas variáveis, em geral com n variáveis $x_1, x_2, x_3, \ldots, x_n$, a derivada que usamos no algoritmo de descida de gradiente é generalizada por um vetor de derivadas parciais - para cada uma das variáveis calculamos as derivadas individualmente. Por exemplo, para uma função $\frac{1}{2}(x_1^2 + 10x_2^2)$, a derivada da variável x_1é obtida declarando a variável x_2 como uma constante e, em seguida, aplicando as regras padrão para calcular a derivada que nos levam a $\frac{1}{2} \cdot 2 \cdot x_1 = x_1$. Por outro lado, a derivada da variável x_2 é calculada declarando a variável x_1 como uma constante e aplicando as regras padrão para calcular a derivada. Agora temos $\frac{1}{2} \cdot 10 \cdot 2 \cdot x_2 = 10 \cdot x_2$. Agora temos que o vetor da derivada por variáveis individuais (tais derivadas são chamadas parciais) é o vetor $[x_1, 10 \cdot x_2]$. Em matemática, e mesmo em *machine learning*, esses vetores são chamados de **gradientes**, daí o nome do próprio algoritmo. Para denotar gradientes, usamos o símbolo triângulo para baixo, chamado *nabla*. Assim, a notação precisa do gradiente da função inicial f' seria$\nabla f(x_1, x_2) = [x_1, 10 \cdot x_2]$ e nos permitiria acompanhar quais direções nos deveríamos mover individualmente durante a descida.

As outras etapas do algoritmo de descida de gradiente não diferem muito no caso de funções multivariadas: esperamos que o algoritmo pare depois que a precisão desejada tenha sido alcançada, ou depois que um certo número de iterações tenha sido executado.

Agora que entendemos como funciona a descida de gradiente para funções de múltiplas variáveis, vamos voltar a calcular os parâmetros β_0 e β_1. Referimos que a equação do erro quadrado médio é $\frac{1}{N}\sum_{i=1}^{N} \quad (yi - (\beta_0 + \beta 1xi))^2$. Como esta é uma função para a qual precisamos encontrar o mínimo, se arregaçarmos as mangas e verificarmos, obtemos que a derivada da função quadrada média por β_0 é exatamente $\frac{2}{N}\sum_{i=1}^{N} \quad (\beta_0 + \beta_1 x_i - y_i)$ e a derivada por $\frac{1}{N}\sum_{i=1}^{N} \quad (\beta_0 + \beta_1 x_i - y_i) \cdot x_i$. Essas derivadas indicam quais as direções que devemos seguir e o quanto devemos corrigir os valores de β_0 e β_1 em cada etapa da iteração de descida de gradiente.

No bloco de anotações, também pode ver como esses valores são calculados por meio de código e, em seguida, passar por todo o processo de descida de gradiente personalizado. Para o conjunto imobiliário que introduzimos, chegaremos aos valores de of $\beta_0 = 2.056$ e $\beta_1 = 1.198$.

Dissemos que existem certos pré-requisitos que uma função precisa satisfazer para que seu mínimo seja encontrado pela técnica de descida de gradiente (é necessário que a função seja diferenciável). Também é importante saber que, em geral, um *mínimo local* pode ser obtido desta forma. Por exemplo, a função na figura abaixo tem vários mínimos locais e apenas um *mínimo global* . Em alguns casos, por exemplo, quando uma função é convexa, os mínimos local e global coincidem, por isso chegamos sempre à solução desejada, o mínimo global. A função de erro quadrado é convexa pelos parâmetros β_0 e β_1.



Mínimos locais e globais.

O campo da matemática que lida com a determinação de valores máximos e mínimos de funções (nós os chamamos de ótimos) é chamado **de otimização matemática** . A descida de gradiente é apenas um algoritmo deste campo.

3.3 Regressão polinomial

E se os seus dados forem realmente mais complexos que não possam ser aproximados por uma simples reta? Surpreendentemente, pode realmente usar um modelo linear para ajustar dados não lineares. Uma maneira simples de fazer isso é adicionar poderes de cada recurso como novos recursos e, em seguida, treinar um modelo linear sobre esse conjunto estendido de recursos. Esta técnica é chamada de Regressão Polinomial.

A Regressão Polinomial é um algoritmo de regressão que modela a relação entre uma variável dependente (y) e independente (x) como polinômio de grau n. A equação de Regressão Polinomial é dada abaixo:

$$y = \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \cdots \ldots \beta_n x_1^n$$

É também chamado de caso especial de Regressão Linear Múltipla em ML. Porque adicionamos alguns termos polinomiais à equação de regressão linear múltipla para convertê-la em regressão polinomial.

É um modelo linear com algumas modificações a fim de aumentar a precisão.

O conjunto de dados utilizado na regressão polinomial para treino é de natureza não linear.

Ele faz uso de um modelo de regressão linear para ajustar as funções e conjuntos de dados complicados e não lineares.

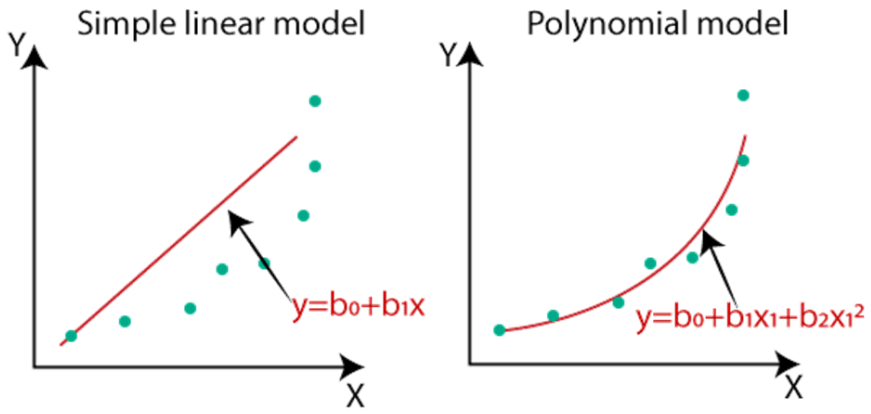
Assim, *"Na regressão polinomial, as características originais são convertidas em características polinomiais de grau requerido (2,3,..,n) e, em seguida, modeladas usando um modelo linear."*

A necessidade de Regressão Polinomial em ML pode ser entendida nas seguintes situações:

- Se aplicarmos um modelo linear a um **conjunto de dados linear**, ele fornece-nos um bom resultado, como vimos na Regressão Linear Simples, mas se aplicarmos o mesmo modelo sem qualquer modificação em um **conjunto de dados não linear**, então ele produzirá um output drástico. Devido ao qual a função de perda aumentará, a taxa de erro será alta e a precisão será diminuída.

- Portanto, para esses casos, **onde os pontos de dados são organizados de forma não linear, precisamos do modelo de Regressão Polinomial**. Podemos entendê-lo de uma maneira melhor

usando o diagrama de comparação abaixo do conjunto de dados linear e do conjunto de dados não linear.



- Na imagem acima, tomamos um conjunto de dados que está organizado de forma não linear. Então, se tentarmos cobri-lo com um modelo linear, podemos ver claramente que ele quase não cobre qualquer ponto de dados. Por outro lado, verificamos que uma curva é adequada para cobrir a maioria dos pontos de dados, que é do modelo Polinomial.

- Assim, *se os conjuntos de dados são organizados de forma não linear, então devemos usar o modelo de Regressão Polinomial em vez de Regressão Linear Simples.*

Nota: Um algoritmo de Regressão Polinomial também é chamado de Regressão Linear Polinomial porque não depende das variáveis, em vez disso, depende dos coeficientes, que são organizados de forma linear.

Equação de Regressão Linear Simples	$y = \beta_0 + \beta_1 x_1$
Equação de Regressão linear Múltipla	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + + \cdots \ldots \beta_n x_n$
Equação de Regressão Polinomial	$y = \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \cdots \ldots \beta_n x_1^n$

Implementação de Regressão Polinomial utilizando Python:

Vamos implementar a Regressão Polinomial usando Python. Vamos entendê-lo comparando o modelo de Regressão Polinomial com o modelo de Regressão Linear Simples. Então, primeiro, vamos entender o problema para o qual vamos construir o modelo.

Descrição do problema: Existe uma empresa de Recursos Humanos, que vai contratar um novo candidato. O candidato disse que o seu salário anterior era 160 mil por ano, e o RH tem que verificar se ele está dizendo a verdade ou não. Então, para verificar isso, eles têm apenas um conjunto de dados de sua empresa anterior em que os salários dos 10 primeiros cargos são mencionados com seus níveis. Ao verificar o conjunto de dados disponíveis, verificamos que existe uma **relação não linear entre os níveis de Cargos e os salários**. O nosso objetivo é construir um **modelo de regressão do detetor de fingimento "Bluffing"**, para que o RH possa contratar um candidato honesto. Abaixo estão os passos para construir tal modelo.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Etapas para Regressão Polinomial:

As principais etapas envolvidas na Regressão Polinomial são dadas abaixo:

- Pré-processamento de dados
- Crie um modelo de Regressão Linear e ajuste-o ao conjunto de dados
- Crie um modelo de Regressão Polinomial e ajuste-o ao conjunto de dados
- Visualize o resultado para o modelo de Regressão Linear e Regressão Polinomial.
- Prevendo o output.

Etapa de pré-processamento de dados:

A etapa de pré-processamento de dados permanecerá a mesma dos modelos de regressão anteriores, exceto por algumas situações. No modelo de Regressão Polinomial, não usaremos dimensionamento de recursos e também não dividiremos nosso conjunto de dados em conjunto de treino e teste. Tem duas razões:

- O conjunto de dados contém muito menos informações que não são adequadas para dividi-lo num conjunto de teste e treino, caso contrário, nosso modelo não será capaz de encontrar as correlações entre os salários e níveis.
- Neste modelo, queremos previsões muito precisas para o salário, por isso o modelo deve ter informações suficientes.

O código para a etapa de pré-processamento é dado abaixo:

```
# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets
```

```
data_set= pd.read_csv('Position_Salaries.csv')

#Extracting Independent and dependent Variable

x= data_set.iloc[:, 1:2].values

y= data_set.iloc[:, 2].values
```

Construindo o modelo de regressão linear:

Agora, vamos construir e ajustar o modelo de regressão linear ao conjunto de dados. Na construção da regressão polinomial, tomaremos como referência o modelo de regressão linear e compararemos ambos os resultados. O código é dado abaixo:

```
#Fitting the Linear Regression to the dataset

from sklearn.linear_model import LinearRegression

lin_regs= LinearRegression()

lin_regs.fit(x,y)
```

No código acima, criamos o modelo Simple Linear usando **lin_regs** objeto da classe **LinearRegression** e ajustámo-lo às variáveis do conjunto de dados (x e y).

Output:

Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

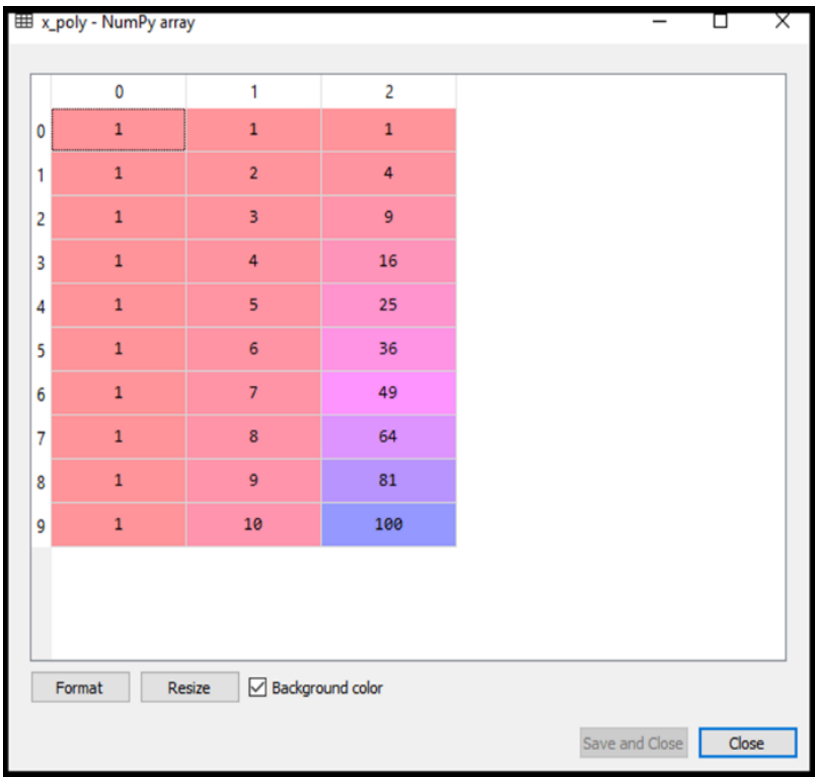
Construindo o modelo de regressão polinomial:

Agora vamos construir o modelo de Regressão Polinomial, mas será um pouco diferente do modelo Linear Simples. Porque aqui vamos usar **PolynomialFeatures** classe de biblioteca de **preprocessing**. Estamos usando essa classe para adicionar alguns recursos extras ao nosso conjunto de dados.

```
#Fitting the Polynomial regression to the dataset

from sklearn.preprocessing import PolynomialFeatures

poly_regs= PolynomialFeatures(degree= 2)

x_poly= poly_regs.fit_transform(x)

lin_reg_2 =LinearRegression()

lin_reg_2.fit(x_poly, y)
```

Nas linhas de código acima, usamos **poly_regs.fit_transform(x),** porque primeiro estamos convertendo nossa matriz de feição em matriz de feição polinomial e, em seguida, ajustando-a ao modelo de regressão polinomial. O valor do parâmetro (grau= 2) depende da nossa escolha. Podemos escolhê-lo de acordo com as nossas características polinomiais.

Depois de executar o código, obteremos outra matriz **x_poly**, que pode ser vista na opção explorador de variáveis:



	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

Em seguida, usamos outro objeto LinearRegression, ou seja, **lin_reg_2**, para ajustar nosso vetor x_poly ao modelo linear.

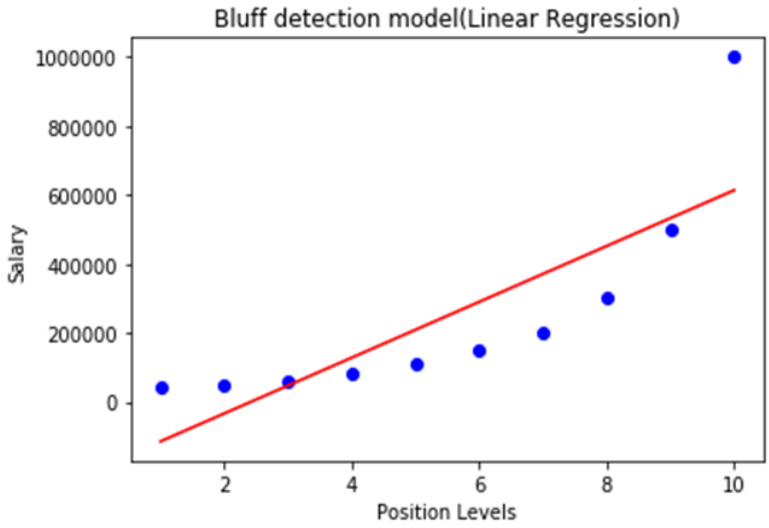
Output:

Out[11]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

Visualizando o resultado da regressão linear:

Agora vamos visualizar o resultado para o modelo de Regressão Linear como fizemos em Regressão Linear Simples. Abaixo apresenta-se o código:

```
#Visulaizing the result for Linear Regression model

mtp.scatter(x,y,color="blue")

mtp.plot(x,lin_regs.predict(x), color="red")

mtp.title("Bluff detection model(Linear Regression)")

mtp.xlabel("Position Levels")

mtp.ylabel("Salary")

mtp.show()
```



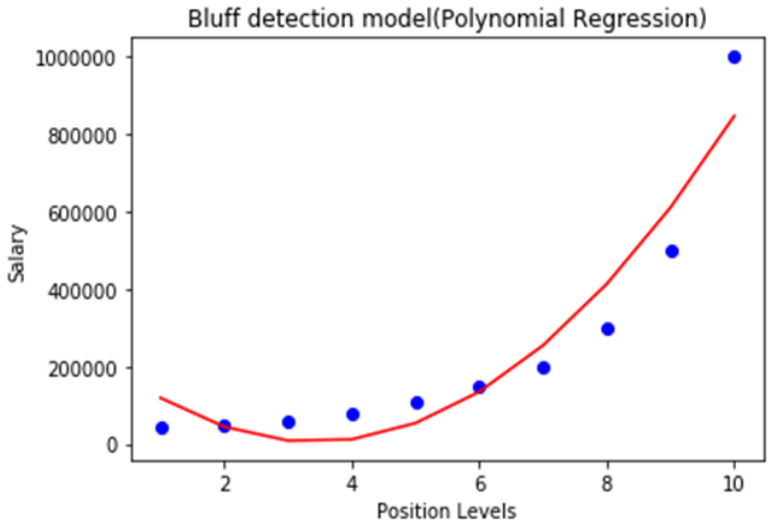
Na imagem de output acima, podemos ver claramente que a reta de regressão está longe do conjunto de dados. As previsões estão em numa reta vermelha, e os pontos azuis são valores reais. Se considerarmos este output para prever o valor do CEO, dará um salário de aprox. 600000$, o que está longe do valor real.

Portanto, precisamos de um modelo curvo para ajustar o conjunto de dados que não seja uma reta.

Visualizando o resultado da Regressão Polinomial

Aqui vamos visualizar o resultado do modelo de regressão polinomial, cujo código é pouco diferente do modelo acima.

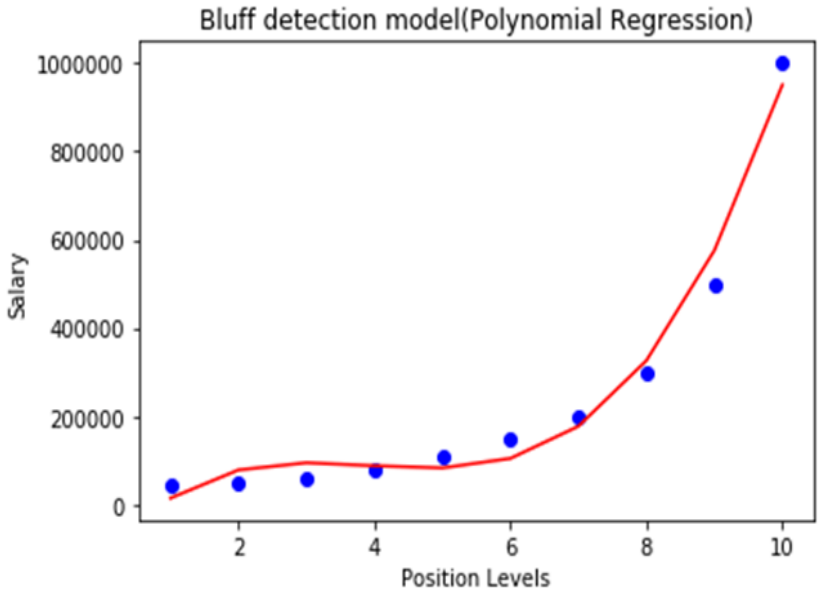
No código acima, considerámos lin_reg_2.predict(poly_regs.fit_transform(x)), em vez de x_poly, porque queremos que um objeto de regressão linear preveja a matriz de características polinomiais.



Como podemos ver na imagem de output acima, as previsões estão próximas dos valores reais. O gráfico acima irá variar conforme vamos mudando o grau.

Para grau= 3:

Se mudarmos o grau = 3, então daremos um gráfico mais preciso, como mostrado na imagem abaixo.



Então, como podemos ver aqui na imagem de output acima, o salário previsto para o nível 6.5 está perto de 170K$-190k$, o que parece que o futuro funcionário está dizendo a verdade acerca do seu salário.

Grau= 4: Vamos novamente mudar o grau para 4, e agora obteremos o gráfico mais preciso. Assim, podemos obter resultados mais precisos aumentando o grau do Polinómio.

Prevendo o resultado final com o modelo de Regressão Linear:

Agora, vamos prever o output final usando o modelo de regressão linear para ver se um funcionário está dizendo a verdade ou mentira. Então, para isso, usaremos o **método predict()** e passaremos o valor 6.5. Abaixo está o código:

```
lin_pred = lin_regs.predict([[6.5]])

print(lin_pred)
```

Output:

[330378.78787879]

Prevendo o resultado final com o modelo de Regressão Polinomial:

Agora, vamos prever o output final usando o modelo de Regressão Polinomial para comparar com o modelo Linear. Abaixo está o código;

```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))

print(poly_pred)
```

Output:

[158862.45265153]

Como podemos ver, o output previsto para a Regressão Polinomial é [158862.45265153], o que é muito mais próximo do valor real, portanto, podemos dizer que o futuro funcionário está dizendo a verdade.

3.4 Regressão Linear Múltipla

Na história introdutória sobre conjuntos de dados, vimos que é usado um número maior de atributos. No entanto, na história da regressão linear, utilizamos apenas um atributo (a área da propriedade). Provavelmente está a questionar o que fazemos quando temos vários atributos e se podemos então aplicar um modelo de regressão linear.

O modelo de regressão linear adaptado a este cenário é chamado de **regressão linear múltipla** e tem a forma $y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_n X_n$. Não se confunda com esta longa expressão - agora os valores $X_1, X_2, X_3, \ldots, X_n$ representam os atributos individuais e os parâmetros $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ …, são os parâmetros do modelo. Por trás dessa generalização está novamente a ideia de uma relação linear entre os atributos individuais e a variável alvo.

O objetivo da aprendizagem é determinar os valores dos parâmetros $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ e, assim, ter uma ideia das dependências. Chegamos a eles da mesma forma que com a regressão linear que conhecemos (também dizemos que é simples): minimizando o erro quadrado médio no conjunto de dados de treino. A técnica de descida de gradiente pode ser generalizada para se adequar a essa configuração de tarefa e pode nos ajudar a encontrar o conjunto de valores $\beta_0, \beta_1, \beta_2, \beta_3, \ldots, \beta_n$ para os quais o erro quadrado médio é menor.

