Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Ciência de Dados na Nuvem: AWS na Prática

Notas de Aula

Vítor Wilher

AWS · Eletiva

Versão 1.0

Resumo. Serviços da AWS aplicados a ciência de dados: armazenamento, processamento, machine learning e deploy na nuvem.

Palavras-chave: AWS; nuvem; machine learning; deploy

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Ciência de Dados na Nuvem: AWS na Prática** percorre um caminho completo que vai da infraestrutura de computação em nuvem até modelos de aprendizado profundo colocados em produção. O ponto de partida é conceitual e operacional ao mesmo tempo: entender o que é computação em nuvem, quais vantagens econômicas e operacionais ela traz, e como a Amazon Web Services organiza seu catálogo de serviços. Em seguida, o curso desce ao nível prático, com a criação de contas na AWS Academy, a configuração de domínios do Amazon SageMaker AI e a abertura de ambientes JupyterLab hospedados na nuvem.

A partir dessa base, a disciplina trata dos dois pilares mais fundamentais da AWS para ciência de dados: **armazenamento** (com ênfase no Amazon S3, mas também EBS, EFS e Glacier) e **computação** (com ênfase no Amazon EC2, além de contêineres, Kubernetes e funções serverless com AWS Lambda). Esses serviços são exercitados por meio de estudos de caso, incluindo automação por linha de comando com o AWS CLI e infraestrutura como código via CloudFormation.

O eixo seguinte é a linguagem de programação. Duas aulas de Python cobrem os fundamentos necessários para qualquer trabalho de ciência de dados: comentários, indentação, variáveis, tipos, operadores, estruturas condicionais, estruturas de repetição e funções. Com essa base, o curso avança para a análise exploratória de dados, tratando de amostragem, funções densidade de probabilidade, intervalos de confiança, análises gráficas com matplotlib e seaborn, correlação, redução de dimensionalidade (PCA e t-SNE), testes de hipótese e valor-p, além dos modelos de regressão linear e logística.

O último terço da disciplina é dedicado à inteligência artificial propriamente dita. Uma aula introduz a IA, seus ciclos históricos de hype e inverno, e o fluxo de um projeto de machine learning. Depois vêm árvores de regressão e Random Forest, acompanhados do modelo de cobrança da AWS e da criação de endpoints de inferência. As duas aulas seguintes constroem, do zero, a teoria de redes neurais artificiais e dos algoritmos de aprendizado (backpropagation, gradiente descendente, Levenberg-Marquardt). O fechamento é um panorama de Deep Learning, com visão computacional, detecção de objetos e processamento de linguagem natural até os modelos baseados em Transformers.

2 Introdução AWS

A primeira aula estabelece o vocabulário e a arquitetura mental da computação em nuvem, além de conduzir o aluno pela configuração prática do ambiente de trabalho na AWS Academy e no Amazon SageMaker.

2.1 O que é Computação em Nuvem

A definição adotada é direta: **computação em nuvem é a entrega sob demanda de poder computacional, banco de dados, armazenamento, aplicativos e outros recursos de TI por meio de uma plataforma de serviços em nuvem pela Internet, com preços de pagamento conforme o uso**.

A consequência prática é que se evitam altos investimentos iniciais em hardware e o trabalho de gerenciá-lo. Em vez disso, acessa-se rapidamente o recurso certo, no tamanho necessário, pagando apenas pelo que for usado. O material resume essa mudança em uma frase-chave: a computação em nuvem permite que o grupo **pare de lidar com infraestrutura como Hardware e passe a lidar com ela como Software**.

Essa é uma distinção importante. Soluções tratadas como software são flexíveis e podem ser alteradas rapidamente e facilmente, com custo reduzido em comparação a mudanças de hardware. Reprovisionar um servidor virtual é uma chamada de API; trocar um servidor físico é uma operação logística.