No caso de um modelo de regressão linear de atributo único, também poderíamos imaginar o significado dos parâmetros β_0 e β_1: eles determinaram o deslocamento e a inclinação da linha que passa pelo conjunto de dados. Assim, eles mostraram-nos a força da dependência linear entre as variáveis de imput e output, ou seja, o quanto o valor da variável de output y muda quando mudamos o atributo x por 1. Agora que temos mais parâmetros, é natural que nos perguntemos que significado lhes podemos dar. Têm o mesmo tipo de dependência. Se imaginarmos que apenas β_0 e β_2: são parâmetros diferentes de zero, então a relação entre a variável alvo y e o atributo X2 é representada pela equação $y = \beta_0 + \beta_2 X_2$, , ou seja, Linear e o mesmo nos diz quanto o valor para a variável alvo y mudará e em que direção quando alteramos o valor para X_2 por 1.

Dado que os parâmetros resumem o conhecimento do conjunto de dados para nós, no caso de regressão linear múltipla, valores maiores de parâmetros indicam uma maior significância de um atributo sobre o valor da variável alvo. Para poder acompanhar essa propriedade, geralmente representamos os valores dos parâmetros calculados com um gráfico de colunas. A figura abaixo mostra os valores dos parâmetros de um modelo que usa um conjunto de dados reais para prever os preços dos imóveis (os populares imóveis de Boston). Sem entrar em muitos detalhes sobre este conjunto, podemos notar imediatamente que o atributo LSTAT tem mais, e negativamente, no valor da variável alvo, enquanto os atributos RM e RAD têm um efeito positivo, quase igualmente. Gráficos desse tipo, que podem nos dar alguma ideia da influência dos atributos, são chamados de gráficos de **importância de atributos** (*feature importance graph*).

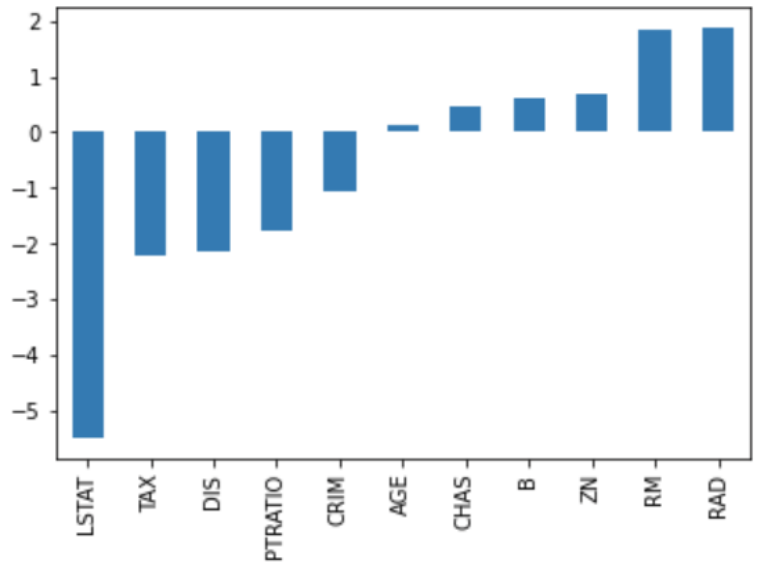


Gráfico da importância dos atributos de regressão múltipla

Outro detalhe que deve ser enfatizado, para não surpreender mais tarde, diz respeito à linearidade. O modelo de regressão linear é **linear em termos de parâmetros**. Isso significa que um modelo cuja forma$y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$ ina qual os graus de valores de atributos são calculados, seria executado como um modelo linear. É semelhante para o modelo $y = \beta_0 + \beta_1 log(X)$, no qual o logaritmo do valor do atributo é figurado. Pode pensar nessas funções de atributo, talvez inesperadas, como transformações que fixam a relação linear entre o atributo e a variável de destino.

3.5 Classificação, tipos de classificação e matriz de confusão

Acredite ou não, certamente já se questionou muitas vezes antes. Quando está arrumando um quarto e separando pedaços de papel para guardar ou jogar fora, ou quando está classificando as suas fotos naquelas de uma excursão, aniversário da tia, ou uma viagem com amigos, está realmente fazendo uma tarefa de classificação: tem grupos em mente, e quando olha para um pedaço de papel ou uma foto, está decidindo a qual grupo pertence. E muitos dos programas que encontra fazem a tarefa de classificação. Por exemplo, o seu cliente de e-mail distingue entre correio desejável e não solicitado e, graças a esta funcionalidade, consegue evitar muitas armadilhas e fraudes na Internet. Além disso, nas redes sociais, muitas vezes recebe recomendações para se conectar com novas pessoas - um programa que está por trás da rede social avalia ativamente se uma pessoa é um seu potencial amigo ou não (geralmente segue os amigos de seus amigos e obtém ideias). Como não temos dúvidas de que é um especialista em organizar a sala e os arquivos no seu computador, vamos aprender como essas habilidades são dominadas pelos programas!

Tipos de classificação

Logo no início, é importante ressaltar que nem todas as classificações são iguais. Portanto, vamos primeiro descobrir quais classificações existem e o que as carateriza. Exemplos de triagem de pedaços de papel ou de triagem de correio são exemplos **de classificação binária** porque temos apenas dois grupos: pedaços de papel a deitar fora e pedaços de papel a armazenar, ou seja, correio desejável e indesejável. Os grupos no mundo da machine learning são chamados de **classes**, então continuaremos a manter essa designação. Para podermos distinguir entre classes, associamo-las a nomes que se aproximam do que realmente contêm. Por exemplo, "folhas de papel" e "lixo eletrónico" são nomes claros o suficiente. Os nomes geralmente são especificados por rótulos, que aparecem no conjunto de dados ao qual a tarefa de classificação é aplicada.

Se tivermos mais de duas classes, estamos a falar de **uma tarefa de classificação multiclasse** . Por exemplo, essa é a tarefa de classificar fotos por eventos onde cada evento pode representar uma classe. Podemos criar três diretórios, ou seja, três classes, dar-lhes os nomes "excursão", "aniversário da tia" e "viagem", e depois atribuir cada uma das fotos a uma dessas classes, colocando-as no diretório apropriado.

Podemos pensar em diferentes tipos de classificação com base nos critérios de afiliação. Por exemplo, um e-mail pode ser desejável ou indesejável, não pode pertencer à classe de e-mails desejáveis e indesejáveis ao mesmo tempo. É semelhante com as fotografias e classes que introduzimos. Por outro lado, um artigo de jornal pode ser simultaneamente sobre o tema da cultura, das viagens e da gastronomia, pelo que podemos associá-lo a um maior número de classes - a que representa a cultura, a que representa as viagens e a que representa a comida. Como neste caso as instâncias têm mais características, ou seja, rótulos, chamamos esse tipo **de classificação multilabelar** . Embora seja muito interessante e útil, não abordaremos a classificação multilabelar com mais conteúdos, mas nos concentraremos na classificação binária e multiclasse.

A que tipo de classificação considera que pertencem as seguintes tarefas:

- triagem do lixo para reciclagem,
- determinar a correção do programa,
- determinação da língua do documento,
- verificar a validade de uma transação bancária.

- próxima sugestão de palavra ao digitar uma mensagem SMS

Classification from the point of view of machine learning

When we think about the classification task from the point of view of machine learning, we are interested in discrete mappings, i.e. mappings that can assign one of finite values to input variables. Most often, the number of classes is smaller, expressed in a single digit number, but you can also recall the *ImageNet* set and the image classification competition in which 1000 classes are used. Variables that can take a finite number of values are called categorical, so we can talk about classification. It's like a mapping that is characterized by a categorical target variable.

Classificação do ponto de vista de *machine learning*

Quando pensamos na tarefa de classificação do ponto de vista da machine learning, estamos interessados em mapeamentos discretos, ou seja, mapeamentos que podem atribuir um dos valores finitos às variáveis de entrada. Na maioria das vezes, o número de classes é menor, expresso em um número de um dígito, mas também pode se lembrar do *conjunto ImageNet* e da competição de classificação de imagem na qual 1000 classes são usadas. As variáveis que podem ter um número finito de valores são chamadas categóricas, então podemos falar sobre classificação É como um mapeamento que é caraterizado por uma variável alvo categórica

$$F(x) = f(x) = \{0, \quad x < 0 \; \frac{1}{2}, \quad 0 \leq x < 1 \; 1, \quad x > 1$$

Um exemplo de uma função discreta.

Como a classificação é uma tarefa supervisionada de *machine learning*, o conjunto de dados usado para treinar o modelo contém pares de inputs e outputs esperados. Os inputs podem ser imagens, mensagens de texto ou dados tabulares, e todas as diretrizes discutidas na secção sobre a preparação de um conjunto de dados se aplicam à sua preparação. O output é sempre o nome da classe. Embora tenhamos introduzido nomes de classe para facilitar o acompanhamento da tarefa de classificação, quando chegamos à parte de preparação dos dados, precisamos transformá-los em valores numéricos também. Aqui podemos ser guiados pelos preparativos que discutimos para trabalhar com atributos categóricos: mapeamento de um conjunto de valores ou *uma codificação a quente (one-hot* coding).

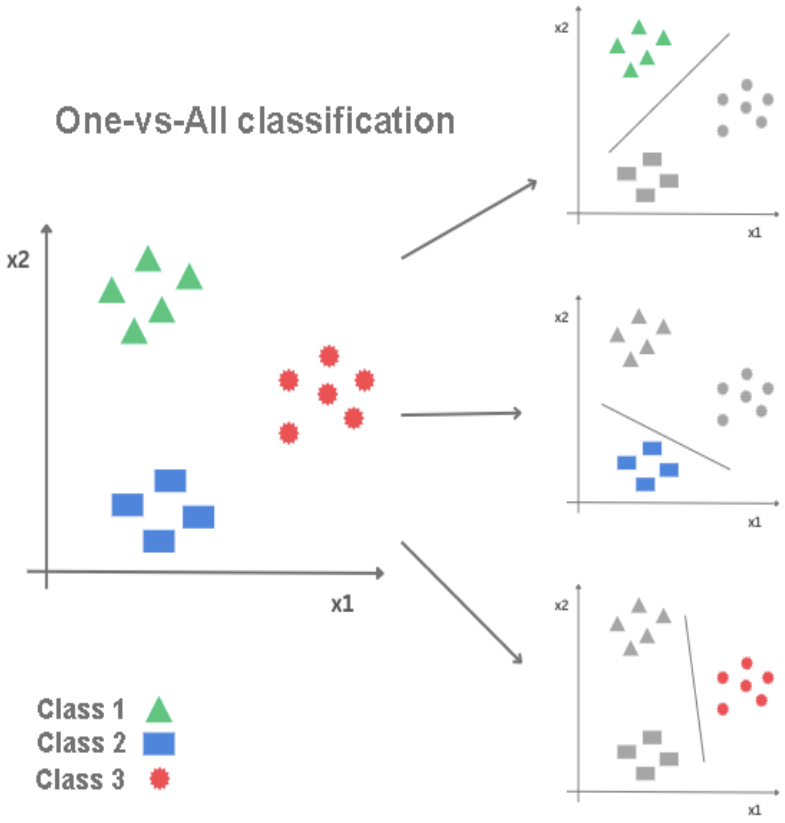
No caso da classificação binária, geralmente mapeamos os nomes das classes com os valores 0 e 1. Por exemplo, a ocorrência do nome da classe "lixo eletrônico" é substituída pelo valor 0 e a ocorrência do nome "lixo eletrônico" pelo valor 1. Muitas vezes, os casos que têm um rótulo de 0 são ditos pertencer a **uma classe negativa**, e os casos que têm um rótulo de 1 são ditos pertencer **a uma classe positiva** .

Quando se trata de classificação multiclasse, usamos *codificação a quente para preparar a variável de destino* . Por exemplo, para a tarefa de classificar fotos por eventos, vamos transformar os outputs em vetores de comprimento três, porque temos exatamente três classes: "excursão", "aniversário da tia" e "viagem". Em seguida, atribuímos a cada um desses valores um vetor que tem um na posição apropriada e zero em todas as posições restantes. Serão esses, por ordem, os valores de (1, 0, 0), (0, 1, 0) e (0, 0, 1). Aqui é importante aderir consistentemente à ordem de classes escolhida.

Abaixo, vamos conhecer os dois algoritmos usados para a tarefa de classificação binária. O problema da classificação multiclasse pode ser resolvido através de algoritmos especialmente concebidos, mas também através de vários classificadores binários associados. Vamos aproximar duas dessas técnicas chamadas "um contra todos" e "um contra um".

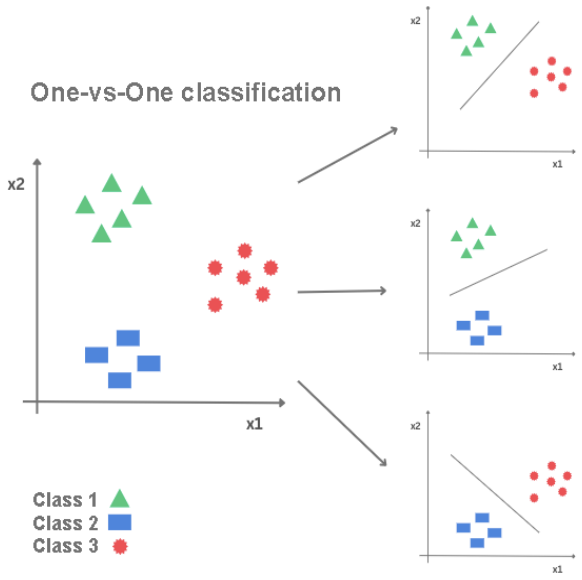
Vamos imaginar que temos três classes: vermelho, verde e azul. A abordagem "um contra todos" implica que precisamos aprender três classificadores binários: um que distingue a classe verde do resto (sindicatos

das classes vermelha e azul), um que distingue a classe azul do resto (sindicatos das classes verde e vermelha), e um que distingue a classe vermelha do resto (sindicatos das classes verde e azul). Quando precisamos classificar uma nova instância, executamos cada um dos três classificadores binários e aplicamos o princípio da mais alta confiança aos resultados obtidos: a instância se junta à classe cujo classificador é o mais seguro. Veremos em breve como a segurança do classificador é avaliada.



Classificação Um contra todos

Novamente, imagine que temos três classes: vermelho, verde e azul. A abordagem de um-para-um envolve treinar classificadores binários que podem distinguir entre cada um dos pares de classes: vermelho e verde, verde e azul, e vermelho e azul. Em geral, se tivermos n classes, o número de classificadores binários que precisamos treinar é $\frac{n \cdot (n-1)}{2}$. Quando precisamos classificar uma nova instância, executamos cada um dos classificadores aprendidos e aplicamos o princípio da votação por maioria aos resultados obtidos: a instância entra na classe para a qual o maior número de classificadores votou.

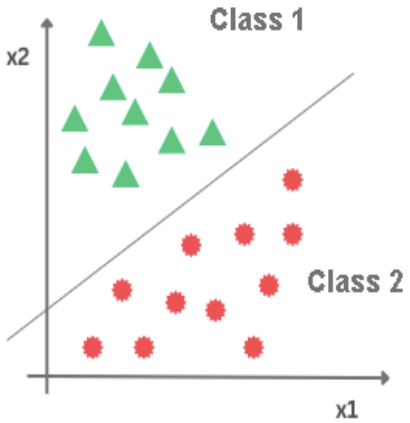


Classificação Um contra todos

3.6 Regressão logística

A regressão logística é um algoritmo bem conhecido usado para criar modelos de classificação binária. Além de saber a qual classe uma instância pertence, ele também calcula a probabilidade de pertencer a essa classe.

Vamos imaginar que temos um conjunto de dados com dois atributos, X_1 e X_2, e que as instâncias desse conjunto são mostradas como na figura abaixo. Ao longo do eixo x, é representado o atributo X_1, ao longo do eixo y, o atributo X_2, e a cor dos pontos indica a classe à qual cada uma dessas instâncias pertence. Concorda que um modelo linear que determina uma reta em um plano poderia nos ajudar a resolver o problema de classificação, separando as classes - uma abaixo desta reta e outra acima. Para podermos concluir desta forma, beneficiaremos da função sigmoide.



A função sigmoide é uma função popular na história da machine learning. É determinado pela equação:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

A imagem dela é parecida com a da imagem abaixo.

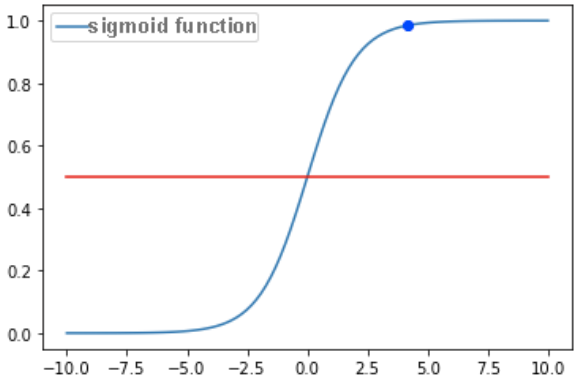


Gráfico da função sigmoide

Podemos notar imediatamente que esta função tem um intervalo de valores de 0 a 1. Quanto menores os valores de x, mais próximo o valor desta função é de 0 e, da mesma forma, quanto maior o valor de x mais próximo o valor da função sigmoide é 1. Para x= 0, o valor da função sigmoide é 0,5. Se declararmos este valor como um limiar e introduzirmos as regras:

1. Se o valor da função sigmoide for maior ou igual a 0,5, associe x a uma classe positiva e
2. Se o valor da função sigmoide for inferior ao limiar de 0,5, associe x à classe negativa

obteremos uma função adequada para a tarefa de classificação.

Também nos parece que quanto maiores os valores de x, mais convincente é a decisão de associar x a uma classe positiva, porque excedemos significativamente o valor limite. Também parece que quanto menores os valores de x, mais plausível é a decisão de juntar x à classe negativa, porque estamos significativamente abaixo do valor limite. Para valores de x, que estão em torno de zero, esses argumentos são mais fracos. Portanto, a função sigmoide também pode ser associada à interpretação da probabilidade de pertencer a uma classe.

Se ligarmos a função sigmoide e a equação do modelo linear, obtemos a equação do modelo de regressão logística, que em geral é

$$y = \sigma(X_1, X_2, \dots, X_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_n X_n)}}.$$

Os argumentos $X_1, X_2, \dots, X_n$ denotam atributos no conjunto de dados, enquanto seus valores uu variam de 0 a 1 e, como vimos, fazem sentido para a tarefa de classificação. A esta equação podemos ainda acrescentar a seguinte interpretação geométrica: os dados são classificados abaixo ou acima da reta que é determinada pela equação de relação linear que inicialmente imaginámos.

Se temos exatamente um atributo, o "certo" a que nos referimos é o real. Se tivermos exatamente dois atributos, o "certo" é realmente plano no espaço. Se tivermos mais de dois atributos, eles são "verdadeiros", em linguagem matemática, hiperplano.

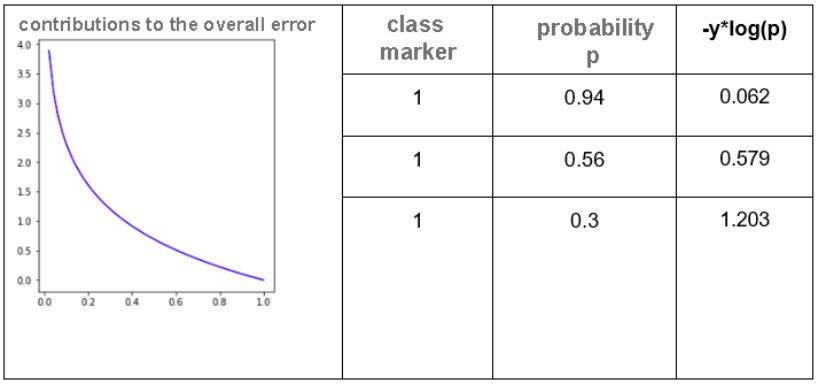
Entropia cruzada

A função de erro que carateriza a regressão logística é chamada de **entropia cruzada** . Vamos primeiro conhecer a intuição que está por trás dessa função, e depois vamos conhecer sua forma matemática.

Dissemos que interpretamos o valor calculado pelo modelo de regressão logística como a probabilidade de pertencer a uma das classes, e que nos guiamos pela regra de que, se esse valor exceder o limiar de 0,5, interpretamo-lo como pertencente a uma classe positiva, e se este valor for inferior a 0,5, interpretamo-lo como pertencendo a uma classe negativa. Se o valor de probabilidade for 0,5, é interpretado como pertencendo a uma classe positiva.

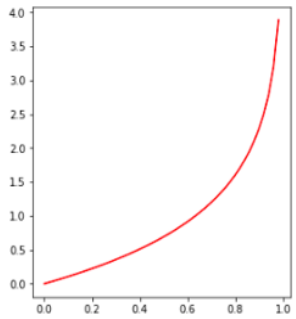
Calculamos a função de erro no conjunto de treino. Nele, sabemos para cada instância quais são as características exatas, então podemos sempre compará-las com as características que ele calculou, ou seja, entramos no modelo.

Suponhamos que, para três casos pertencentes a uma classe positiva, o modelo de regressão logística determinou os valores 0,94, 0,56 e 0,3, respectivamente. No primeiro caso, o valor está próximo da unidade, por isso indica uma determinada decisão do modelo. No segundo caso, este valor é menor e mais próximo do limiar de classificação, mas suficiente para uma boa decisão do modelo. No terceiro caso, o valor está abaixo do limite e faria com que o modelo cometesse um erro. Ao projetar a função de erro, queremos penalizar os cálculos do modelo que para positivo o valor é superior a 1, ou seja, fazer suas contribuições para o erro geral do modelo maior. Uma dessas funções que satisfaz a propriedade requirida é $-log(x)$, cujo gráfico é mostrado na figura abaixo. Precisamos de um sinal de menos para o erro para obter um valor positivo porque o logaritmo é negativo para os valores do argumento da função que são de 0 a 1. No gráfico também podemos ver que os valores da função são pequenos para argumentos mais próximos de 1, ou seja, que os valores da função são maiores para argumentos que estão mais próximos de zero. Então, agora, em ordem, as contribuições para o erro total das instâncias extraídas serão $-log(0.94) = 0.062, -log(0.56) = 0.579, -log(0.3) = 1.203$ e exatamente a proporção de tamanho que queríamos. Também podemos registrá-los em uma tabela, da maneira que fizemos e na tarefa de regressão linear. Na primeira coluna, colocaremos o marcador de classe (o valor exato), na segunda coluna a probabilidade p calculada pelo modelo, e na terceira coluna inserimos o valor $-log(p)$. Observe que o nome da coluna diz $-y * log(p)$, mas como $y = 1$ isso é o mesmo que $-log(p)$.



class marker	probability p	-y*log(p)
1	0.94	0.062
1	0.56	0.579
1	0.3	1.203

Vamos agora selecionar três instâncias da classe negativa e discutir as expectativas que temos da função de erro em seu caso. Que as probabilidades, respectivamente, calculadas pelo modelo de regressão logística sejam 0,03,0,48 e 0,74. Agora, no primeiro caso, o valor do modelo é próximo de zero, então indica uma certa decisão de pertencer à classe negativa. No segundo caso, este valor está próximo do limiar de classificação, mas está abaixo dele, pelo que, mais uma vez, é suficiente que o modelo decida sobre uma classe negativa. No caso da terceira instância, o valor de probabilidade está acima do limite, então o modelo errará e classificará a instância como positiva. O que esperamos da função de erro para instâncias negativas é que sua parcela do erro total seja a mais alta possível quanto mais longe estiverem de zero. Uma dessas funções que satisfaz essa propriedade é $-log(1-p)$. Como y = 0 para todas as instâncias, o símbolo no nome da coluna $-(1-y) * (1-p)$ não muda nada.

contributions to the overall error	class marker	probability p	-(1-y)*log(1-p)
	0	0.03	0.030
	0	0.48	0.654
	0	0.74	1.347

O valor total das funções de entropia cruzada é obtido quando as contribuições de erro de todas as instâncias positivas e todas negativas são somadas (semelhante ao que fizemos na regressão linear e no problema do erro médio-quadrado). Isto está escrito no forma

$$-\sum_{i=1}^{N} \left(y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)\right)$$

onde o primeiro fator soma as contribuições dos erros das instâncias positivas, e o segundo fator contribui com os erros das instâncias negativas. O valor de yi é a característica exata da classe no conjunto de treino, e pi é a probabilidade calculada pelo modelo de regressão logística. Este erro é chamado **de entropia cruzada binária**.

Os valores de parâmetros β desconhecidos no modelo de regressão logística são encontrados selecionando o valor do parâmetro para o qual a função de erro cruzado é a menor. A técnica de descida de gradiente também pode nos ajudar neste caso.

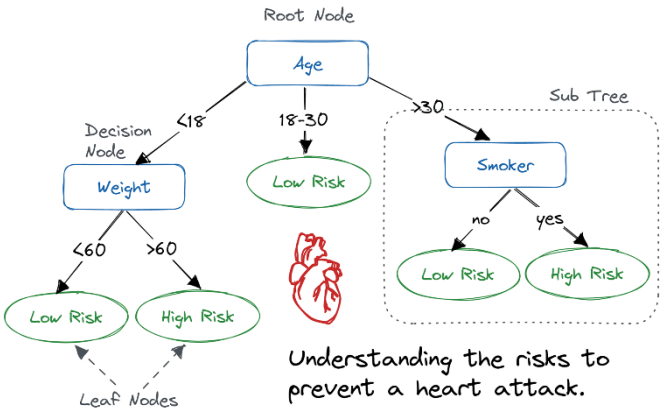
Agora vamos conhecer um algoritmo de classificação um pouco diferente.

3.7 Árvore de decisão

As Árvores de Decisão são algoritmos versáteis de Machine Learning que podem executar tarefas de classificação e regressão, e até mesmo tarefas de multioutput. São algoritmos muito poderosos, capazes de encaixar conjuntos de dados complexos.

Uma árvore de decisão é uma estrutura de árvore semelhante a um fluxograma onde um nó interno representa um recurso (ou atributo), o ramo representa uma regra de decisão e cada nó folha representa o resultado.

O nó mais alto em uma árvore de decisão é conhecido como nó raiz. Ele aprende a particionar com base no valor do atributo. Ele particiona a árvore de uma maneira recursiva chamada particionamento recursivo. Esta estrutura semelhante a um fluxograma ajuda-o na tomada de decisões. É a visualização como um diagrama de fluxograma que imita facilmente o pensamento a nível humano. É por isso que as árvores de decisão são fáceis de entender e interpretar.



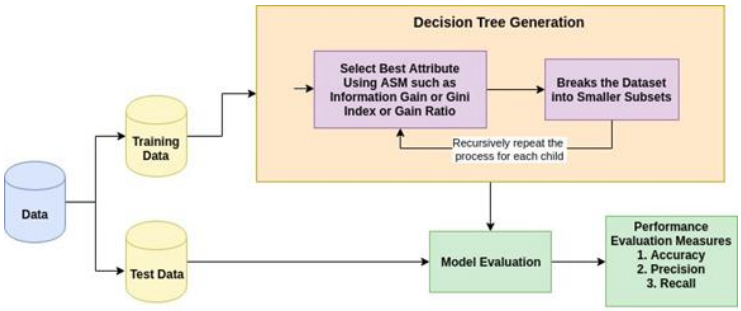
Uma árvore de decisão é um tipo de caixa branca do algoritmo de ML. Ele partilha a lógica interna de tomada de decisão, que não está disponível no tipo de caixa preta de algoritmos, como com uma rede neural. Seu tempo de treino é mais rápido em comparação com o algoritmo de rede neural.

A complexidade temporal das árvores de decisão é função do número de registos e atributos nos dados fornecidos. A árvore de decisão é um método livre de distribuição ou não paramétrico que não depende de pressupostos de distribuição de probabilidade. As árvores de decisão podem lidar com dados de alta dimensão com boa precisão.

Como funciona o algoritmo da árvore de decisão?

A ideia básica por trás de qualquer algoritmo de árvore de decisão é a seguinte:

1. Selecione o melhor atributo usando as Medidas de Seleção de Atributos (ASM) para dividir os registros.
2. Faça desse atributo um nó de decisão e divida o conjunto de dados em subconjuntos menores.
3. Inicie a construção da árvore repetindo este processo recursivamente para cada criança até que uma das condições corresponda:
    - Todas as tuplas pertencem ao mesmo valor de atributo.
    - Não há mais atributos restantes.
    - Não há mais casos.



Medidas de seleção de atributos

A medida de seleção de atributos é uma heurística para selecionar o critério de divisão que particiona os dados da melhor maneira possível. Também é conhecido como regras de divisão porque nos ajuda a determinar pontos de interrupção para tuplas em um determinado nó. O ASM fornece uma classificação para cada recurso (ou atributo) explicando o conjunto de dados fornecido. O melhor atributo de pontuação será selecionado como um atributo de divisão. No caso de um atributo de valor contínuo, os pontos de divisão para ramos também precisam ser definidos. As medidas de seleção mais populares são Ganho de Informação, Índice de Ganho e Índice de Gini.

Ganho de informação

Claude Shannon inventou o conceito de entropia, que mede a impureza do conjunto de entrada. Em física e matemática, a entropia é referida como a aleatoriedade ou a impureza em um sistema. Na teoria da informação, refere-se à impureza em um grupo de exemplos. O ganho de informação é a diminuição da entropia. O ganho de informação calcula a diferença entre a entropia antes da divisão e a entropia média após a divisão do conjunto de dados com base em determinados valores de atributo. O algoritmo da árvore de decisão ID3 (Dicotomiser Iterativo) usa ganho de informação.

$$Info(D) = -\sum_{i=1}^{m} pi \cdot log_2 pi$$

Onde Pi é a probabilidade de uma tupla arbitrária em D pertencer à classe Ci.

$$Info_A(D) = \sum_{j=1}^{V} \frac{Dj}{D} XInfo(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Onde:

- Info(D) é a quantidade média de informação necessária para identificar o rótulo de classe de uma tupla em D.
- |Dj|/|D| atua como o peso da partição jth.
- InfoA(D) é a informação esperada necessária para classificar uma tupla de D com base no particionamento por A.

O atributo A com o maior ganho de informação, Gain(A), é escolhido como o atributo de divisão no nó N().

Índice de ganho

O ganho de informação é tendencioso para o atributo com muitos resultados. Isso significa que ele prefere o atributo com um grande número de valores distintos. Por exemplo, considere um atributo com um identificador exclusivo, como customer_ID, que tenha zero info(D) devido à partição pura. Isso maximiza o ganho de informações e cria particionamentos inúteis.

C4.5, uma melhoria do ID3, usa uma extensão para o ganho de informação conhecida como a taxa de ganho. A taxa de ganho lida com a questão do viés normalizando o ganho de informações usando Split Info. A implementação Java do algoritmo C4.5 é conhecida como J48, que está disponível na ferramenta de mineração de dados WEKA.

$$Info_A(D) = \sum_{j=1}^{V} \frac{Dj}{D} \times \left(\frac{|D_j|}{|D|}\right)$$

Onde:

- $\frac{|D_j|}{|D|}$ atua como o peso da j-ésima partição.
- v é o número de valores discretos no atributo A.
- A taxa de ganho pode ser definida como

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

O atributo com a maior taxa de ganho é escolhido como o atributo de divisão (Source).

Índice de Gini

Outro algoritmo de árvore de decisão CART (Classification and Regression Tree) usa o método Gini para criar pontos de divisão.

$$Gini(D) = 1 - \sum_{i=1}^{m} Pi^2$$

Onde pi é a probabilidade de uma tupla em D pertencer à classe Ci.

O Índice de Gini considera uma divisão binária para cada atributo. Pode calcular uma soma ponderada da impureza de cada partição. Se uma divisão binária no atributo A particiona os dados D em D1 e D2, o índice de Gini de D é:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

No caso de um atributo de valor discreto, o subconjunto que fornece o índice gini mínimo para o escolhido é selecionado como um atributo de divisão. No caso de atributos de valor contínuo, a estratégia é selecionar cada par de valores adjacentes como um possível ponto de divisão, e um ponto com um índice de Gini menor é escolhido como o ponto de divisão.

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

O atributo com o índice de Gini mínimo é escolhido como o atributo de divisão.

Edifício do Classificador de Árvore de Decisão em Scikit-learn

Importando bibliotecas necessárias

Vamos primeiro carregar as bibliotecas necessárias.

```
# Load libraries

import pandas as pd

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
```

```
from sklearn.model_selection import train_test_split # Import train_test_split function

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

Carregando dados

Vamos primeiro carregar o conjunto de dados Pima Indian Diabetes necessário usando a função CSV de leitura dos pandas. Pode baixar o Kaggle data set para acompanhar.

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age',
'label']

# load dataset

pima = pd.read_csv("diabetes.csv", header=None, names=col_names)

pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Seleção de recursos

Para entender o desempenho do modelo, dividir o conjunto de dados em um conjunto de treino e um conjunto de testes é uma boa estratégia.

Vamos dividir o conjunto de dados usando a função train_test_split(). É necessário passar três parâmetros: características (features), alvo (target) e tamanho do conjunto de teste.

```
# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# 70% training and 30% test
```

Construção do modelo de Árvore de Decisão

```
# Create Decision Tree classifer object

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
```

```
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset

y_pred = clf.predict(X_test)
```

Avaliação do Modelo

Vamos estimar com que precisão o classificador ou modelo pode prever o tipo de cultivares.

A precisão pode ser calculada comparando os valores reais do conjunto de testes e os valores previstos.

```
# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.6753246753246753

Obtivemos uma taxa de classificação de 67,53%, o que é considerado uma boa precisão. Pode melhorar essa precisão ajustando os parâmetros no algoritmo da árvore de decisão.

Visualizando árvores de decisão

Pode usar a função export_graphviz do Scikit-learn para exibir a árvore dentro de um caderno Jupyter. Para plotar a árvore, também precisa instalar graphviz e pydotplus.

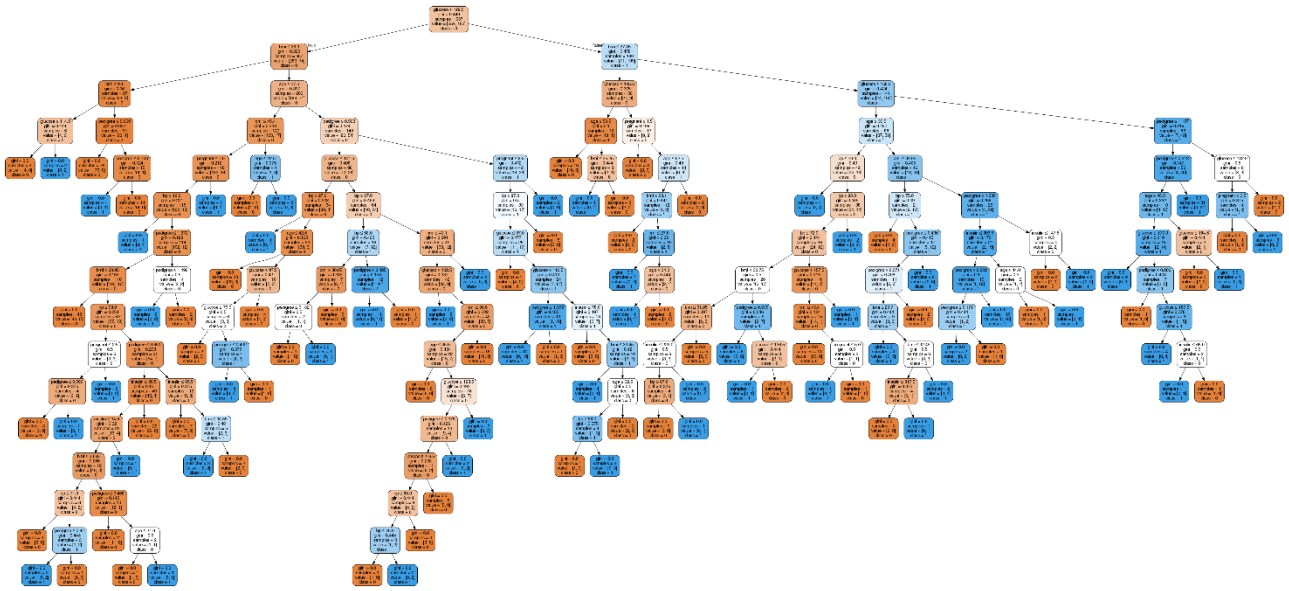
pip install graphviz

pip install pydotplus

A função *export_graphviz* converte o classificador da árvore de decisão em um arquivo de pontos, e pydotplus converte esse arquivo de pontos em png ou forma exibível no Jupyter.

```
from sklearn.tree import export_graphviz

from sklearn.externals.six import StringIO

from IPython.display import Image

import pydotplus


dot_data = StringIO()

export_graphviz(clf, out_file=dot_data,

                filled=True, rounded=True,

                special_characters=True,feature_names                                    =
feature_cols,class_names=['0','1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('diabetes.png')
```

```
Image(graph.create_png())
```



No gráfico da árvore de decisão, cada nó interno tem uma regra de decisão que divide os dados. Gini, referido como razão de Gini, mede a impureza do nó. Pode dizer que um nó é puro quando todos os seus registros pertencem à mesma classe, como nós conhecidos como nó folha.

Aqui, a árvore resultante não é podada. Esta árvore não podada é inexplicável e não é fácil de entender. Na próxima secção, vamos otimizá-lo por poda.

Otimizando o desempenho da árvore de decisão

- **critério: opcional (default="gini") ou Escolha a medida de seleção de atributos.** Este parâmetro nos permite usar a medida de seleção de atributos diferentes. Os critérios suportados são "gini" para o índice de Gini e "entropia" para o ganho de informação.
- **splitter : string, opcional (default="best") ou Split Strategy.** Este parâmetro permite-nos escolher a estratégia de divisão. As estratégias suportadas são "melhores" para escolher a melhor divisão e "aleatórias" para escolher a melhor divisão aleatória.
- **max_depth : int ou None, opcional (default=None) ou Maximum Depth of a Tree.** A profundidade máxima da árvore. Se Nenhum, então os nós são expandidos até que todas as folhas contenham menos de min_samples_split amostras. O maior valor de profundidade máxima causa sobreajuste, e um valor mais baixo causa subajuste (Source).

No Scikit-learn, otimização do classificador de árvore de decisão realizada apenas por pré-poda. A profundidade máxima da árvore pode ser usada como variável de controle para pré-poda. No exemplo a seguir, pode plotar uma árvore de decisão nos mesmos dados com max_depth=3. Além dos parâmetros de pré-poda, também pode tentar outra medida de seleção de atributos, como entropia.

```
# Create Decision Tree classifer object

clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
```

```
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset

y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.7705627705627706

Bem, a taxa de classificação aumentou para 77,05%, o que é melhor precisão do que o modelo anterior.

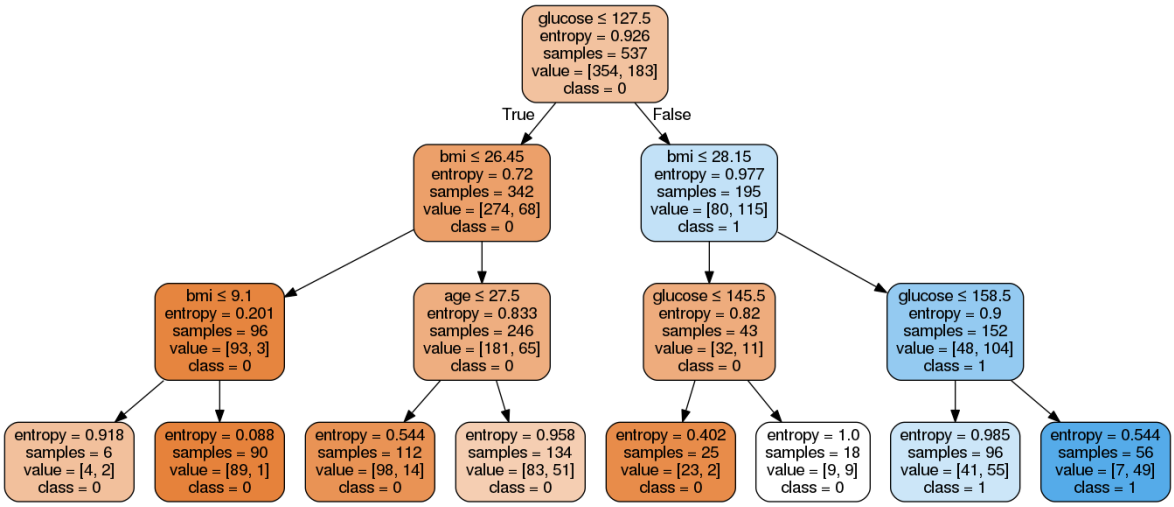
Visualizando árvores de decisão

Vamos tornar nossa árvore de decisão um pouco mais fácil de entender usando o seguinte código:

```
from six import StringIO from IPython.display import Image

from sklearn.tree import export_graphviz

import pydotplus

dot_data = StringIO()

export_graphviz(clf, out_file=dot_data,

                filled=True, rounded=True,

                special_characters=True,                    feature_names                    =
feature_cols,class_names=['0','1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('diabetes.png')