2.2 As seis vantagens da computação em nuvem

Os slides listam seis vantagens que sustentam a adoção da nuvem:

1. **Troque despesas de capital por despesas variáveis.** Em vez de investir pesado em data centers e servidores antes mesmo de saber como serão usados, paga-se apenas quando se consome recurso de computação, e somente pelo que for consumido.
2. **Aproveite economias de escala massivas.** Como o uso de centenas de milhares de clientes é agregado na nuvem, provedores como a AWS obtêm economias de escala maiores, o que se traduz em preços mais baixos no modelo "pague conforme o uso".
3. **Pare de adivinhar a capacidade necessária.** Prever capacidade antes do lançamento de uma aplicação costuma produzir recursos ociosos e caros ou capacidade insuficiente. Na nuvem, acessa-se a capacidade exata, aumentando ou reduzindo em poucos minutos.
4. **Aumente a velocidade e a agilidade.** Novos recursos de TI ficam a um clique de distância, reduzindo o tempo de disponibilização de semanas para minutos.
5. **Pare de gastar com operação e manutenção de data centers.** O foco vai para projetos que diferenciam o negócio, não para instalar, empilhar e alimentar servidores.
6. **Vá para o mercado global em minutos.** Implanta-se a aplicação em várias regiões do mundo com poucos cliques, oferecendo menor latência e melhor experiência ao cliente com custo mínimo.

2.3 Modelos de serviço: IaaS, PaaS e SaaS

Com a popularização da nuvem surgiram diferentes modelos para atender necessidades específicas.

Infraestrutura como Serviço (IaaS) contém os blocos básicos da TI em nuvem: acesso a recursos de rede, computadores (virtuais ou em hardware dedicado) e espaço de armazenamento. É o modelo que proporciona o mais alto nível de flexibilidade e controle de gerenciamento, sendo o mais semelhante à infraestrutura tradicional que equipes de TI já conhecem.

Plataforma como Serviço (PaaS) elimina a necessidade de gerenciar a infraestrutura subjacente (tipicamente hardware e sistemas operacionais) e permite concentrar-se na implantação e no gerenciamento das aplicações. Ganha-se eficiência ao não se preocupar com aquisição de recursos, planejamento de capacidade, manutenção de software ou aplicação de patches.

Software como Serviço (SaaS) oferece um produto completo, executado e gerenciado pelo provedor. Trata-se, na maioria dos casos, de aplicações voltadas para o usuário final. O exemplo canônico

é o e-mail baseado na web: envia-se e recebe-se mensagens sem gerenciar servidores ou sistemas operacionais.

2.4 Modelos de implantação

Três modelos de implantação são apresentados:

- **Cloud**: a aplicação é totalmente implantada na nuvem e todas as suas partes executam ali. Pode ter sido criada diretamente na nuvem ou migrada. Pode ser construída sobre componentes de infraestrutura de baixo nível ou usar serviços de nível mais alto, que abstraem gerenciamento, arquitetura e escalabilidade.
- **Híbrido**: conecta infraestrutura e aplicações entre recursos na nuvem e recursos existentes fora dela. O método mais comum é a ligação entre a nuvem e a infraestrutura local (on-premises), expandindo a organização na nuvem enquanto se mantém conexão com os sistemas internos.
- **On-premise**: implantação local com ferramentas de virtualização e gerenciamento de recursos, às vezes chamada de "nuvem privada". Não oferece muitos dos benefícios da nuvem, mas é buscada pela capacidade de fornecer recursos dedicados.

2.5 O catálogo de serviços da AWS

A AWS organiza seus serviços em categorias amplas: Compute, Storage, Database, Networking and Content Delivery, Analytics, Machine Learning, Security/Identity/Compliance, Management and Governance, Cost Management, Developer Tools, Migration and Transfer, Application Integration, Internet of Things, Media Services, entre outras.

Os serviços cobertos no curso incluem:

- **Compute**: Amazon EC2, AWS Lambda, AWS Elastic Beanstalk, Amazon EC2 Auto Scaling, Amazon ECS, Amazon EKS, Amazon ECR, AWS Fargate.
- **Storage**: Amazon S3, Amazon S3 Glacier, Amazon EFS, Amazon EBS.
- **Database**: Amazon RDS, Amazon DynamoDB, Amazon Redshift, Amazon Aurora.
- **Networking**: Amazon VPC, Amazon Route 53, Amazon CloudFront, Elastic Load Balancing.
- **Security e Identity**: AWS IAM, Amazon Cognito, AWS Shield, AWS Artifact, AWS KMS.
- **Management e Governance**: AWS Trusted Advisor, AWS CloudWatch, AWS CloudTrail, AWS Well-Architected Tool, AWS Auto Scaling, AWS CLI, AWS Config, AWS Management Console, AWS Organizations.
- **Cost Management**: AWS Cost and Usage Report, AWS Budgets, AWS Cost Explorer.
- **Machine Learning**: Amazon SageMaker AI e Notebooks.