Image(graph.create_png())
```

Aqui, concluímos as seguintes etapas:

- Importadas as bibliotecas necessárias.
- Criado um objeto `StringIO` chamado `dot_data` para manter a representação de texto da árvore de decisão.
- Exportou a árvore de decisão para o formato de `dot` (pontos) usando a função `export_graphviz` e gravou o output no buffer de `dot_data`.
- Criado um objeto de gráfico `pydotplus` a partir da representação em formato de `dot` (pontos) da árvore de decisão armazenada no buffer de `dot_data`.
- Gravou o gráfico gerado em um arquivo PNG chamado "diabetes.png".
- Exibida a imagem PNG gerada da árvore de decisão usando o objeto Image do módulo `IPython.display`.



As you can see, this pruned model is less complex, more explicable, and easier to understand than the previous decision tree model plot.

Como pode ver, este modelo podado é menos complexo, mais explicável e mais fácil de entender do que o gráfico de modelo de árvore de decisão anterior.

Prós da árvore de decisão

- As árvores de decisão são fáceis de interpretar e visualizar.
- Ele pode facilmente capturar padrões não lineares.
- Requer menos pré-processamento de dados do usuário, por exemplo, não há necessidade de normalizar colunas.
- Ele pode ser usado para engenharia de recursos, como previsão de valores ausentes, adequado para seleção de variáveis.
- A árvore de decisão não tem suposições sobre a distribuição devido à natureza não paramétrica do algoritmo. (Fonte)

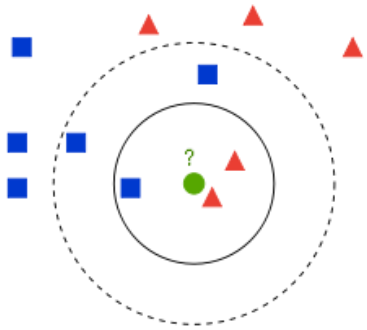
Contras da árvore de decisão

- Sensível a "dados barulhentos". Pode sobreajustar dados barulhentos.
- A pequena variação (ou variância) nos dados pode resultar na árvore de decisão diferente. Isso pode ser reduzido por ensacamento e impulsionamento de algoritmos.
- As árvores de decisão são tendenciosas com o conjunto de dados de desequilíbrio, por isso é recomendável equilibrar o conjunto de dados antes de criar a árvore de decisão.

3.8 Algoritmo dos K Vizinhos Mais Próximos (kNN)

Existem também modelos não paramétricos de machine learning. O modelo obtido usando o algoritmo do vizinho k-mais próximo é exatamente assim. Vamos descobrir como funciona!

Deixe nosso conjunto de treino consistir em pares de números (x_1, x_2) e os nomes de classe correspondentes. Os pares podem ser representados como pontos no plano, onde a primeira coordenada x_1 denota o valor no eixo x e a segunda coordenada x_2 denota o valor no eixo y. Na prática, os valores de x_1 e x_2 estão sempre associados a alguns atributos específicos, por exemplo, temperatura e humidade, mas agora podemos pensar neles como alguns valores gerais. Cada par de números pertence a uma de duas classes: triângulos vermelhos ou quadrados azuis. Como existem apenas duas classes, pode assumir-se que é uma classificação binária. Agora imagine que o círculo verde representa uma nova instância, um novo par de números, para o qual precisamos determinar a qual classe ele pertence: se é um triângulo vermelho ou um quadrado azul.



Kit de treino

O algoritmo de vizinho k-mais próximo é um algoritmo de classificação que diz que primeiro corrigimos o número de vizinhos (instâncias circundantes) k para um valor específico e, em seguida, determinamos quantos dos vizinhos k-mais próximos existem vermelhos e azuis: o vizinho vermelho é uma instância pertencente à classe vermelha e o vizinho azul é uma instância pertencente à classe azul. Por exemplo, se fixarmos o número k a 3, os três vizinhos mais próximos do círculo verde estão dentro de um círculo sólido. Existem dois triângulos vermelhos e um quadrado azul.

Além disso, o algoritmo do vizinho k-mais próximo diz que a nova instância, um novo par de pontos é adicionado à classe do vizinho mais numeroso: se os vizinhos vermelhos são mais numerosos, dizemos que a nova instância pertence à classe vermelha e, da mesma forma, se os vizinhos azuis são mais numerosos, dizemos que a nova instância pertence à classe azul. Também pode pensar nisso como o ditado "com quem você está, é como você é" no mundo da machine learning.

No nosso exemplo, quando o valor de k é fixado em 3, concluímos que devemos associar o círculo verde à classe vermelha porque temos dois vizinhos vermelhos e um vizinho azul.

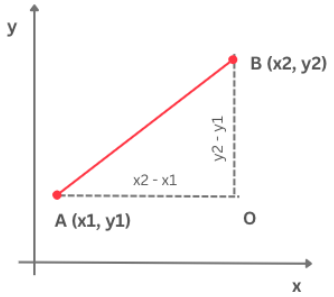
Vamos ver o que acontece se corrigirmos o número k em 5. Na figura é mostrado por um círculo tracejado. Uma vez que existem agora três quadrados azuis e dois triângulos vermelhos, a conclusão seria que o círculo verde deveria ser unido à classe azul.

Esta secção é emparelhada com o Exercise 8 e o Jupyter Notebook 08-k-nearest_neighbors.ipynb. TPara acompanhar o conteúdo, clique no link e, em seguida, no botão Open in Colab para abrir o conteúdo no *Google Collab*. Se estiver visualizando os blocos de anotações em sua máquina local, localize o bloco de anotações com o mesmo nome entre os conteúdos e execute-o.

O material que o acompanha contém o conjunto de pontos acima mencionado e uma aplicação na qual pode examinar o que acontecerá se escolher um valor diferente do número k. Como o algoritmo precisa decidir quais vizinhos há mais, é sábio escolher valores ímpares do número k.

Note que além do número de vizinhos k, o resultado do algoritmo também depende de como medimos as distâncias para os vizinhos! Para encontrar os vizinhos mais próximos, precisamos de alguma forma medir a distância até eles.

Até agora, encontramos uma distância chamada distância euclidiana. A distância euclidiana entre os pontos A e B é calculada como o comprimento dos comprimentos que ligam os pontos A e B. Por exemplo, para os pontos A =(0,0) i B =(3,4) a distância euclidiana é calculada como $\sqrt{(3-0)^2+(4-0)^2}=5$.



Distância Euclideana

Há muitas outras distâncias também. Por exemplo, pode ficar intrigado com a distância de Manhattan. Ao contrário da distância euclidiana, que calcula a "hipotenusa" de um triângulo definido pelos pontos A, B e O (se seguirmos a figura anterior), a distância de Manhattan calcula a soma das "pernas" deste triângulo. Para os pontos A e B, o valor da distância de Manhattan seria $|3-0|+|4-0|=7$.

A distância que escolhemos depende da natureza da tarefa e do significado que os atributos com os quais estamos trabalhando têm. Em geral, podemos tentar mais distâncias e escolher aquela para a qual obtemos os melhores resultados. Falaremos sobre isso mais adiante. É importante notar que uma função deve satisfazer certas propriedades matemáticas para ser declarada uma distância, portanto, nem todas as funções podem ser úteis.

Assim como outros algoritmos de machine learning, o algoritmo do vizinho mais próximo x é treinado sobre o conjunto de treino. É interessante notar que a fase de aprendizagem neste algoritmo é realmente reduzida ao armazenamento apenas do conjunto de dados. Em outros algoritmos, como a regressão linear ou a regressão logística, vimos que, nesta fase, os valores de alguns parâmetros que aparecem no modelo são calculados procurando a função de erro mínimo. O algoritmo do vizinho mais próximo de x não é assim. O mapeamento que aprendemos Não é sobre uma função específica, é sobre os dados em si e as etapas que precisam ser tomadas. É por isso que é comum chamar modelos que possuem essa propriedade **de modelos não paramétricos**.

O algoritmo dos K Vizinhos Mais Próximos executa todo o trabalho durante a aplicação, ou seja, decide a qual classe a nova instância pertence. Quando precisamos classificar uma nova instância, primeiro calculamos a distância da nova instância de todas as instâncias no conjunto de dados de treino. Em seguida, classificamos essas distâncias do menor para o maior. Mantemos as primeiras distâncias k (porque são as distâncias para k vizinhos mais próximos) e escolhemos instâncias do conjunto de treino a que se referem. Continuamos a monitorizar o que está a acontecer no espaço dos seus marcos e procuramos os marcos mais numerosos, ou seja, a maior classe. Como vimos no exemplo introdutório, a nova instância deve ser associada à classe que é mais numerosa.

Este algoritmo é fácil de implementar, por isso vamos arregaçar as mangas e começar!

Vamos imaginar que estamos trabalhando com um conjunto de dados que usamos até agora e que cada instância tem um formulário (x1, x2), onde mark é o valor 0 para vermelho ou 1 para azul.

Para medir a distância entre instâncias, usaremos a função euklidsko_rastojanje, que é definida pelo seguinte bloco de código:

```
def euclidean_distance(instance1, instance2):
  return np.sqrt((instance1[0]-instance2[0])**2 + (instance1[1]-instance2[1])**2)
```

O algoritmo do x vizinho mais próximo em si é representado pelo seguinte bloco de código:

```
def kNN(k, instances, new_instance, classes={0: 'red', 1: 'blue'}):
# first, calculate the distances between the new instance and all instances in the dataset
  distances = [euclidean_distance(instance, new_instance) for instance in instances]
  # then sort the distances, extract the k smallest ones and the corresponding instances
  # declare them as neighbors
  neighbors = np.argsort(distances)[0:k]
  # then read the labels of the neighbors and count them
  neighbor_labels = [instances[neighbor][2] for neighbor in neighbors]
  label_counts = np.bincount(neighbor_labels)
  # the label of the new instance will be the label of the most frequent neighbor
  label = np.argmax(label_counts)
  return classes[label]
```

Nele, como já discutimos, realizamos as seguintes etapas:

1. Calculamos a distância da nova instância para todas as instâncias no conjunto de dados.
2. E então nós os colocamos no meio da noite, e então os colocamos no escuro.
3. E somos nós que temos o direito de ser os vizinhos.
4. No conjunto de vizinhos isolados, contamos os mais numerosos,
5. Concluímos que a nova instância pertence à classe do vizinho mais numeroso.

Ainda temos que aprender a escolher o melhor valor do número k. Falaremos sobre isso na próxima secção.

3.9 Hiperparâmetros

Vimos que no algoritmo do x vizinho mais próximo é necessário fixar o valor do número k com antecedência, e que escolhas diferentes levam a conclusões diferentes. Como sabemos qual valor escolher? Esta pergunta segue todos os outros algoritmos de machine learning em que aparecem alguns valores que precisamos de definir antecipadamente. Tais valores são chamados **de hiperparâmetros** ou **metaparâmetros** .

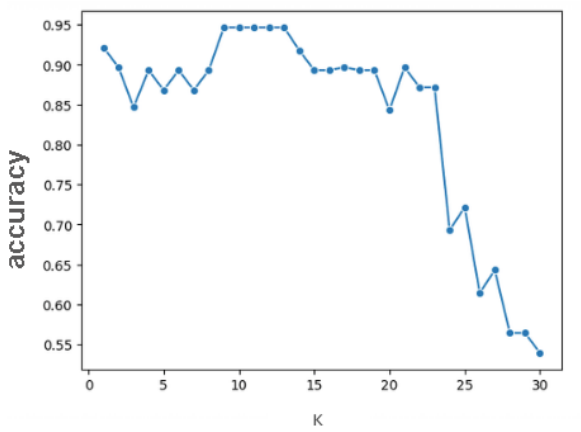
Mencionamos que, ao dividir um conjunto de dados, sempre destacamos um conjunto de treino, um conjunto de testes e um conjunto de validação. Não usamos o conjunto de validação até agora. Na verdade, precisamos dele sempre que há alguns hiperparâmetros no nosso algoritmo de aprendizagem cujo melhor

valor precisamos determinar. A história que vamos partilhar aplica-se a todos os algoritmos, mas vamos continuar a usar o algoritmo do k vizinho mais próximo.

Vamos voltar à questão de como escolher o melhor valor do hiperparâmetro k. É natural pensar: vamos tentar vários valores, por exemplo, todos os números de 1 a 10, e então vamos escolher o melhor valor! Vamos fazer isso, mas vamos ter muito cuidado sobre onde tentamos o quão boa é a nossa escolha. Se fizermos isso num conjunto de testes, estaremos quebrando a regra de ouro da machine learning sobre a separação estrita entre o conjunto de testes e o desenvolvimento do modelo: usaremos o conjunto de testes para decidir qual é o melhor valor do hiperparâmetro k e, então, quando treinarmos o modelo, usaremos novamente o conjunto de testes para avaliar o quão bom ele é! Concordará que não faz muito sentido!

É correto fazer o seguinte: testaremos quais valores de hiperparâmetros são melhores no conjunto de validação. Este conjunto não partilha informação nem com o conjunto de treino nem com o conjunto de testes, pelo que contribuirá para a objetividade das nossas conclusões. Agora que estabelecemos isso, podemos começar a trabalhar determinando o melhor valor do hiperparâmetro k.

Para cada um dos valores do hiperparâmetro k que queremos testar, treinaremos separadamente o modelo no conjunto de treino e calcularemos sua medida de qualidade no conjunto de validação. Neste caso, é preciso. Os valores obtidos podem ser exibidos graficamente colocando diferentes valores do parâmetro k ao longo do eixo x e valores de precisão ao longo do eixo y. O valor do hiperparâmetro k para o qual obtemos o melhor valor da medida de qualidade no conjunto de validação é o valor do hiperparâmetro que estamos procurando. Isso geralmente é visto no gráfico como a região onde os valores são mais altos.



Demonstrar a precisão do modelo no conjunto de validação

Com base no gráfico anterior, podemos ver que os valores ótimos do hiperparâmetro k são, na verdade, 9, 10, 11, 12 e 13, porque todos eles resultam na mesma e mais alta precisão do modelo.

Gráficos semelhantes podem ser desenhados para valores de hiperparâmetros e funções de erro. Em seguida, definimos valores diferentes do hiperparâmetro ao longo do eixo x e os valores da função de erro ao longo do eixo y. Agora é importante observar os valores do hiperparâmetro para o qual a função de erro é menor.

Quando vários hiperparâmetros estão presentes em um algoritmo de aprendizagem, o objetivo é encontrar a melhor combinação de hiperparâmetros. Também o determinamos com base no conjunto de validação, rastreando o sucesso do modelo e procurando a combinação que dá o melhor valor da medida de qualidade (ou, igualmente, rastreando o erro do modelo e procurando a combinação que dá o menor valor de erro). O problema é que este processo pode ser bastante lento e computacionalmente exigente para um grande número de hiperparâmetros: por exemplo, se quisermos examinar 10 valores diferentes de k e 3 funções de

distância diferentes, na verdade temos 10 x 3 = 30 combinações diferentes, então temos que treinar e avaliar 30 modelos diferentes.

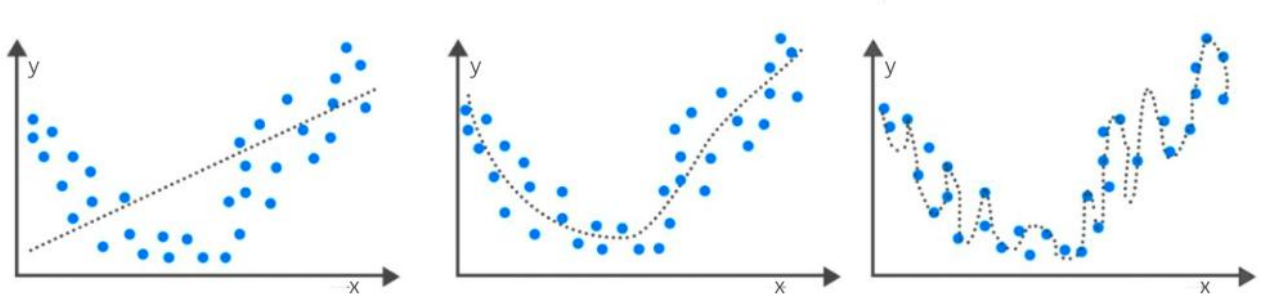
3.10 Generalização, subadaptação e sobreadaptação

Nesta secção, vamos aprender sobre os conceitos de generalização, superadaptação e subajuste que são frequentemente encontrados na história de machine learning.

Imagine que Mike, Ana e Luka estão a preparar-se para um exame de matemática e que todos usam a mesma coleção. Mike estava relaxado e apenas praticando as tarefas apenas ligeiramente, Ana foi diligente e praticou de forma cuidadosa e abrangente durante toda a semana, enquanto Luka decidiu memorizar as tarefas. Consegue adivinhar quem se saiu melhor no teste? Claro, Ana!

Uma coleção de problemas pode ser pensada como um conjunto abstrato de dados que consiste em entradas (textos de tarefas) e saídas (soluções). Um modelo de machine learning pode, como Pera, aprender apenas algumas conexões nos dados e cometer muitos erros na prática. Tal propriedade do modelo é chamada de **sub ajuste**. O modelo também pode exagerar no nível de detalhes, como Luke, e perder o poder de lidar com alguns dados novos. Tal propriedade do modelo é chamada de **sobreajuste**. Seria melhor que o modelo adotasse as informações certas e pudesse, como Anna, resolver com sucesso tarefas familiares e algumas novas. Essa propriedade do modelo é chamada de **generalização**.

Um exemplo de subadaptação e sobreadaptação pode ser ilustrado pela figura seguinte. Imagine que ao longo do eixo x estão os valores de um atributo, ao longo do y-oce estão os valores da variável de destino, e que a linha tracejada mostra o modelo. O modelo à esquerda não é a melhor escolha considerando a disposição dos pontos, parece muito simples. Os dados se parecem mais com um "vidro", então um modelo quadrado que tenha esse formulário pode ser uma solução melhor. Podemos vê-lo na imagem do meio. Na imagem à direita, vemos um modelo que segue consistentemente cada ponto do conjunto de dados e está completamente adaptado a ele.



Exemplo de subadaptação e sobreadaptação

A tarefa de encontrar o modelo ideal e equilibrar entre sub ajustar e sobreajustar não é fácil. Felizmente, o campo da machine learning define os protocolos e técnicas que podemos usar para acompanhar cada uma dessas situações. Por exemplo, grandes diferenças no desempenho do modelo no conjunto de treino e no conjunto de teste indicam que o modelo foi readaptado. Isso geralmente se deve à escolha de um modelo mais complexo do que o necessário (como na imagem no canto superior direito) ou ao treino do modelo por mais tempo. Por outro lado, baixos valores de medidas de qualidade tanto no conjunto de treino quanto no conjunto de testes indicam que o modelo não aprendeu o suficiente com os dados, é muito simples (como na imagem superior esquerda) ou precisa de mais atributos.

Uma boa generalização é uma propriedade que permite que modelos de machine learning sejam aplicados com sucesso na prática. Apenas uma pequena parte dos dados disponíveis é usada para treiná-los e, no

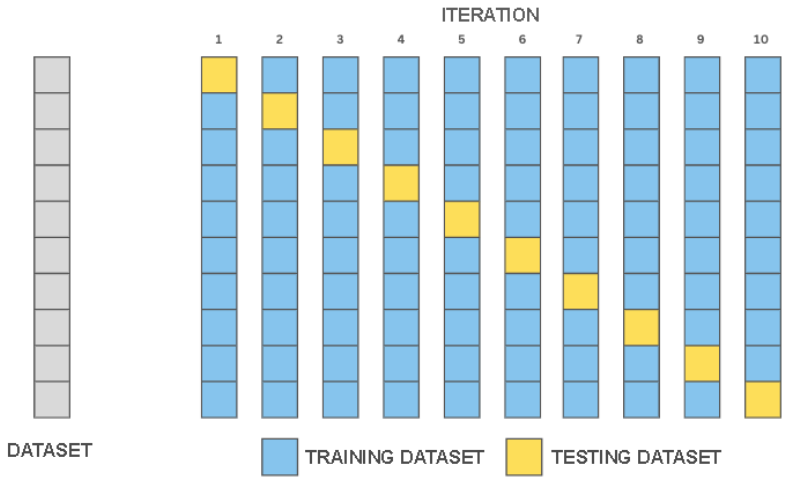
entanto, esperamos que eles se comportem bem durante a aplicação e sobre novos dados que nunca encontraram. É por isso que é importante que os conjuntos de dados sejam representativos, ou seja, sejam suficientemente ricos e diversificados para se adequarem ao problema abordado, bem como um acompanhamento cuidadoso de possíveis subadaptações e sobreadaptações do modelo.

3.11 Validação, validação cruzada

Já dissemos muitas vezes que, antes de aplicar um algoritmo de machine learning, os dados são divididos em um conjunto de treino e um conjunto de testes (incluímos o conjunto de validação quando precisamos). Também mencionamos que o processo de divisão é aleatório. Deve ter-se perguntado se algumas divisões diferentes, em comparação com as que escolhemos, levariam a resultados diferentes do trabalho do modelo. Talvez seja para uma divisão específica do conjunto de dados que obtemos resultados mais otimistas ou drasticamente piores. E é uma espécie de ajuste.

Sempre que os tamanhos dos conjuntos de dados e os algoritmos selecionados permitirem, é desejável realmente executar várias divisões do conjunto de dados inicial em um conjunto de treino e um conjunto de testes para que cada instância no conjunto de dados tenha a oportunidade de ser encontrada em ambos os conjuntos. Um desses procedimentos que descreveremos é chamado **de validação cruzada**. No exemplo, usaremos um algoritmo de regressão linear, mas a história é geral e aplica-se a todos os algoritmos.

Vamos dividir o conjunto de dados em 10 partes, como na figura abaixo. Na primeira etapa, extraímos a primeira parte do conjunto de testes e mantemos as nove partes restantes para treino. Para facilitar o acompanhamento, o conjunto de teste é colorido de amarelo na imagem, e os conjuntos de treino são coloridos de azul. Agora vamos treinar o primeiro modelo de regressão linear no conjunto de treino e calcular o valor de seu erro quadrado médio no conjunto de teste. O valor resultante pode ser marcado com MSE_1 . Na segunda etapa, separamos a segunda parte do conjunto de testes e as nove partes restantes para o conjunto de treino. Agora, na imagem, a segunda parte é pintada de amarelo, e as partes restantes são pintadas de azul. Vamos treinar novamente o modelo de regressão linear no conjunto de treino (que é o segundo modelo agora) e calcular o valor de seu erro quadrado médio no conjunto de teste. Agora vamos marcar esse valor com MSE_2 . Vamos continuar este processo até chegarmos à última, décima, parte: agora vamos mantê-lo como um conjunto de testes, e vamos usar as partes restantes para treinar o modelo. Vamos treinar o décimo modelo de regressão linear sobre ele e, em seguida, calcular o erro quadrado médio MSE_{10} no conjunto de teste.



Validação cruzada com 10 camadas

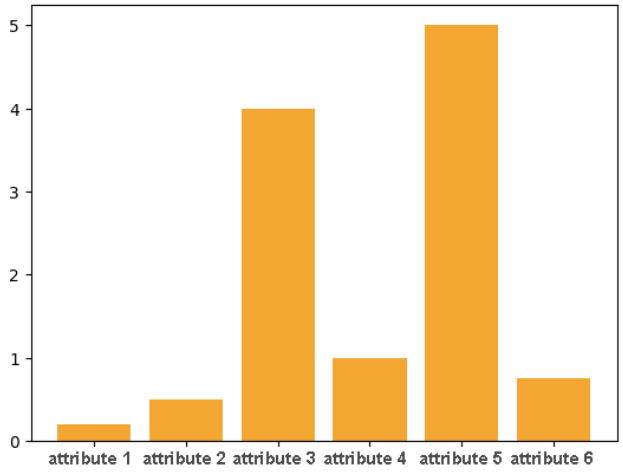
Como temos 10 divisões diferentes do conjunto de dados, também temos 10 valores diferentes do erro quadrado médio. A média dos valores obtidos (MSE_1 + MSE_2 +... + MSE_{10})/10 realmente indica melhor como nosso modelo se comporta e nos ajuda a resolver os dilemas que tivemos no início em relação ao impacto da divisão no sucesso do modelo. O que não está muito claro é qual dos 10 modelos diferentes temos à nossa disposição devemos escolher. O menor erro ou algum outro? Na verdade, devemos agora treinar um novo modelo ao longo de todo o conjunto de dados e continuar a usá-lo, aproximamos o seu comportamento e classificamo-lo com os comportamentos de cada um dos 10 modelos treinados.

Esse processo é chamado de validação cruzada de 10 camadas Validação cruzada de 10 vezes. Na prática, divisões de 3 e 5 camadas também são usadas, e as escolhas dependem do tamanho do conjunto de dados e do tipo de algoritmos usados. Além disso, há uma divisão na qual o número de camadas corresponde ao número de instâncias no conjunto de dados, chamada *leave-one-out cross validation*.

3.12 Regularização

As regularizações são outro conjunto de técnicas que podem ser usadas para controlar reajustes de modelos. O seu principal objetivo é evitar que modelos complexos, que nos ajudam a aprender um conjunto mais rico de dependências em dados, se tornem excessivamente adaptados.

Introduziremos a regularização usando o exemplo de um modelo de regressão linear. Suponhamos que treinamos o modelo e obtivemos os valores dos parâmetros, cuja representação gráfica se parece com a figura.



Os parâmetros que são os maiores (absolutos) em termos de seu valor também são os mais importantes para previsões de modelos. Na figura, estes são os parâmetros que correspondem aos atributos 3 e 5 e os seus valores, como podemos ver, são significativamente superiores aos valores dos outros parâmetros. Nesse sentido, esses atributos podem ignorar o impacto dos atributos restantes nos valores de previsão, para que possamos interpretar esse comportamento do modelo como uma forma de reajuste de dados.

É por isso que é desejável, até certo ponto, limitar os valores dos parâmetros - queremos que o modelo aprenda os parâmetros e que eles reflitam as propriedades dos dados, mas também queremos monitorar seu valor para evitar ajustes excessivos. Essa técnica é chamada de **regularização**. No contexto da regressão linear, podemos fazer isso adicionando a soma dos quadrados dos parâmetros ao erro quadrado médio do modelo:

$$\frac{1}{N}\sum_{i=1}^{N} \left(y_i - (\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)\right)^2 + \lambda(\beta_1^2 + \beta_2^2 + \cdots + \beta_n^2)$$

O valor de λ na expressão é um hiperparâmetro que afeta a força da regularização. Se o seu valor for 0, a regularização não terá efeito. Ao dar alguns valores diferentes de zero, equilibramos a aprendizagem determinada pelo erro quadrado médio e o reajuste medido pelos valores da soma dos quadrados dos parâmetros. Os quadrados existem por razões técnicas, primeiro para evitar que os valores dos coeficientes sejam suprimidos uns contra os outros e, em seguida, para preservar as propriedades da função de erro para a aplicação do algoritmo de otimização. Tal forma estendida de regressão linear complementada por um termo de regularização é chamada de regressão de crista.

Um pouco mais tarde, voltaremos à história da regularização quando introduzirmos as redes neurais. São modelos muito complexos, pelo que muitas vezes podem ser readaptados aos dados. Também veremos como podemos segui-lo.

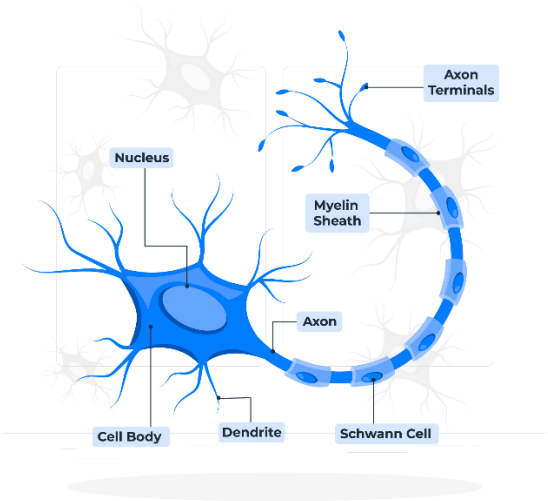
4. Redes Neurais

Bem-vindos ao tema das **Redes Neurais**! As redes neurais são um componente-chave do deep learning, um subconjunto do machine learning que imita a estrutura e a função do cérebro humano. Nesta secção, abordaremos os elementos básicos das redes neurais, incluindo neurónios, camadas e funções de ativação. Exploraremos diferentes tipos de arquiteturas de redes neurais, como redes neurais convolucionais e recorrentes, e suas aplicações na resolução de problemas complexos. Haverá ainda experiências práticas em treino e teste de redes neurais usando ferramentas e estruturas populares.

4.1 Redes Neurais

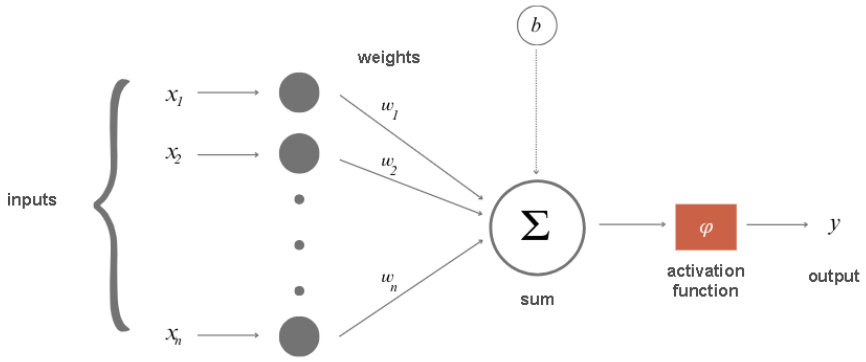
Nesta secção, vamos abordar as redes neurais, um grupo especial de algoritmos de machine learning. A elas devemos muitas descobertas interessantes no mundo da inteligência artificial.

Das aulas de biologia sabe-se que a célula é a unidade básica de estrutura e função de todos os seres vivos. Do ponto de vista da inteligência artificial e da aprendizagem, as mais interessantes são as células cerebrais, chamadas **neurónios**. Os neurónios são compostos por um corpo com um núcleo e extensões mais curtas e mais longas, chamadas **axónios** e **dendrites**. As extensões permitem que os neurónios se liguem a outros neurónios. Estes pontos de ligação são chamados **sinapses**. As sinapses permitem que os sinais — ou seja, impulsos elétricos gerados por um neurónio — sejam transmitidos a outro neurónio. Curiosamente, um único neurónio pode estar ligado a milhões de outros neurónios. Isto significa que recebe e processa sinais provenientes de muitos outros neurónios e, com base nos seus mecanismos internos, ajusta o sinal que envia para os neurónios seguintes. É comum chamar-se a este estado de ativação neural. Dura apenas uma fração de segundo, mas permite realizar cálculos subtis e gera um sinal que é transmitido por todo o sistema nervoso.



O neurónio que encontramos na inteligência artificial é uma abstração matemática dos neurónios do cérebro. É descrito como uma função de várias variáveis $f(x_1, x_2, \ldots, x_n)$, onde cada uma das variáveis $x_1, x_2, \ldots, x_n$corresponde a um único sinal que chega ao neurónio. Como nem todos os sinais são igualmente importantes para a atividade do neurónio, a cada um deles é atribuído um peso $w_1, w_2, \ldots, w_n$ que indica a sua

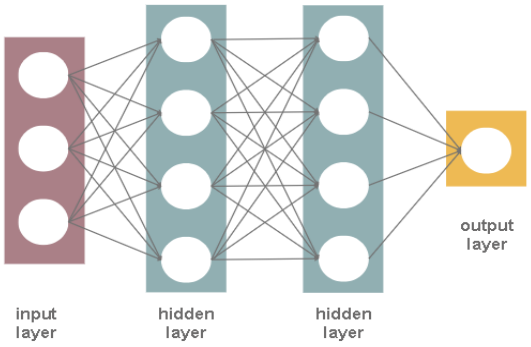
importância. Valores mais elevados destes pesos indicam que o sinal é mais importante, e valores mais baixos indicam o contrário. Assim, a estimulação total dos neurónios corresponde à soma ponderada $w_1x_1 + w_2x_2 + \cdots + w_nx_n$. Para influenciar comportamentos adicionais do neurónio, é adicionado um termo livre b, a esta soma, de forma que a estimulação total do neurónio seja, na realidade $w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$. Este valor é então passado a uma **função de ativação** φ, cuja função é calcular o **output do neurónio**. Dependendo da escolha da função de ativação, os valores dos outputs obtidos também serão diferentes. Se juntarmos tudo de forma sistemática, obtemos que, para os sinais recebidos $x_1, x_2, \ldots, x_n$, o output do neurónio é:$y = \varphi(w\_1x\_1 + w\_2x\_2 + \cdots + w\_nx\_n + b$. Pode também seguir o procedimento que acabámos de descrever na ilustração abaixo.



Abstração matemática dos neurónios

Vamos analisar mais de perto o significado do parâmetro b. Um neurónio natural é caraterizado pelo chamado **limiar de ativação** – se o sinal total recebido pelo neurónio for superior ao valor do limiar de ativação, o neurónio é ativado, processa o sinal e encaminha o resultado desse processamento para outros neurónios. Um papel semelhante, no modelo matemático dos neurónios, é desempenhado pelo parâmetro b. Se o sinal total for superior ao limiar de ativação b, ou seja, se $w1x1 + w2x2 + \cdots + wnxn > b$, o neurónio será ativado. Assim, o parâmetro b permite-nos influenciar comportamentos adicionais dos neurónios. A expressão $w_1x_1 + w_2x_2 + \cdots + w_nx_n > b$ também pode ser escrita como $w_1x_1 + w_2x_2 + \cdots + w_nx_n - b > 0$, e, nesse sentido, o parâmetro b é também uma parte integrante da soma.

Quando os neurónios estão ligados entre si, temos uma **rede neural**. Uma rede neural é geralmente composta por **camadas**, ou seja, grupos de neurónios associados entre si.



Camadas de redes neurais

A camada de input é uma camada que está localizada na entrada de uma rede neural. Os sinais de input x1, x2,..., xn desta camada estão relacionados com os valores dos atributos que temos no conjunto de dados, e assim abordamos a aplicação prática das redes neurais. Por exemplo, se tivermos um conjunto de dados contendo três atributos, temperatura, umidade e pressão atmosférica, a camada de input terá três neurónios: o primeiro corresponderá ao primeiro atributo, temperatura, o segundo corresponderá ao segundo atributo, umidade, e o terceiro neurónio ao terceiro atributo, ou seja, a pressão atmosférica. Para um exemplo particular do conjunto de dados com os valores de temperatura, humidade e pressão atmosférica ascendendo, respetivamente, a 19 °C, 77% e 1011.2 mb, w teremos os valores do sinal x1 = 19, x2 = 77 and x3 = 1011.2. No espírito da história anterior, o primeiro neurónio da camada de entrada recebe e processa apenas o sinal x1 passando-o sem qualquer modificação (isso é possível para a seleção da função de ativação φ(x)= x e os valores w_1 = 1 e b = 0). Os outros dois neurónios e seus sinais x_2 e x_3 também são válidos. Isso significaria que a camada de input nos permite entrar na rede.

A camada de output é uma camada que está localizada na saída de uma rede neural. Como pode imaginar, permite-nos ler os resultados que a rede neural calculou. Dependendo da tarefa a ser resolvida, o número de neurónios nesta camada também dependerá.

Em tarefas de regressão, como esperamos um único valor numérico como resultado (quantidade de precipitação ou algo semelhante), um único neurónio é suficiente. O seu resultado deve corresponder à previsão que esperamos. Para a tarefa de classificação, vamos considerar separadamente a classificação binária e a classificação multiclasse. Como a classificação binária espera dois valores, 0 ou 1, poderás pensar inicialmente que são necessários dois neurónios. No entanto, se pensares bem, verás que até um único neurónio é suficiente: se a sua saída ultrapassar um determinado limiar, um valor pré-definido, podemos considerá-la como um resultado de 1, caso contrário, como um resultado de 0. No caso da classificação multiclasse, podemos ter várias classes, pelo que é prático introduzir um neurónio para cada classe.

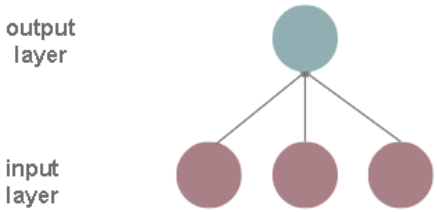
Numa tarefa de classificação multiclasse, esperamos que todas as saídas dos neurónios da camada de saída sejam 0, exceto uma que tenha um valor de 1 - então saberemos exatamente que classe pertence.

As camadas da rede neural que estão localizadas entre as camadas de input e output são chamadas **de camadas ocultas**. Redes neurais que têm mais de uma camada oculta são geralmente designadas por **Redes Neurais Profundas**. É daí que vem o nome **deep learning** . *Deep Learning* é a área de machine learning que os estuda, também conhecida como **Shallow Learning**. *A aprendizagem superficial* é uma forma de aprendizagem.

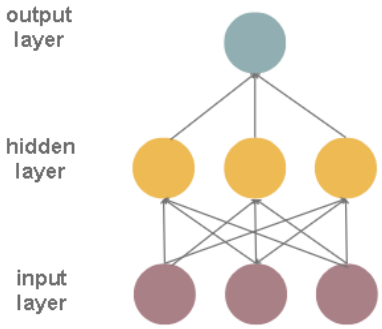
Redes Neurais Totalmente Ligadas são redes nas quais cada neurónio da camada anterior está ligado a cada neurónio da camada seguinte. A imagem que mostra as camadas da rede neural representa também uma rede totalmente ligada, pois todos os neurónios da camada de entrada estão ligados a todos os neurónios da primeira camada oculta; depois, todos os neurónios da primeira camada oculta estão ligados a todos os neurónios da segunda camada oculta e, finalmente, todos os neurónios da segunda camada oculta estão ligados a todos os neurónios (apenas um, na nossa imagem) da camada de saída. As formas pelas quais os neurónios das camadas estão ligados entre si determinam a arquitetura das redes neurais e algumas propriedades específicas das mesmas, que por sua vez, determinam ainda mais em quais áreas elas podem ser utilizadas. Na próxima secção, vamos conhecer algumas desses redes.

Agora vamos considerar o que realmente obtivemos com a introdução de neurónios e redes neurais. Suponhamos que temos três atributos x_1, x_2 *and* x_3. A relação linear entre um atributo e uma variável alvo é matematicamente descrita pela equação $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$. Se, em vez dos parâmetros β, escrevermos w e, em vez de β0, escrevermos b e colocarmos no fim, obtemos, na verdade, a soma ponderada $w_1 x_1 + w_2 x_2 + w_3 x_3 + b$ calculada por um neurónio com base nos sinais que recebe. Isso significa que, se não houvesse função de ativação φ e o neurónio modelariam apenas, uma relação linear entre os atributos (sinais) e os outputs. Isto pode também ser representado graficamente por uma rede que consiste

em apenas uma camada de input com três neurónios e uma camada de output com um neurónio, como na figura abaixo.



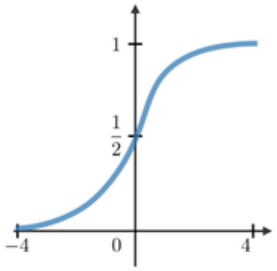
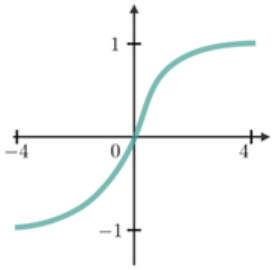
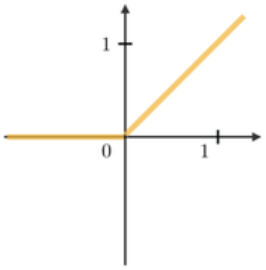
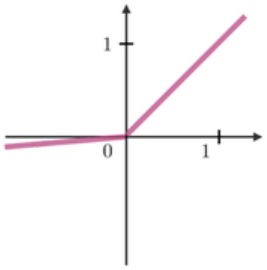
Se a função de ativação não existisse, do ponto de vista da modelação de dependências, faria alguma diferença adicionar uma nova camada escondida? Suponhamos que é uma camada de cor amarela na figura seguinte.



Agora, cada neurónio da camada escondida calcula uma combinação linear dos atributos, e o neurónio da camada de output calcula uma combinação linear dos valores da camada escondida. Isto significa que o neurónio da camada de output está, mais uma vez, a calcular apenas uma combinação linear dos atributos, e que não avançámos muito na representação de relações mais complexas entre atributos e outputs. Além disso, mesmo que adicionássemos 100 camadas escondidas, não sairíamos deste ponto — estaríamos sempre a modelar uma dependência linear.

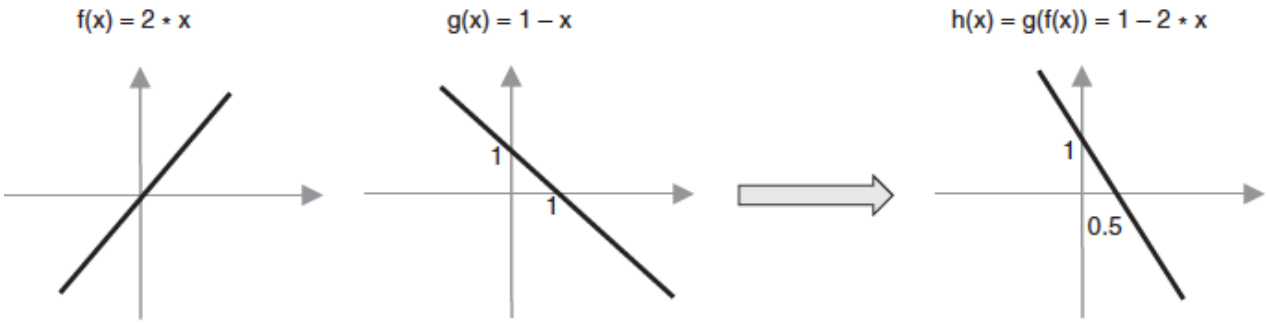
É por isso que a inclusão de uma função de ativação nos cálculos dos neurónios muda significativamente o conjunto de possibilidades que temos. Se usarmos uma função de ativação não linear, conseguimos modelar relações não lineares entre os atributos e a variável alvo. Assim, a existência de uma função de ativação não linear na camada escondida do exemplo anterior permite que o neurónio da camada de output possa agora calcular uma combinação **não linear** dos atributos. Neste contexto, faz muito mais sentido adicionar novas camadas. Ao combinarmos as não linearidades de várias camadas, conseguimos modelar relações complexas entre atributos e saídas.

Para encaixar todas as peças do puzzle, falta discutir quais são as funções de ativação não lineares mais populares em machine learning. São elas: a **função sigmoide**, que conhecemos na história da regressão logística, a **tangente hiperbólica**, a **Rectified Linear Unit (ReLU)** e a **Leaky Rectified Linear Unit (Leaky ReLU)**. As fórmulas pelas quais estas funções são calculadas, bem como os seus gráficos, estão apresentados na figura abaixo. Como se pode ver, estas funções não são de facto lineares — os seus gráficos também não o são.

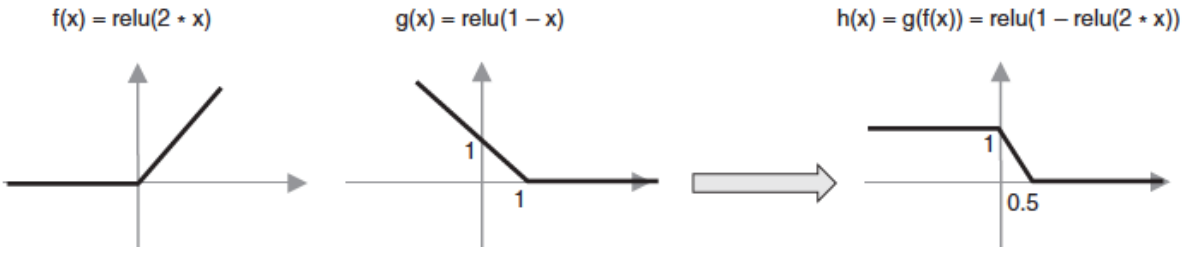
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

As funções de ativação mais comuns

Para completar a explicação sobre a combinação de diferentes funções de ativação, vamos observar as funções f(x)= 2x e g(x)= 1 − x. Podemos ver que ambas são funções lineares de uma variável. Ao combiná-las, a composição das funções, obtemos a função g(f(x))= 1 − 2x, que também é uma função linear de uma variável. Pode-se ver os gráficos de todas as três funções na imagem abaixo.