Um exemplo de solução simples encadeia essas categorias: usuários acessam uma aplicação dentro de uma **Virtual Private Cloud (VPC)**, cujo processamento roda em **Amazon EC2**, com arquivos em **Amazon S3** e dados estruturados em **Amazon DynamoDB**.

2.6 Principais serviços de computação

- **Amazon EC2**: permite criar e gerenciar servidores virtuais (instâncias) na nuvem de forma rápida, segura e escalável, com controle total dos recursos e pagamento pelo uso, ajudando a construir aplicações resistentes a falhas.
- **Amazon EC2 Auto Scaling**: ajusta automaticamente a quantidade de servidores EC2 conforme a demanda. Pode escalar de forma **dinâmica** (reagindo em tempo real) ou **preditiva** (com base em previsões), e ambos podem ser combinados.
- **EC2 Image Builder**: facilita a criação e manutenção de imagens de máquinas virtuais e contêineres, automatizando o processo com segurança. É gratuito, exceto pelos recursos da AWS utilizados.
- **AWS Lambda**: executa código sem servidores, cobrando apenas pelo tempo de execução. Escala automaticamente, é altamente disponível e pode ser ativado por outros serviços ou aplicações.

2.7 Principais serviços de armazenamento

- **Amazon EBS**: armazenamento em bloco persistente para instâncias EC2, com alta disponibilidade e durabilidade por replicação automática, desempenho consistente e de baixa latência, com pagamento pelo que for provisionado.
- **Amazon EFS**: sistema de arquivos elástico e escalável para cargas Linux, que cresce ou diminui automaticamente, oferece acesso simultâneo a milhares de instâncias EC2 com baixa latência e não exige mudanças nas aplicações. É um serviço regional, indicado para migração de aplicações, big data, hospedagem web, desenvolvimento, backups e armazenamento para contêineres.
- **Amazon S3**: armazenamento de objetos altamente escalável, seguro e durável, usado em backups, sites, aplicações, IoT e big data. É projetado para **99,999999999% de durabilidade** (os chamados "onze noves").

2.8 Principais serviços de banco de dados

- **Amazon RDS**: simplifica criação, operação e expansão de bancos relacionais, automatizando provisionamento de hardware, configuração, atualização e backup.
- **Amazon DynamoDB**: banco chave-valor e de documentos com desempenho de milissegundos de um único dígito em qualquer escala, com segurança integrada, backup, restauração e cache em memória.
- **Amazon Aurora**: banco relacional compatível com MySQL e PostgreSQL, até cinco vezes mais rápido que MySQL padrão e três vezes mais rápido que PostgreSQL padrão.
- **Amazon Redshift**: executa consultas analíticas em petabytes de dados armazenados localmente e diretamente em exabytes armazenados no Amazon S3.

2.9 Amazon SageMaker AI e o ambiente de trabalho

O **Amazon SageMaker** oferece acesso a **foundation models** (modelos LLM, ou Large Language Models) por meio de APIs, permitindo integrá-los diretamente nas aplicações. Com poucos ajustes

é possível usar modelos prontos para geração de texto, análise de sentimentos, tradução e outras tarefas, aproveitando a infraestrutura escalável da AWS.

O **JupyterLab do Amazon SageMaker** é um ambiente interativo na nuvem para criar, treinar e testar modelos de machine learning. Permite escrever e executar código Python, visualizar dados, treinar modelos e acompanhar experimentos em notebooks, sem configurar servidores manualmente. Além disso, é possível desenvolver **APIs** para acessar os modelos, programar com o **SDK da AWS** (conjunto de bibliotecas para interagir com os serviços por código, treinando modelos, fazendo deploy e gerenciando recursos) e conectar repositórios do **GitHub** para versionamento e colaboração.

O material menciona ainda o **AWS Backup**, que centraliza e automatiza a proteção de dados em serviços da AWS. É um serviço totalmente gerenciado, baseado em políticas, que ajuda a atender exigências de conformidade regulatória. Com o AWS Organizations é possível implantar centralmente políticas de proteção de dados em toda a organização, cobrindo recursos como EC2, EBS, RDS, DynamoDB e EFS.

2.10 Registro na AWS Academy e criação do Domínio SageMaker

A parte operacional da aula segue um roteiro bem definido. Primeiro, o registro na plataforma **AWS Academy**: cria-se uma conta Canvas com o e-mail que recebeu o convite, define-se uma senha, aceita-se o Termo de Condição e usa-se o botão **Start Lab** para iniciar a sessão de laboratório. Ao iniciar, o indicador carrega e fica verde, e um **temporizador de sessão** começa a contar. Há também o botão para encerrar o Lab.

Em seguida, cria-se o **Domínio do Amazon SageMaker AI**. O roteiro passa por: abrir o console AWS, pesquisar por Amazon SageMaker AI, entrar na seção de **Domínios** e usar o botão de criação. Escolhe-se a opção "Configurar para organizações", dá-se um nome ao domínio, deixa-se em branco a definição de quem usará o SageMaker e seleciona-se a função **LabRole** como papel do IAM. Mensagens relacionadas a problemas de permissão nessa etapa devem ser ignoradas.

Configuram-se então as aplicações. No **JupyterLab**, ativa-se a opção correspondente, define-se o tempo em 60 minutos e o máximo em 600 minutos. No **Canvas**, insere-se um ARN personalizado da função do IAM, no qual entre `iam:` e `:role` fica o ID da conta, isto é, no formato `iam:USER_ID:role`. O ID da conta é encontrado clicando na seta no topo da página do console e copiando o identificador. Para o **Code Editor**, repetem-se os mesmos passos do JupyterLab.

Na configuração de rede, abre-se uma nova guia para verificar as sub-redes, escolhem-se **pelo menos duas sub-redes** e seleciona-se o **grupo de segurança padrão**. Após a revisão final das configurações, envia-se o formulário. A criação do domínio leva tipicamente de cinco a oito minutos, sendo necessário atualizar a guia do navegador ocasionalmente até que o status mude para **Ready**.

Por fim, cria-se um **Perfil de Usuário** dentro do domínio, escolhendo um nome de usuário e novamente a função **LabRole**. Nas etapas seguintes mantém-se a opção "Inherit settings from domain" (herdar configurações do domínio) selecionada, prosseguindo até o envio.

2.11 JupyterLab dentro da AWS

Um caminho alternativo e mais direto para obter um ambiente de notebooks é a seção **Notebooks** do Amazon SageMaker AI. Cria-se uma instância de notebook, define-se um nome e confirma-se

a criação. O status inicial é **Pending**; em média cinco minutos depois muda para **InService**, indicando disponibilidade. Basta então clicar no nome do notebook e abrir o JupyterLab.

3 S3 e EC2

Esta aula aprofunda os dois pilares mencionados na introdução: armazenamento de dados e máquinas virtuais.

3.1 Armazenamento de dados na AWS

Os quatro serviços centrais são:

- Amazon Elastic Block Store (Amazon EBS)
- Amazon Simple Storage Service (Amazon S3)
- Amazon Elastic File System (Amazon EFS)
- Amazon Simple Storage Service Glacier

3.2 Amazon EBS e armazenamento em bloco

O **Amazon EBS** fornece armazenamento em bloco altamente disponível e durável para uso com instâncias do Amazon EC2. Um ponto essencial: **os dados continuam existindo mesmo após o dispositivo (a instância EC2) ser desligado**, o que caracteriza a persistência do volume.

Armazenamento em bloco é uma forma de armazenamento na qual os arquivos são divididos em blocos individuais, que podem ser armazenados separadamente. A principal diferença em relação ao armazenamento em objeto está em como os dados são manipulados:

- No **armazenamento em bloco**, é possível alterar apenas uma parte (um bloco) do arquivo.
- No **armazenamento em objeto**, é necessário atualizar o arquivo inteiro, mesmo para pequenas mudanças.