Para completar a explicação sobre a combinação de diferentes funções de ativação, vamos observar as funções f(x)= ReLU(2x) e g(x)= ReLU(1 − x), que diferem das funções anteriores na medida em que apresentam a função de ativação de uma unidade linear retificada. Portanto, ambas as funções não são lineares. Ao combiná-los, ou seja, ao compô-los, obtemos a função g(f(x))= ReLU(1 − ReLU(2x)), que também é não-linear, e que tem uma nova "forma": permite-nos expressar uma relação ligeiramente diferente entre a variável de input e o output.



A escolha da função de ativação adequada depende da natureza da tarefa e de algumas das propriedades que a rede neural deve apresentar durante o treino. Como isso é feito, iremos explicar na próxima secção.

4.2 Formação de redes neurais

Como vimos, cada neurónio de uma rede neural está ligado, de alguma forma, a outros neurónios da rede. Estas ligações são descritas por pesos w, que representam, na verdade, os parâmetros da rede neural que precisam de ser aprendidos durante o treino. O número de parâmetros numa rede neural é, geralmente, elevado. Suponhamos, por exemplo, que uma rede neural totalmente ligada tem 5 neurónios na camada de input, uma camada escondida com 10 neurónios e uma camada de output com 3 neurónios — o número de parâmetros a aprender seria 93. Na prática, as redes neurais têm milhares, milhões ou até **milhares de milhões** de parâmetros! É por isso que é necessário um grande volume de dados para as treinar.

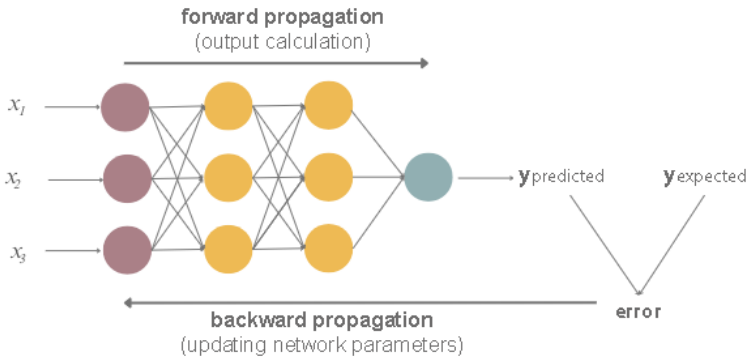
O número de parâmetros numa rede neural totalmente ligada, que tem 5 neurónios na camada de input, uma camada escondida com 10 neurónios e uma camada de output com 3 neurónios, é 93?

A camada de input não contém parâmetros desconhecidos – limita-se a fazer entrar os dados na rede. Cada neurónio da camada escondida está ligado a cada neurónio da camada de input, o que significa que cada uma dessas ligações tem 5 parâmetros e um termo independente (bias). Isto dá um total de 10x5 + 10x1 = 60 parâmetros. Cada neurónio da camada de output está ligado a cada neurónio da camada escondida, o que significa que cada uma dessas ligações tem 10 parâmetros e um termo independente. Isto dá um total de 3x10 + 3x1 = 33 parâmetros. Ao somar ambos os valores, obtemos 93 parâmetros.

As redes neurais, tal como outros modelos, são treinadas com um **conjunto de treino** e avaliadas com um **conjunto de teste**. Como as redes neurais são modelos complexos que conseguem aprender relações complexas entre atributos e saídas, podem ser facilmente **sobre ajustadas** (overfitting) aos dados. É por isso que usamos sempre um **conjunto de validação** durante o treino da rede. Este ajuda-nos a acompanhar o progresso do treino de forma mais precisa e a detetar mais cedo o sobre ajuste e outras propriedades indesejáveis do modelo.

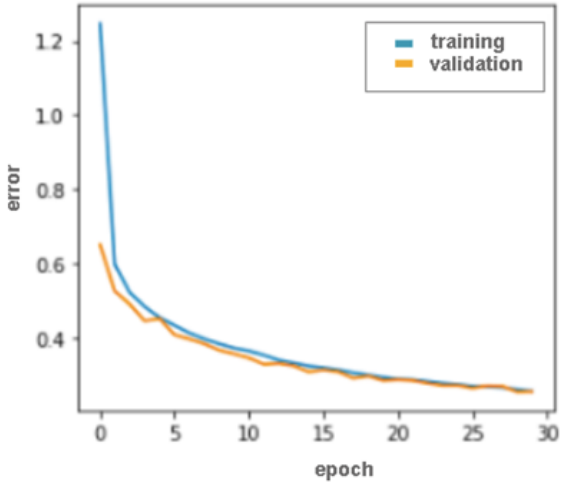
Na introdução do tópico, dissemos que os parâmetros desconhecidos do modelo são determinados através da definição de uma **função de erro**, e depois da aplicação de técnicas de otimização (como o **gradiente descendente**) com o objetivo de encontrar os valores de parâmetros que minimizem essa função de erro. Este procedimento aplica-se também às redes neurais, mas os valores de erro **não são calculados para instâncias individuais**, mas sim para **grupos de instâncias**. A motivação para esta opção é, em primeiro lugar, a necessidade de lidar com grandes quantidades de dados e de **paralelizar e acelerar** todo o processo.

É por isso que todos os dados do conjunto de treino são inicialmente divididos em **pacotes** (*batches*) de tamanho igual. Estes pacotes são então processados pela rede, um a um, e calcula-se o valor da função de erro, comparando os valores esperados com os valores obtidos da variável alvo.Depois, em proporção à sua contribuição, os erros da rede neural são propagados **de trás para a frente na rede** e os valores dos parâmetros são atualizados. Este processo de atualização dos parâmetros da rede é chamado **backpropagation** (propagação para trás) e permite-nos ajustar iterativamente os valores dos parâmetros até encontrar os valores ótimos. Caso contrário, teríamos de começar tudo do zero.



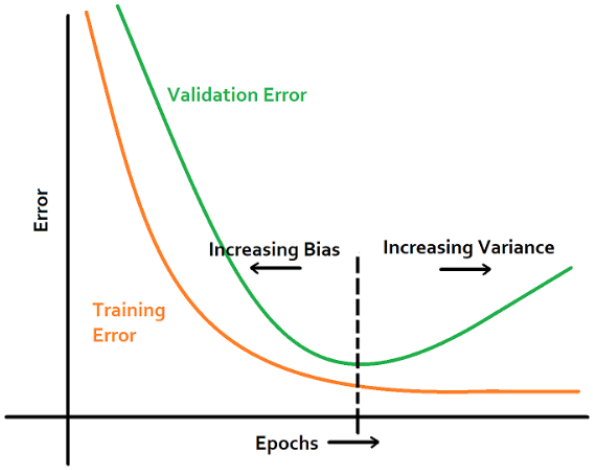
Uma passagem por todo o conjunto de dados, ou seja, um processamento de todos os lotes do conjunto de treino, é chamada de **época** (*epoch*). As redes neurais são treinadas ao longo de várias épocas. Depois de uma época ser concluída, os dados são "baralhados" (*shuffled*), voltam a ser divididos em lotes e passam novamente pela rede. O número de épocas necessário para treinar o modelo depende do sucesso do treino e dos recursos disponíveis. Como se trabalha com grandes volumes de dados, as redes requerem hardware especializado capaz de **paralelizar os cálculos** (por exemplo, **placas gráficas** ou **unidades tensorais**), pelo que o treino de redes é muitas vezes **dispendioso e demorado**.

Este método de treino da rede por épocas permite-nos acompanhar o progresso do treino com mais detalhe. No final de cada época, calcula-se o erro do modelo no conjunto de treino e o erro no conjunto de validação. Estes dois valores são depois representados num gráfico, em que o número da época aparece no eixo dos x e o valor do erro no eixo dos y. Podes ver um desses gráficos na imagem abaixo. Um bom treino é caraterizado por uma diminuição progressiva e comparável desses dois valores até um valor de erro satisfatório — quanto mais próximo de zero estivermos, melhor será o modelo. Recorda que esta conclusão tem por base o facto de o conjunto de validação conter dados separados do conjunto de treino e que a rede nunca viu antes.



Se repararmos que os valores da função de erro no conjunto de treino estão a diminuir, mas no conjunto de validação estão a aumentar, concluímos que o modelo está a sobreajustar-se (*overfitting*) e devemos parar o treino. Se os valores da função de erro do modelo na época anterior ao início do sobre ajuste forem satisfatórios, podemos manter essa versão do modelo para posterior teste no conjunto de teste (normalmente, durante o treino da rede, são guardadas várias versões do modelo com esse objetivo, ou para serem utilizadas caso seja necessário parar e retomar o processo de treino). Caso contrário, será necessário experimentar uma arquitetura de rede ligeiramente diferente ou um conjunto diferente de hiper

parâmetros. Como cada camada da rede tem as suas próprias configurações (número de neurónios, função de ativação, conjunto inicial de parâmetros), as camadas podem estar ligadas de diferentes formas, é necessário acompanhar simultaneamente todas as definições do algoritmo de otimização (por exemplo, o gradiente descendente e o respetivo passo de aprendizagem), e que devem ser cumpridas certas expectativas em termos de qualidade do modelo, treinar uma rede neural é uma tarefa exigente e complexa. É por isso que muitas vezes se diz que representa a arte de treinar (*the art of coaching*).



Monitorização do sobre ajuste da rede neural com base nos gráficos dos valores da função de erro no conjunto de treino e no conjunto de validação.

4.3 Redes Neurais Convolucionais (CNN)

Aprender a representar dados

As redes neurais podem ajudar-nos a isolar certos atributos abstratos nos dados e a aprender representações adequadas para resolver problemas.

Nos exemplos que utilizámos até agora, baseámo-nos, na maioria das vezes, na existência de um determinado conjunto de atributos no conjunto de dados. De facto, muitos domínios geram dados com esta estrutura — atributos dispostos em colunas e instâncias em linhas individuais. Como vimos na parte introdutória da preparação de dados, mesmo quando dispomos destes atributos, não é nada intuitivo decidir quais devemos escolher para criar um modelo. Isso colocou-nos na posição de experimentar diferentes combinações ou de desenvolver técnicas que nos ajudem na seleção de atributos.

Devido à complexidade das funções que modelam, as redes neurais têm a capacidade de aprender a filtrar e agrupar os atributos relevantes. Esta propriedade das redes neurais é especialmente importante quando se trabalha com dados não tabulares — já colocámos várias vezes a questão de como representar, por exemplo, imagens, dados textuais ou gravações de áudio. Apesar de termos algum conhecimento sobre esses formatos, é difícil descrever com precisão e de forma concisa o que contêm, de um modo que seja utilizável. Esta dificuldade, entre outros fatores, levou-nos a adotar o paradigma de programação orientado pelos dados (*data-driven*). As redes neurais conseguem (como veremos em breve) aprender atributos abstratos a partir dos dados na sua forma original, os quais são úteis para resolver problemas com sucesso.

A seguir, vamos apresentar as redes neurais convolucionais, usadas principalmente no trabalho com imagens e vídeos, para aprender atributos visuais da entrada, e depois as redes neurais recorrentes e os

transformers, tipos de redes neurais usados para aprender os atributos de dados sequenciais, como texto ou som.

Redes Neurais Convolucionais

As redes neurais convolucionais são um tipo de rede neural utilizado principalmente na área da visão por computador, para trabalhar com imagens e conteúdos de vídeo

As imagens a preto e branco são representadas por matrizes de píxeis. O número de linhas dessa matriz corresponde à altura da imagem, enquanto o número de colunas corresponde à sua largura. Os valores individuais dos píxeis são números que variam entre 0 e 255, onde 0 representa preto e 255 representa branco. Todos os valores intermédios representam tons de cinzento. Consegues adivinhar o que está por trás da imagem representada pela matriz de píxeis abaixo? Se te afastares o suficiente e desenhares os contornos nas zonas de tons mais escuros, talvez consigas perceber o que está na imagem. Ajuda o facto de, na comunidade de machine learning, as imagens de gatos serem uma escolha bastante comum.

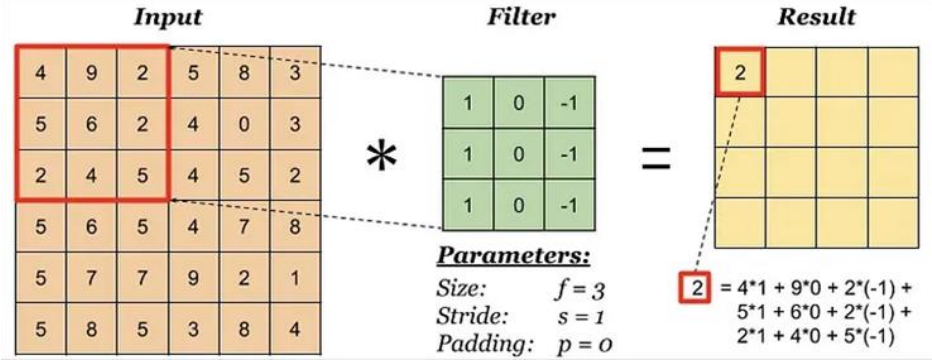
```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
 [  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
 [  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
 [  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
 [  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
 [  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
 [  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
 [  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
 [  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
 [  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
 [  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
 [  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
 [ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
 [ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]]
```

Também se pode verificar o próximo bloco e ver qual é a imagem.

Tal como para nós, humanos, é difícil perceber o que está numa imagem representada por um conjunto de números, o mesmo acontece com as redes neurais convolucionais. Estas começam a sua análise da imagem reconhecendo primeiro elementos simples, como linhas horizontais e verticais, e depois combinam-nos e adicionam complexidade, até chegarem a descrições mais complexas da imagem, que as ajudam a resolver a tarefa em questão. A pergunta natural que se coloca agora é: como é que se começa por contornos

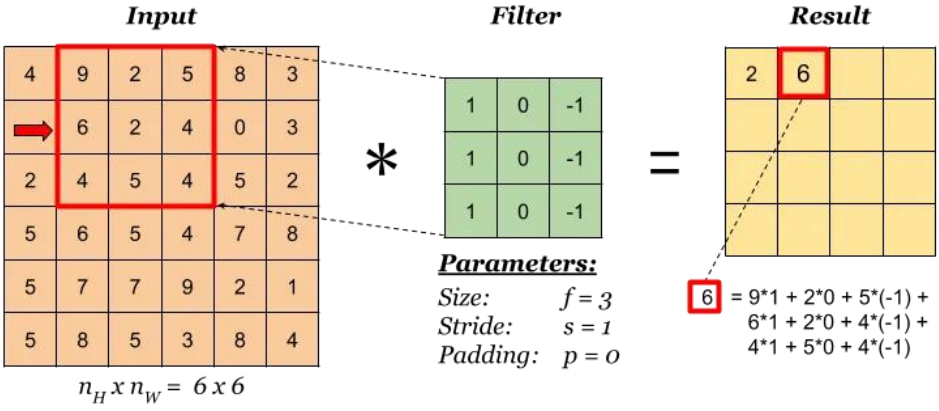
simples e se constrói a partir daí para obter estruturas mais complexas? A resposta será dada pelo **operador de convolução**.

A forma mais simples de introduzir o operador de convolução é através de ilustrações. Imaginemos que temos, à entrada, uma matriz de 6x6 píxeis que representa a imagem, e uma pequena matriz, o chamado filtro, com dimensões de 3x3 píxeis. Suponhamos que estas são as matrizes mostradas na imagem abaixo. Começamos a aplicar o operador de convolução sobre a imagem (iremos marcá-lo com o símbolo✳) ao sobrepor a parte superior esquerda da imagem com o filtro, depois multiplicamos os valores correspondentes e escrevemos a soma desses valores numa nova matriz. Esta nova matriz será o resultado da aplicação do operador de convolução.



Convolução - Passo 1

Continuaremos a aplicar o operador de convolução: vamos sobrepor o filtro com a parte da imagem localizada no canto superior esquerdo, mas para que agora seja deslocado um pixel da borda esquerda, ou seja, em comparação com a posição anterior. Novamente, vamos multiplicar os valores individuais, somá-los e escrevê-los na matriz resultante.



Convolução - Passo 2

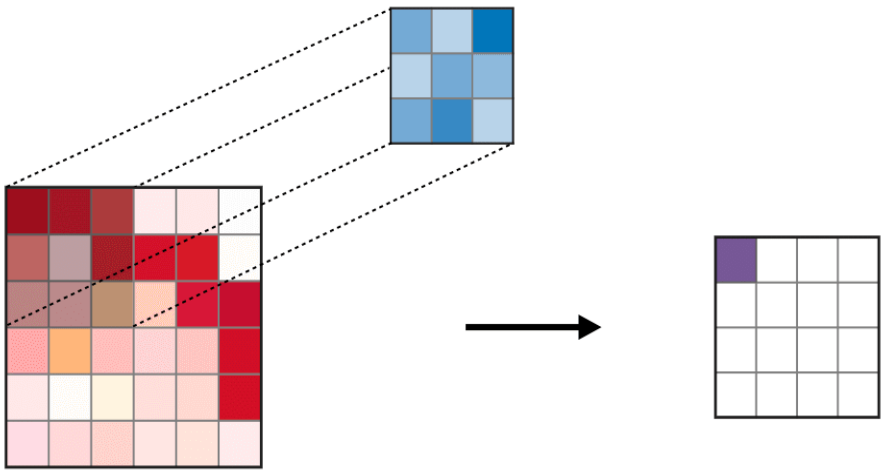
O filtro pode ser movido uma posição para a direita até a borda. Então precisamos baixá-lo uma posição para baixo e colocá-lo de volta ao lado da borda. Em seguida, podemos continuar o processo até chegarmos ao canto inferior direito. Como resultado desta operação, obteremos uma matriz de dimensões de 4x4 pixels, cujos valores são mostrados na imagem abaixo.

2	6	-6	5
0	4	0	-1
-5	0	3	6
-2	5	0	3

Convolução - o resultado

A distância que o filtro se desloca em cada iteração é definida por um hiperparâmetro chamado deslocamento (*stride*). No nosso caso, o deslocamento teve o valor de 1, porque movemos o filtro uma posição para a direita (e depois uma posição para baixo). Para podermos influenciar as dimensões da matriz resultante ao aplicar a operação de convolução (normalmente queremos preservar as dimensões da matriz de entrada), podemos adicionar uma moldura à volta da imagem inicial. Essa moldura é, geralmente, um bloco de zeros, uns ou números cujos valores correspondem ao píxel mais próximo da imagem. Esta técnica chama-se preenchimento (*padding*) e a sua largura é sempre indicada aquando da aplicação da operação de convolução. O preenchimento é especialmente importante quando as características que queremos que o nosso modelo aprenda estão próximas da margem da imagem.

A animação abaixo ilustra todo o processo de aplicação dos filtros à imagem, ou seja, à sua matriz. Como pode ver, foi utilizado um deslocamento de tamanho 1 e um preenchimento de tamanho 0.



Animação da operação de convolução

Se aplicarmos o filtro do exemplo anterior usando a operação de convolução sobre a imagem inicial do gatinho, obtemos a imagem abaixo. Podes ver que todas as linhas verticais que aparecem na imagem foram realçadas.

Extração de borda vertical

Ficou surpreendido com o filtro superior que detetou bordas verticais?

Aqui está uma explicação. Olhe para a imagem da esquerda para a direita. A primeira vez que se move da parte clara da imagem para a parte escura da imagem, consegue-se, realmente ver uma borda vertical. Vamos agora aplicar um filtro da esquerda para a direita com uma operação de convolução. O maior resultado de uma iteração da convolução será quando o lado direito do nosso filtro (a coluna numerada -1) estiver posicionado exatamente na borda vertical. Como a borda é escura, os valores que correspondem a essa cor são pequenos porque a cor preta é representada por zero. Os valores à esquerda da borda são claros, de modo que os números que correspondem a essas cores são maiores (o valor para branco é 255). Quando os pequenos valores correspondentes à cor preta das bordas são multiplicados por -1, ou seja, com a parte direita do filtro, e os valores grandes, que correspondem às cores claras à esquerda da borda, são multiplicados por 0 e 1, ou seja, o resultado é um valor maior do que se o filtro fosse encontrado em qualquer outro lugar onde não há filtro.

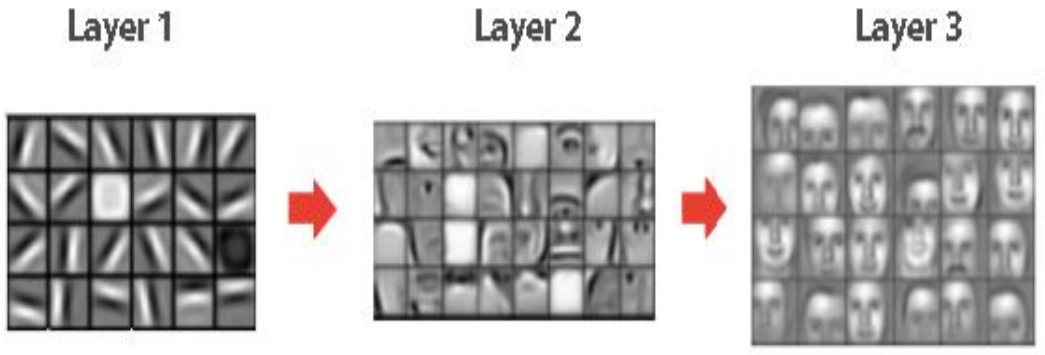
Como se separaria as bordas horizontais na imagem? Qual filtro que usaria?

Tudo o que se precisa fazer é girar o filtro que separa as bordas verticais! Isso faz sentido?

Agora que sabemos como separar arestas verticais e horizontais, podemos combinar de várias maneiras para destacar linhas que não são apenas horizontais e verticais. Ao combinar ainda mais estes resultados, podemos até extrair contornos esféricos. Isto é o que queríamos dizer quando dissemos que começamos com características claras e fáceis de aprender e, em seguida, construímos passo a passo sobre a complexidade das características que podemos aprender.

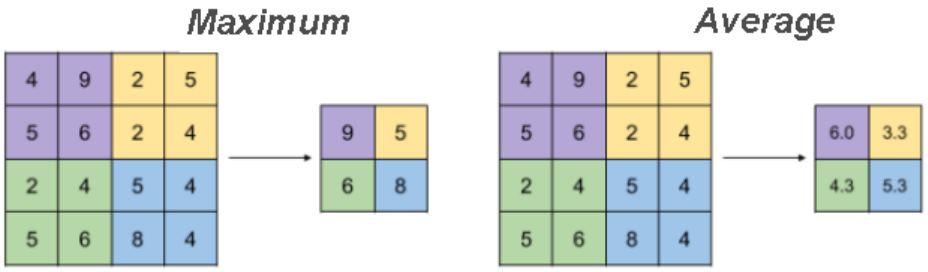
As camadas de uma rede neural caraterizadas pela aplicação do operador convolucional são chamadas **camadas convolucionais** . Enquanto os pioneiros no campo da visão computacional criaram filtros manualmente, o objetivo de treinar redes neurais convolucionais é aprender por si mesmos os valores que figuram nelas.

Na imagem abaixo, pode-se ver as representações dos filtros aprendidos nas camadas de uma rede neural convolucional que reconhece rostos. Na camada mais baixa, são algumas linhas horizontais, verticais e diagonais, na segunda camada já são contornos que correspondem a partes da face como nariz, olhos e boca, enquanto na terceira camada são filtros que correspondem aos contornos da face.



Classification process of passing through successive layers of the image

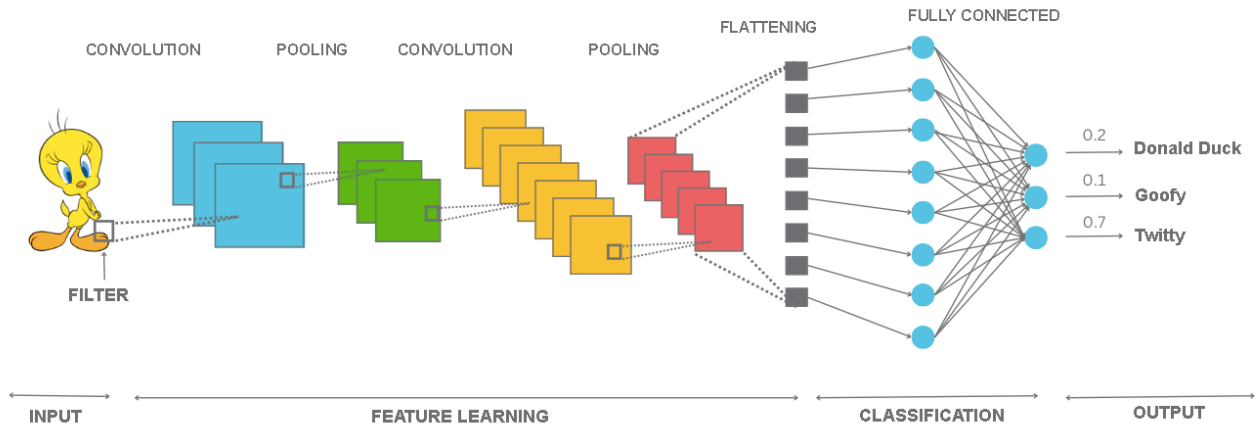
Além da operação convolucional, as redes convolucionais também são caraterizadas pela operação de agregação. Como o nome sugere, o objetivo desta operação é agregar, ou seja, consolidar as entradas. Na imagem abaixo, pode-se ver dois tipos de operadores de agregação: o que usa o máximo e o que usa a média para agregar as informações. Assim como o operador convolucional, este operador é aplicado a blocos de entrada percebendo o bloco e realizando o cálculo necessário nele. O valor obtido desta forma é inserido numa nova matriz. Na figura, ambos os operadores são aplicados a blocos 2x2. Intuitivamente falando, enfatizamos a parte mais dominante com máximos, enquanto que ao calcular a média, levamos em conta a contribuição de todas as partes.



Ao aplicar a operação de agregação, obtemos a capacidade de reduzir a dimensão da entrada, mas ao mesmo tempo reter algumas das informações contidas. Na imagem, consegue-se ver que, aplicando o operador de agregação, reduzimos a matriz de 4x4 para 2x2. Por que razão precisamos de reduzir as dimensões? Como resultado, a rede precisa nos dar uma resposta específica, digamos se há um gato na imagem ou não, para o qual precisamos de um pequeno número de neurónios.

As camadas de uma rede neural que são caraterizadas pela aplicação do operador de agregação são chamadas camadas de **agregação.** Não há parâmetros adicionais neles que a rede precise aprender, mas, como vimos, eles nos ajudam a controlar as dimensões das matrizes com as quais trabalhamos.

Agora que sabemos quais são os blocos de construção de uma rede neural convolucional, vamos ver como a podemos ligar e obter um modelo funcional que pode nos ajudar a resolver a tarefa de classificação. Vamos imaginar que precisamos resolver uma tarefa de classificação multiclasse em que para cada imagem de um personagem de desenho animado precisamos determinar se é Piu-Piu, Pateta ou Pato Donald. Vejamos uma ilustração da arquitetura de uma rede neural convolucional profunda que escolhemos para resolver esta tarefa e discutir qual é a motivação para a sua criação.



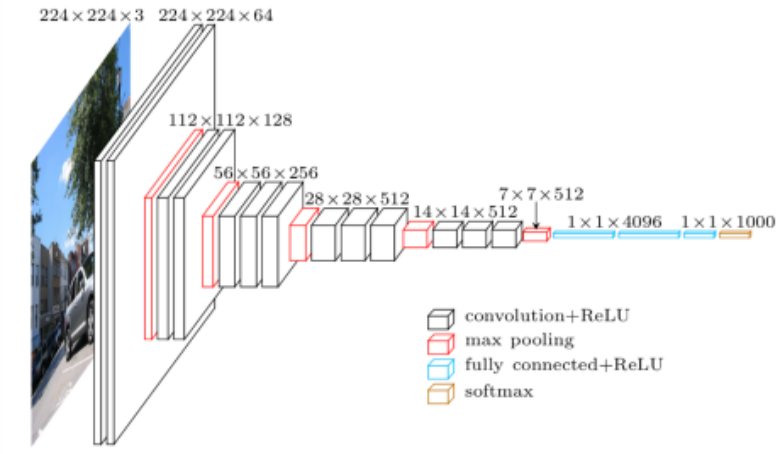
No início da rede, existe uma camada de entrada que contém os pixéis da imagem. A seguir, temos uma camada convolucional (um bloco azul). O objetivo desta camada é utilizar o operador de convolução para extrair o primeiro conjunto de atributos abstratos. Depois disso, colocamos uma camada de *pooling* (bloco verde), seguida por outra camada convolucional (bloco laranja) e outra camada de *pooling* (bloco vermelho). Na prática, as camadas convolucionais e de *pooling* são combinadas e frequentemente colocadas lado a lado, pois as camadas de *pooling* têm como função resumir aquilo que as camadas convolucionais aprenderam. A segunda camada convolucional permite-nos aplicar um segundo operador de convolução aos atributos já extraídos pela primeira camada convolucional, formando assim atributos mais complexos da imagem. Este bloco de camadas é seguido por uma camada (bloco cinzento) cuja tarefa é "corrigir" a matriz (ou mais precisamente, o tensor) que obtivemos até esse ponto e reorganizar os seus valores de modo a que fiquem todos dispostos lado a lado numa única matriz unidimensional. As camadas com esta função chamam-se camadas de achatamento (*flattening layers*). As camadas com esta função chamam-se camadas de achatamento (*flattening layers*). Além da camada de entrada (já corrigida), também vemos na imagem uma camada escondida (*hidden layer*) e uma camada de saída, na qual existem exatamente três neurónios – um para cada uma das personagens de banda desenhada. Os valores de saída destes neurónios correspondem à probabilidade de que a imagem de entrada pertença à classe que cada um representa. No caso da imagem do Twitter que temos na entrada, podemos notar que o valor de saída do terceiro neurónio – que corresponde exatamente a essa classe – é o mais elevado e tem o valor 0,7.

Agora também podemos ver qual é a arquitetura da VGGNet, uma rede convolucional popular que é ativamente utilizada na prática.

A VGGNet é uma rede neural convolucional profunda desenvolvida pela equipa de Oxford chamada Visual Geometry Group (daí o nome VGGNet). No prestigiado Large Scale Visual Recognition Challenge, realizado em 2014, esta rede revelou-se a melhor na resolução do problema de localização de objetos numa imagem e a segunda melhor no problema de classificação de objetos a partir de uma imagem. Para a tarefa de classificação, foram utilizadas mais de 1,2 milhões de imagens do conjunto ImageNet (que já conhecemos), com classificação possível em 1000 classes distintas. Para treinar esta rede foram necessários entre 15 e 20 dias, utilizando 4 placas gráficas NVIDIA Titan Black (as melhores da época). Antes da VGGNet, a rede mais destacada nestas tarefas era a AlexNet, que ficou conhecida por ter introduzido a prática de usar placas gráficas para treinar redes neurais, permitindo assim o avanço significativo da área de aprendizagem profunda (deep learning).

A arquitectura da rede VGG-16, uma versão da rede com 16 camadas, está representada na imagem abaixo. Como podemos ver, a rede espera à entrada uma imagem a cores com dimensões de 244x244 píxeis, e combina camadas convolucionais com camadas de pooling (com máximo). Termina com uma rede neural totalmente conectada com 1000 neurónios na saída, onde cada neurónio corresponde a uma classe

específica do conjunto ImageNet. A própria rede possui 138 milhões de parâmetros e necessita de cerca de 500 MB de memória para os armazenar.



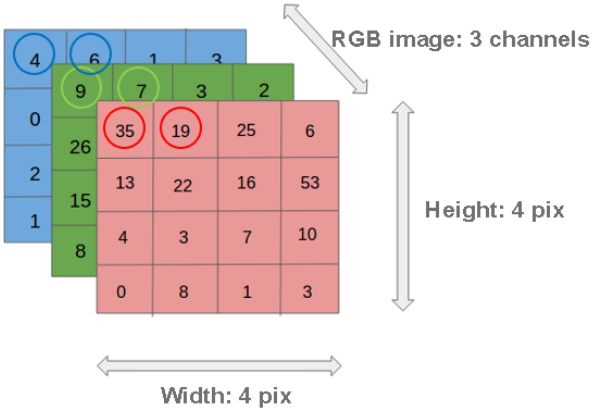
Ecrã de rede VGG-16

Aumentar o número de camadas da parte convolucional da rede geralmente conduz a melhores resultados na prática. No entanto, o número de camadas não pode aumentar indefinidamente. Isto deve-se não só às limitações de recursos, tempo e custo, mas também às propriedades matemáticas das redes neurais profundas, que dificultam ainda mais a aplicação do algoritmo de retropropagação e o próprio treino da rede.

Se tiver interesse neste problema matemático, pode tentar ler mais sobre os gradientes que *desaparecem e explodem*, especialmente na parte relativa às redes convolucionais e às ligações residuais (*residual connections*).

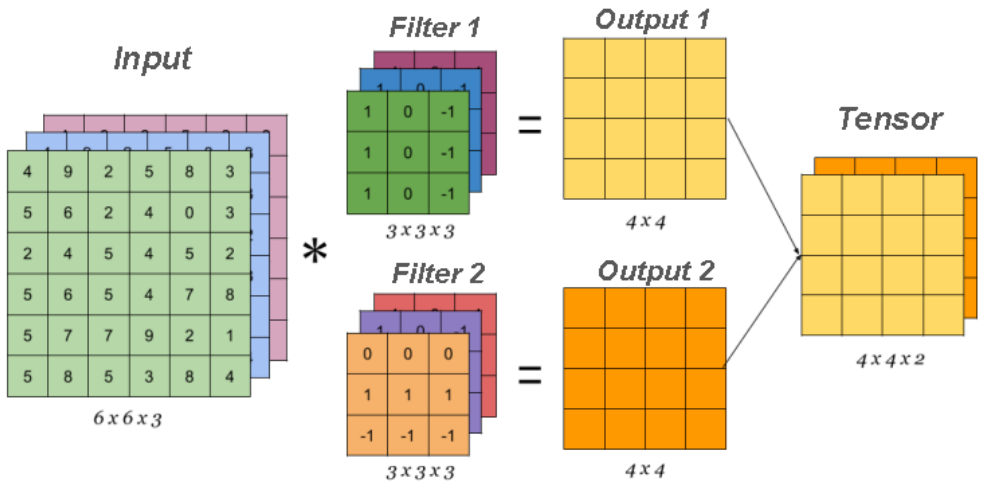
Antes de entrarmos na tarefa de trabalhar com redes neurais convolucionais, vamos analisar a questão do trabalho com imagens coloridas. Até agora, ainda não as mencionámos.

Quando precisamos de apresentar uma imagem a cores, aquela que utiliza o formato de cor RGB e exibe todas as cores como combinações de vermelho, verde e azul, usamos três matrizes. Uma matriz é atribuída a cada uma das cores. O número de matrizes que usamos para representar imagens chama-se **canais**. Assim, imagens a preto e branco têm apenas um canal, enquanto imagens a cores têm três canais.



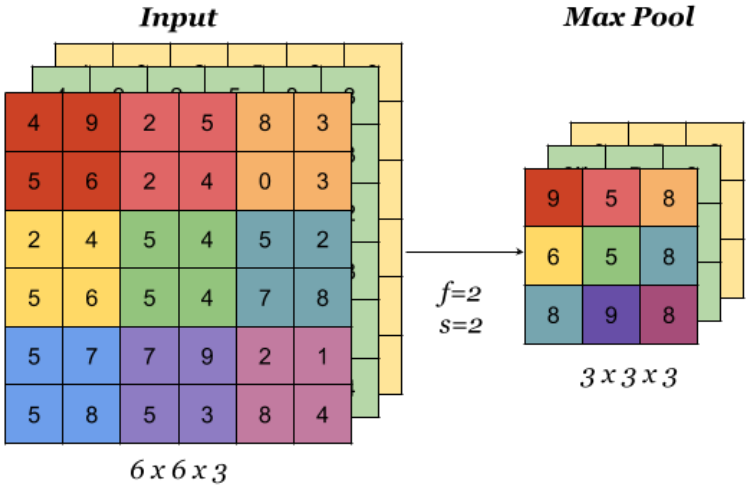
Exibição de imagens que usam o formato de cor RGB

A presença de cores afeta o desempenho da operação convolucional, ajustando o número de canais do filtro para o número de canais da imagem à qual estamos aplicando-o. Depois, é necessário emparelhar cada um dos canais de filtro com o canal de imagem (vermelho com vermelho, azul com azul e verde com verde) e executar a operação convolucional como se estivesse trabalhando com um único canal. Então, precisamos adicionar as matrizes que obtemos dessa maneira e declarar a matriz resultante como o resultado final. Na imagem abaixo, podemos ver dois filtros numa camada convolucional que são aplicados sobre a imagem de entrada de cores. Como resultado da aplicação de cada um desses filtros, obteremos matrizes separadas que, quando "fundidas", representam o resultado final da camada convolucional. Na prática, vários filtros são geralmente colocados ao nível de uma camada convolucional, de modo que os tensores são obtidos como resultados. Operações convolucionais são então aplicadas a esses tensores da mesma maneira - apenas o cuidado é tomado para garantir que o número de canais de filtro corresponde à dimensão do tensor (digamos, para a próxima aplicação no exemplo que consideramos, este seria o número 2) e que o canal de entrada correspondente é emparelhado com o canal de filtro correspondente.

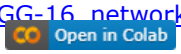


Aplicando o operador convolucional a entradas multicanal

Quanto à operação de *pooling*, ela é aplicada a cada canal da imagem de entrada. Por exemplo, se a imagem de entrada tiver 3 canais, a operação de *pooling* será aplicada a cada canal separadamente. Isto também significa que a operação de *pooling* preserva o número de canais no momento da aplicação. Na imagem abaixo podes ver uma ilustração deste processo.



Aplicando operadores de agregação a entradas multicanal

Esta seção é emparelhada com o Jupyter Notebook 11-VGG-16_network_and_classification.ipynb. Para acompanhar o conteúdo, clique no link e, em seguida, clique 

no botão para abrir o conteúdo no *Google Colab* . Se estiver a visualizar os blocos de notas no dispositivo, localize o bloco de notas com o mesmo nome entre os conteúdos e execute-o. Para instruções mais detalhadas, consulta a secção *Hands-on* e a aula Jupyter Notebooks para a prática.

Agora vamos experimentar como a *rede convolucional VGG-16* realmente funciona! Não esquecer de seguir o caderno com o código.

Uma vez treinado, um modelo de rede neural pode ser partilhado com a comunidade, disponibilizando os parâmetros que o compõem. Neste exemplo, iremos utilizar o modelo disponível na biblioteca **Keras**. A biblioteca Keras é uma biblioteca de código aberto amplamente utilizada na comunidade de machine learning (*machine learning*). Para tirar partido do modelo de rede **VGG-16**, precisamos de executar os seguintes dois comandos:

```
from tensorflow.keras.applications import VGG16

model = VGG16(weights = 'imagenet')
```

Podemos ler as informações sobre o modelo que carregamos usando a função model.summary(). Seu resultado é uma descrição das camadas de rede seguida de informações sobre os tamanhos de entrada que essas camadas esperam. Agora, pode executar o comando:

```
model.summary()
```

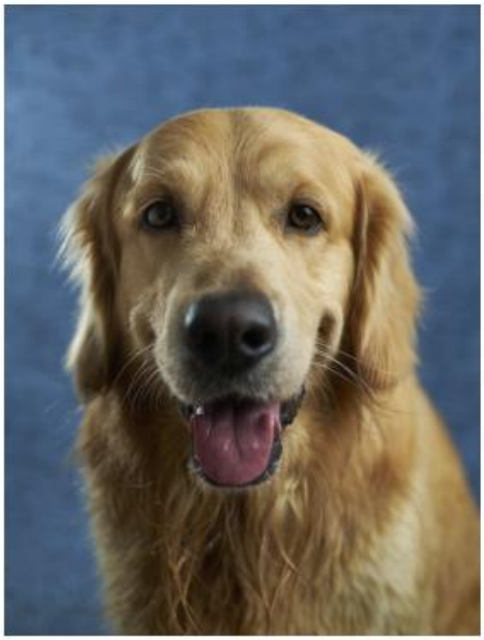
Não se preocupe se não compreender todos os detalhes que são exibidos após a execução deste comando. É importante saber que o input deve ser uma imagem a cores com dimensões de 224x224 píxeis (é por isso que aparece **224, 224, 3** ao lado da camada de entrada) e que tens uma das 1000 classes no output. Pode também comparar a impressão com a imagem da VGG-16 que discutimos anteriormente para descobrir mais informações.

É importante salientar que não iremos treinar a rede VGG-16 — usaremos apenas um modelo já treinado. Por isso, não devemos alterar os parâmetros do modelo durante a sua utilização — cada parâmetro tem a sua própria contribuição. O número total de parâmetros do modelo, que podemos verificar no resumo da rede, é de pouco mais de 138 milhões.

A ideia é que, a imagem em que vamos testar o modelo seja uma imagem arbitrária da web. Para fazer isso, usaremos várias bibliotecas Python padrão. Por padrão, a função URL ucitaj_sliku nos ajudará a arrastar a imagem que desejamos.

```
def load_image(url_path):

        response = request.urlopen(url_path, context=ssl_context).read()

        return Image.open(BytesIO(response))
```

Para o teste, escolhemos uma imagem de um Golden Retriever do endereço https://unsplash.com/photos/x5oPmHmY3kQ, que pode ser usada livremente. Podes escolher a imagem que quiseres! É importante ter em mente que a classe do objeto na imagem deve ser conhecida pelo modelo. Como *o modelo VGG-16 foi* treinado em mais de 1,2 milhão de imagens, ele conhece muitas classes, até 1000 diferentes. O Golden Retriever é um deles. Se dermos ao modelo uma imagem com um objeto que ele não conhece, ele nos dará previsões das classes cujas imagens mais se assemelham às nossas. Veremos no final quais classes se assemelham ao Golden Retriever.



O herói da história do modelo VGG-16

Como a imagem que precisa ser encaminhada para o modelo precisa ser especialmente preparada, faremos o seguinte:

definir suas dimensões para 224x224 e dizer-lhe para usar três canais de cor RGB:

```
test_image = test_image.resize((224, 224))

test_image = test_image.convert('RGB')
```

Transforme a imagem num formato matricial adequado:

```
matrix_form_test_image = image.img_to_array(test_image)
```

criar um pacote que contenha a nossa imagem:

```
batch = np.expand_dims(matrix_form_test_image, axis=0)
```

Execute o pré-processamento numérico de imagens na forma de normalização:

```
test_image_batch = preprocess_input(batch)
```

Só assim poderemos transmitir o modelo de classificação. A função que nos ajudará é chamada de previsão.

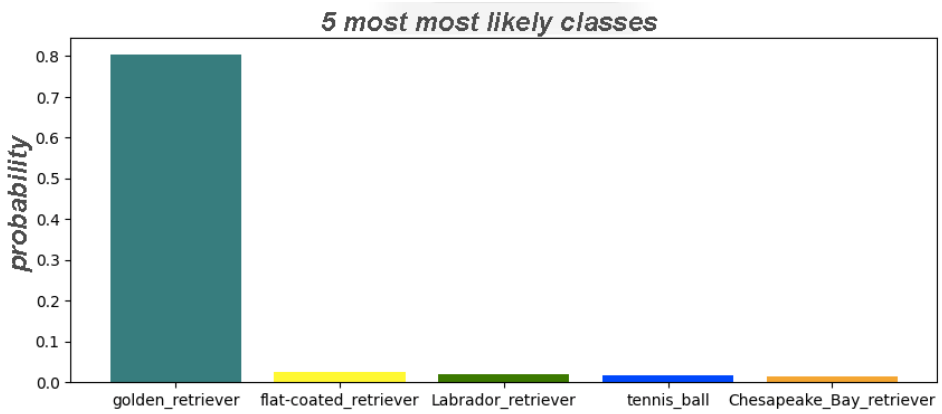
```
model_predictions = model.predict(test_image_batch)
```

A variável predikcije_modela, na qual armazenámos as previsões do modelo, é uma matriz com 1000 elementos, e contém as probabilidades de pertencer a cada uma das 1000 classes que o modelo reconhece. Para extrair a classe à qual a nossa imagem pertence, podemos usar a função decode_predictions, que nos devolverá as 5 classes mais prováveis, juntamente com as respetivas probabilidades e nomes. Isto permite-nos ter uma noção de quão confiante está o modelo na classificação. Depois de executarmos o próximo comando, obteremos informações sobre as classes mais prováveis.

```
most_likely_classes = decode_predictions(model_predictions)[0]
```

Quando representamos graficamente essas previsões com o código descrito abaixo, obtemos um gráfico com barras que nos permite analisar mais facilmente os resultados.

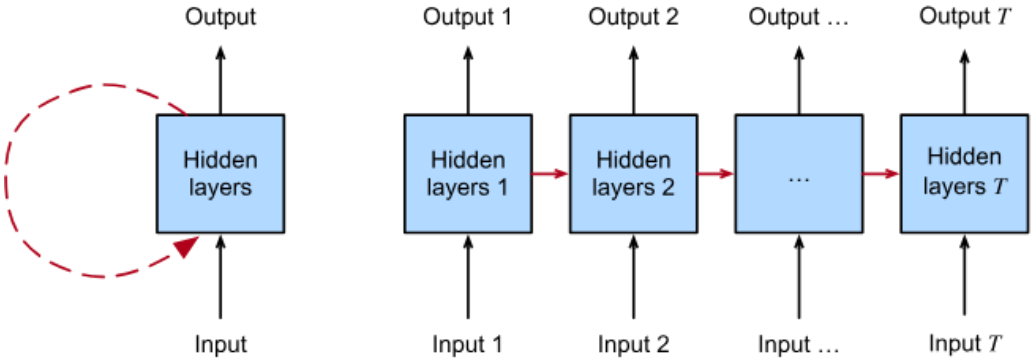
```
class_names = [item[1] for item in most_likely_classes]

class_probabilities = [item[2] for item in most_likely_classes]

plt.figure(figsize=(10, 4))

plt.bar(class_names, class_probabilities, color=['teal', 'yellow', 'green', 'blue',
'orange'])

plt.title('Top 5 Most Likely Classes')

plt.ylabel('Probability')

plt.show()
```



Como podemos ver, o modelo previu com grande certeza (a probabilidade foi de 0,804) que a imagem escolhida era de um golden retriever. Algumas das outras classes que o modelo considerou foram outros tipos de retrievers. Curiosamente, uma bola de ténis também apareceu na lista de resultados. Provavelmente porque, no conjunto de treino, também existem imagens de retrievers a correr atrás de bolas de ténis. Este comportamento do modelo deve ser investigado mais a fundo na prática.

4.4 Redes neurais recorrentes

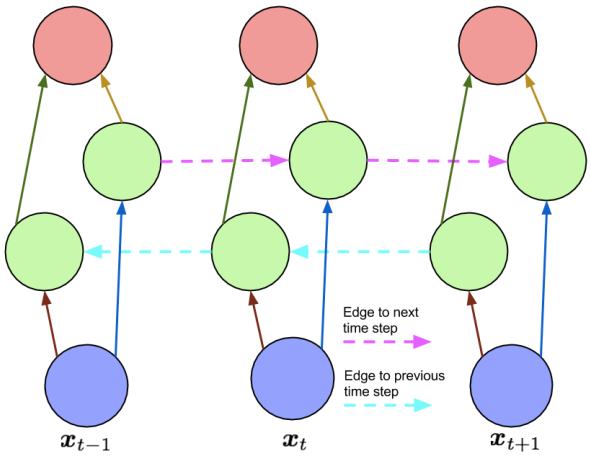
As **Redes Neurais Recorrentes** (Recurrent Neural Networks – RNNs) são um tipo de rede neural utilizado, principalmente, para o processamento de dados sequenciais. As redes recorrentes processam a sequência elemento por elemento.Os dados sequenciais, ou sequências, são compostos por elementos que surgem uns após os outros. Exemplos disso são: dados textuais (em que os elementos são palavras individuais); gravações de áudio (elementos são amostras individuais); séries temporais (elementos são medições ao longo do tempo); sequências genéticas (elementos são nucleótidos individuais), entre outros. As redes recorrentes processam a sequência elemento a elemento.Para processar um elemento na posição t, todos os elementos anteriores têm de ser processados. E, para que os elementos da sequência sejam ligados num todo coerente, os valores das camadas ocultas são partilhados entre o processamento de elementos sucessivos da entrada. Este funcionamento é normalmente representado graficamente, como mostrado na figura abaixo.



Rede neural recorrente

(imagem retirada de https://d2l.ai/chapter_recurrent-neural-networks/index.html)

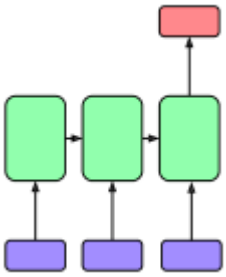
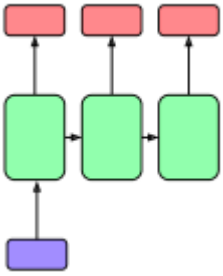
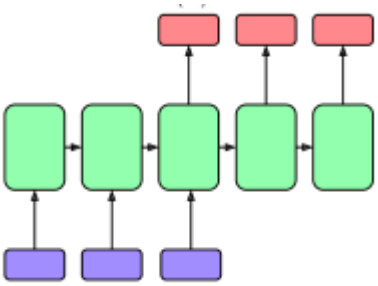
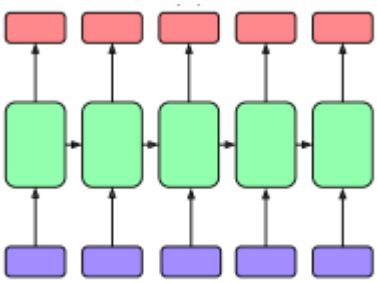
As redes neurais recorrentes são, hipoteticamente, capazes de processar sequências infinitamente longas, elemento por elemento. No entanto, ao treinar este tipo de redes, observou-se que elas esquecem. Se as sequências forem demasiado longas, a rede começa a esquecer o que viu no início e guarda apenas as informações mais recentes ao nível das camadas ocultas. Esta observação levou à criação de neurónios especiais chamados LSTM (*Long Short-Term Memory*) e GRU (*Gated Recurrent Unit*), cujos detalhes não abordaremos aqui devido à sua complexidade. Uma das soluções para este problema são as redes neurais recorrentes bidireccionais (*Bidirectional Recurrent Neural Networks*): nestas redes, por um lado, a sequência é processada do início para o fim, e por outro, do fim para o início. A representação de entrada dos elementos individuais é composta pelas representações combinadas desses dois percursos, como ilustrado na figura abaixo.



Rede neural recorrente bidirecional - elementos sequenciais

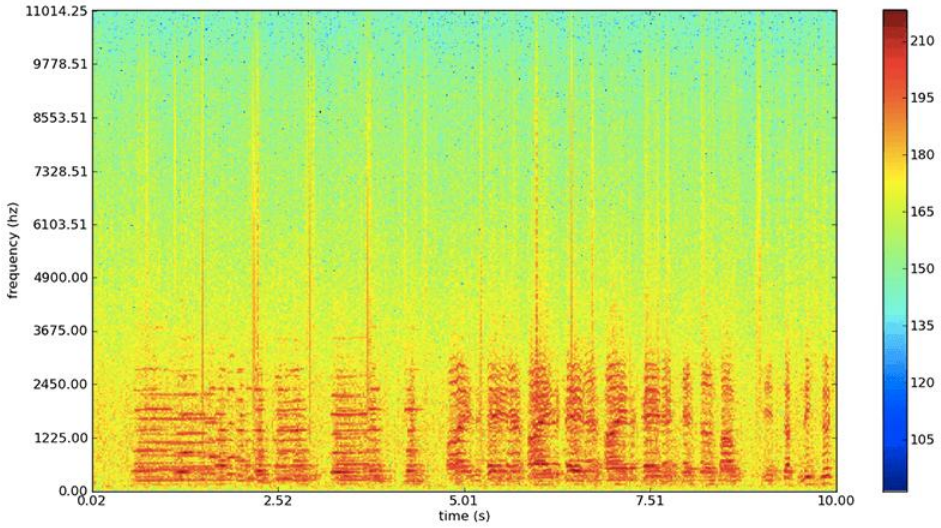
(imagem retirada de https://www.arxiv-vanity.com/papers/1506.00019/)

Existem várias arquiteturas populares de redes neurais recorrentes. Na tabela abaixo, vamos apresentar brevemente alguns dos exemplos mais conhecidos, mostrando-os graficamente na coluna da esquerda e descrevendo a rede e as suas áreas de aplicação na coluna da direita.

ARQUITETURA	CLARIFICAÇÃO E EXEMPLOS DE APLICAÇÃO
	Este tipo de rede corresponde a tarefas em que a entrada é uma sequência e a saída é uma representação vetorial de um comprimento fixo. Redes deste tipo são chamadas codificadores, e os vetores obtidos de comprimentos fixos por contexto. As tarefas em que encontramos este tipo de redes são várias tarefas de classificação, como a classificação de faixas de áudio ou a classificação de texto.
	Ao contrário do exemplo anterior, a entrada para este tipo de rede é uma representação vetorial de comprimento fixo, e a saída é uma sequência. Este tipo de rede chama-se **descodificador** (decoder). As tarefas em que utilizamos descodificadores incluem a geração de títulos para imagens.
	Este tipo de rede é uma combinação dos dois tipos anteriores e é chamado de **arquitetura codificador-decodificador** (encoder-decoder). A tarefa do codificador é criar uma representação (contexto) baseada na sequência de entrada, que o decodificador pode usar para gerar uma nova sequência de saída. Este tipo de rede é utilizado em tarefas como tradução automática ou geração de resumos (abstracts).
	Este tipo de rede permite a geração de saídas para cada elemento da entrada. Como podes ver, tanto a entrada como a saída são sequências. As tarefas em que encontramos este tipo de rede incluem, por exemplo, tarefas de anotação (ou marcação) de elementos individuais.

Uma das principais desvantagens das redes neurais recorrentes é a incapacidade de serem paralelizadas: para processar um elemento na posição *t*, é necessário que todos os elementos anteriores já tenham sido processados. Por isso, o treino de redes neurais recorrentes requer muito mais tempo e recursos do que o treino das redes neurais convolucionais que conhecemos na secção anterior. Estas limitações levaram ao aparecimento do mecanismo de atenção e dos transformers, redes neurais que serão abordadas com mais detalhe na próxima secção.

As gravações de áudio também podem ser processadas usando redes neurais convolucionais. Ou seja, uma gravação de áudio pode ser dividida em fragmentos, pequenos trechos com alguns segundos de duração, e depois podem ser criados espectrogramas para cada parte. Um espectrograma é uma representação gráfica de todas as frequências de som presentes numa gravação áudio. As imagens resultantes podem então ser usadas como entradas para redes neurais convolucionais e utilizadas para análise de áudio.

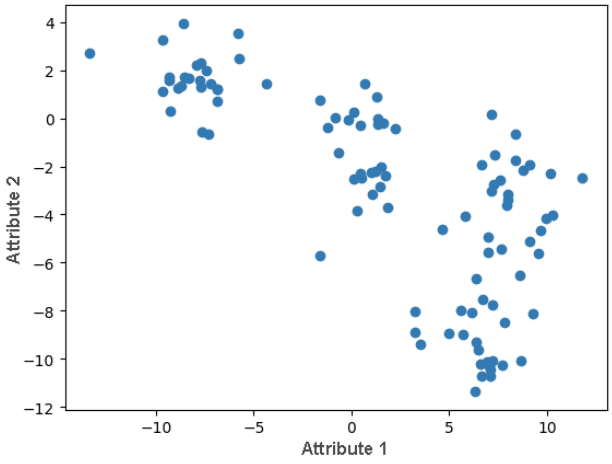


Um exemplo de espectrograma

4.5 Algoritmo dos K-Médias (K-Means)

O nome "k-médias" é um nome invulgar para um algoritmo. Continua a ler para descobrires o que está por detrás desta escolha.

Para facilitar o acompanhamento da explicação do algoritmo x-médias, vamos utilizar o conjunto de dados mostrado na figura abaixo. Consiste em 100 pares de pontos — imagina que estes representam os valores de dois atributos numéricos.



Esta secção esta associada ao Jupyter Notebook 12-k-means.ipynb. Para acompanhar o conteúdo, clique no link e, em seguida, clique Open in Colab no botão para abrir o conteúdo no *ambiente do Google Colab*. Se estiveres a visualizar os notebooks no teu computador local, procura o notebook com o mesmo nome entre os ficheiros e executa-o. Para instruções mais detalhadas, consulta a secção *Hands-on Zone* e a lição sobre o *Jupyter Exercise Notebook*.

No material de apoio, poderá gerar todas as imagens e animações.

O algoritmo **k-médias** (k-means) tem como objetivo encontrar *k* grupos (ou *clusters*) no conjunto de dados. Os grupos que este algoritmo procura são determinados pelo **centróide**, uma instância que representa o centro do grupo.

O primeiro passo do algoritmo é a inicialização. Nesta fase, precisamos de selecionar aleatoriamente *x* centróides. Depois, repetimos os seguintes passos. Para cada instância, calculamos a distância euclidiana até cada um dos *x* centróides, e associamos a instância ao grupo cujo centróide estiver mais próximo. Após atribuir todas as instâncias, passamos ao passo seguinte. Para cada um dos *x* grupos, selecionamos novos centróides. Fazemo-lo calculando a média das instâncias que pertencem a cada grupo e declarando esse valor como o novo centróide. Depois disso, voltamos ao passo 1.

O algoritmo de agrupamento termina quando os valores dos centróides estabilizam. Isto significa que, em duas iterações sucessivas, os centróides variam muito pouco — menos do que uma precisão previamente definida.

O próprio algoritmo k-médias não é difícil de programar, por isso vamos fazê-lo juntos. Antes disso, vamos considerar alguns detalhes técnicos:

Uma instância de um conjunto de dados é um par de números, por exemplo (2, -3). Isto significa que um centróide também será um par de números e terá duas coordenadas;

Se (10, 2) e (4, -4) forem duas instâncias de um conjunto de dados, calcularemos a média delas como (10 + 42, 2 − 42) = (7, -1);

Se (0, 0) e (3, 4) forem duas instâncias de um conjunto de dados, calcularemos a distância euclidiana entre elas será (3 − 0)2 + (4 − 0)2.

No conjunto de dados, procuraremos quatro grupos. Mais à frente discutiremos porque escolhemos este número. Por agora, vamos preparar-nos para programar o algoritmo.

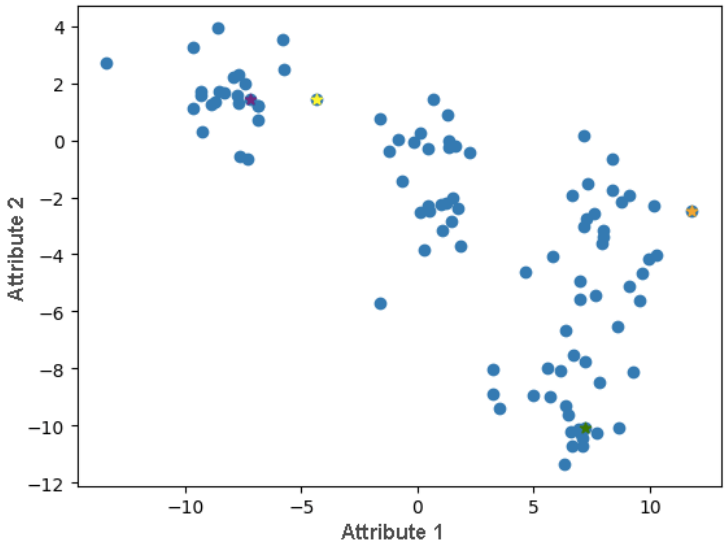
A variável k representa o número de grupos. Neste caso, k = 4. Vamos representar os centróides com a variável centroide. Como temos *k* grupos, isto será um *array* de comprimento *k*. Cada centróide, como dissemos, é um par de números, portanto os elementos deste array serão pares de números.

Durante o processo de agrupamento, é necessário registar a que grupo está associada cada instância. Para isso, usaremos rótulos (labels), tal como nas tarefas de classificação. Estes podem ser os valores 0, 1, 2 e 3 — em geral, os valores 0, 1, 2, ..., k-1. Guardaremos todos os rótulos numa série de clusters de rótulos.

Vamos agora introduzir a função generisi_centroide(X, k), que gera os centróides iniciais. Os seus argumentos são o conjunto de instâncias **X** e o número de grupos **k**, e a função seleciona aleatoriamente *k* números do intervalo de 0 a 100 e devolve as instâncias que se encontram nessas posições.

```
def generate_centroids(X, k):

    N = X.shape[0]

    indices = np.random.randint(low=0, high=N, size=k)

    return X[indices]
```

Na figura abaixo, os centróides gerados são mostrados. Cada um deles está na cor do aglomerado que representa.



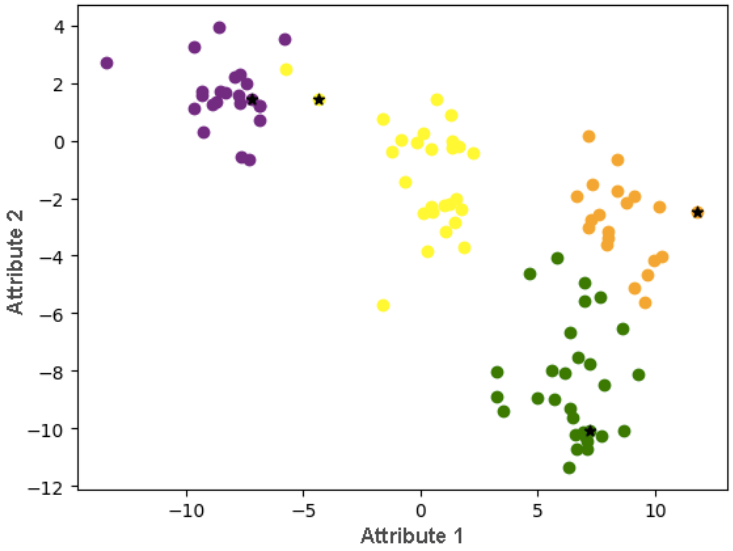
Valores iniciais dos centróides

Agora vamos escrever uma função podeli_podatke(X, centróides, k) para dividir um conjunto de instâncias em clusters. Esta função recebe como argumentos o conjunto de instâncias X, os centróides atuais (centroide) e o número de grupos k. Para cada instância, calculamos a distância até cada um dos centróides, selecionamos o centróide mais próximo e concluímos que a instância pertence ao grupo correspondente a esse centróide.

```
def divide_data(X, centroids, k):

  # initialize the list of cluster labels

  cluster_labels = []

  # iterate through the dataset instance by instance

  for x in X:

    # initialize the list of distances to centroids

    distances_to_centroids = []

    # then for each centroid ...

    for centroid in centroids:

      # ... calculate the distance between the instance and the centroid

      d = calculate_distance(x, centroid)

      #... and add it to the list of distances

      distances_to_centroids.append(d)
```

```
    # when we have visited all centroids,
    # choose the centroid closest to the instance x
    label = np.argmin(distances_to_centroids)
    # conclude that the instance belongs to the cluster
    # determined by that centroid
    cluster_labels.append(label)
  # the result of the function is an array of cluster labels
  return np.array(cluster_labels)
```

Na imagem abaixo, podes ver a primeira iteração da divisão de instâncias em grupos (clusters).

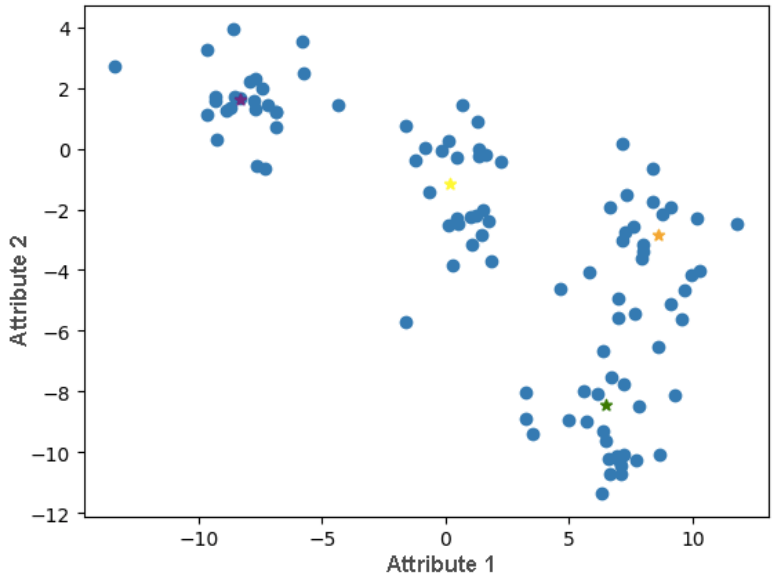


Agora vamos escrever uma função calculate_new_centroids(X, cluster_labels, k) que pode calcular os valores de novos centróides com base na divisão atual de instâncias em grupos (clusters). Os seus argumentos são o conjunto de instâncias X, as etiquetas atuais label_cluster e o número de grupos k**.** Para cada um dos grupos, esta função deve extrair as instâncias que lhe pertencem e, em seguida, calcular a sua média.

```
def calculate_new_centroids(X, cluster_labels, k):
  # initialize the list of new centroids
  new_centroids = []
  # for each cluster
  for i in range(0, k):
    #... extract the instances that belong to it
```

```
    instance_indices = cluster_labels == i

    instances_in_cluster = X[instance_indices]

    # then calculate the new centroid value

    # by averaging all instances in the cluster

    new_centroid = np.average(instances_in_cluster, axis=0)

    # add the calculated new centroid to the list of all centroids

    new_centroids.append(new_centroid)

  # the result of the function is an array of new centroids

  return np.array(new_centroids)
```

Os novos centróides são agora apresentados na imagem abaixo. Vais notar que os centróides dos grupos amarelo e roxo se "afastaram" um do outro.

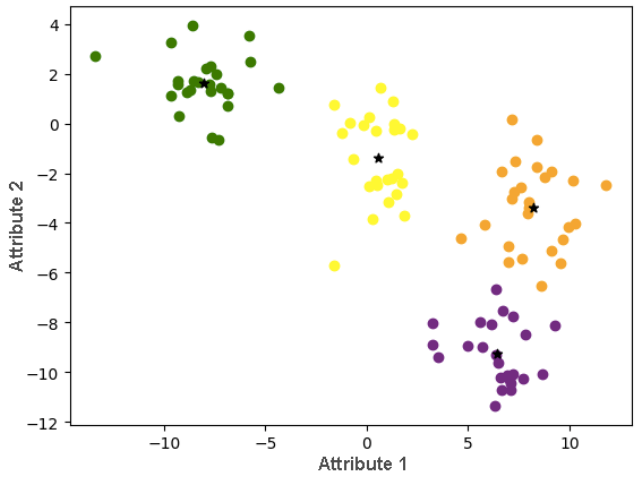


Fica agora por consolidar as tarefas de cada um dos passos numa única função que as repita um número suficiente de vezes. Essa função será chamada izvrsi_klasterovanje(X, k, epsilon = 1e-4, number_of_iterations = 300) em que X representa o conjunto de instâncias, k o número de grupos (clusters), epsilon define a proximidade mínima entre centróides para que o algoritmo pare (isto é, quando os centróides mudam menos do que este valor). Há também um número máximo de iterações max_iterations que adicionalmente fornecemos um critério de paragem adicional.

```
def execute_clustering(X, k, epsilon=1e-4, max_iterations=300):

  # step of initializing centroids

  centroids = generate_centroids(X, k)

  # in each iteration of the loop
```

```
  for i in range(0, max_iterations):

    # step 1: dividing instances into clusters

    cluster_labels = divide_data(X, centroids, k)

    # step 2: calculating new centroids

    new_centroids = calculate_new_centroids(X, cluster_labels, k)

    # checking stopping criteria

    # if they are met, we stop the algorithm

    if np.linalg.norm(new_centroids - centroids) < epsilon:

      break

    # otherwise, we move to the next iteration

    centroids = new_centroids.copy()

  # the result of the function is the final cluster labels and centroid values

  return cluster_labels, new_centroids
```

A execução desta função leva-nos também à divisão final do conjunto de instâncias em grupos (*clusters*), que está representada na figura abaixo.



No caderno de código que acompanha este material, pode também ver uma animação que acompanha esta divisão. Alguns passos dependem de decisões aleatórias (por exemplo, se uma instância estiver igualmente próxima de vários centróides), por isso não se preocupe se alguns valores diferirem ligeiramente.

Referências

Curriculum: Fundamentals of Artificial Intelligence and Machine Learning

https://petlja.org/sr-Latn-RS/kurs/11203/0

https://scikit-learn.org/

„Machine Learning and Artificial Intelligence“, Ameet V Joshi

„Deep Learning“, Ian Goodfellow Yoshua Bengio Aaron Courville

„Machine Learning For Absolute Beginners“, Oliver Theobald

„Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow“, Aurélien Géron

“Artificial Intelligence A Modern Approach”, Stuart J. Russell and Peter Norvig

„Explorations in Artificial Intelligence and Machine Learning“, A CRCPress FreeBook

“The Hundred-Page Machine Learning Book”, Andriy Burkov

“Machine Learning in Action”, Peter Harrington

“Machine Learning A Probabilistic Perspective”, Kevin P. Murphy

“Introduction to Machine Learning with Python”, Andreas C. Müller and Sarah Guido

“Machine Learning Yearning”, Andrew Ng

“Deep Learning with TensorFlow 2 and Keras”, Antonio Gulli, Amita Kapoor, Sujit Pal

“NeuralNetworksandDeepLearning”, CharuC.Aggarwal