Essa diferença afeta diretamente desempenho, latência e custo. Bloco é mais rápido e usa menos banda, mas pode custar mais; objeto é mais barato, porém menos eficiente para alterações pequenas. Uma analogia útil dos slides: o Amazon EBS funciona como um HD ou SSD do computador, sendo onde ficam instalados o sistema operacional e os arquivos de que o servidor precisa para funcionar.

3.3 Amazon S3 e suas classes de armazenamento

O **Amazon S3 é object-level storage**, ou seja, armazenamento no nível de objeto. As classes de armazenamento disponíveis atendem a padrões de acesso diferentes:

- **S3 Standard**: para dados acessados com frequência, com alta durabilidade, disponibilidade e desempenho. Ideal para aplicações em nuvem, sites dinâmicos, distribuição de conteúdo, aplicativos móveis, jogos e big data.

- **S3 Intelligent-Tiering**: otimiza custos automaticamente movendo os dados entre camadas de acesso frequente e infrequente. Indicado para dados de longo prazo com padrões de acesso imprevisíveis.
- **S3 Standard-IA (Infrequent Access)**: para dados acessados raramente, mas que precisam de acesso rápido quando necessário. Mais barato que o Standard, porém cobra taxa na recuperação. Bom para backups e recuperação de desastres.
- **S3 One Zone-IA**: armazena os dados em apenas uma zona de disponibilidade, em vez de três. Mais barato, mas com menor resiliência. Bom para backups secundários ou dados que podem ser recriados facilmente.
- **S3 Glacier**: armazenamento seguro, barato e durável para arquivamento de longo prazo. A recuperação pode levar minutos ou horas. Alternativa econômica ao armazenamento local.
- **S3 Glacier Deep Archive**: a opção mais barata do S3, feita para retenção de sete a dez anos ou mais, típica de setores regulados como finanças e saúde. Os dados são replicados em três zonas, a recuperação leva até 12 horas e a durabilidade é de onze noves, praticamente sem risco de perda.

3.4 Amazon EC2 e a família de serviços de computação

Na AWS, uma **instância EC2 é uma máquina virtual que roda na nuvem**, permitindo ter servidores sob demanda para executar aplicações, testar sistemas ou hospedar serviços sem infraestrutura física própria.

A família de serviços de computação relacionada inclui:

- **Amazon EC2**: fornece máquinas virtuais redimensionáveis.
- **Amazon EC2 Auto Scaling**: garante disponibilidade da aplicação, permitindo definir condições para iniciar ou encerrar instâncias automaticamente.
- **Amazon ECR (Elastic Container Registry)**: armazena e recupera imagens Docker.
- **Amazon ECS (Elastic Container Service)**: serviço de orquestração de contêineres que suporta Docker.
- **VMware Cloud on AWS**: provisiona nuvem híbrida sem necessidade de hardware personalizado.
- **AWS Elastic Beanstalk**: forma simples de executar e gerenciar aplicações web.
- **AWS Lambda**: computação serverless, pagando apenas pelo tempo de execução utilizado.
- **Amazon EKS (Elastic Kubernetes Service)**: executa Kubernetes gerenciado na AWS.
- **Amazon Lightsail**: serviço simples para criar aplicações ou sites.
- **AWS Batch**: executa batch jobs em qualquer escala.
- **AWS Fargate**: executa contêineres sem gerenciar servidores ou clusters.
- **AWS Outposts**: executa serviços selecionados da AWS em data centers locais.
- **AWS Serverless Application Repository**: descobre, implanta e publica aplicações serverless.

Sobre orquestração, o material define: **Kubernetes é uma plataforma que organiza e gerencia contêineres, garantindo que as aplicações rodem de forma estável, escalável e automática, sem que seja necessário controlar servidor por servidor**.

3.5 Estudo de caso: AWS Lambda e a função myStopinator

O primeiro estudo de caso vem do módulo 6 do AWS Academy Cloud Foundations e consiste em criar uma função Lambda que interrompe automaticamente uma instância EC2. O roteiro é:

1. Pesquisar por **Lambda** na caixa de serviços e escolher "Criar uma função".
2. Selecionar "Criar do zero", com nome da função `myStopinator`, runtime **Python 3.11**, alterando a função de execução padrão para usar uma função existente chamada `myStopinatorRole`.
3. Criar a função.
4. Adicionar um **trigger** do tipo **EventBridge (CloudWatch Events)**, criando uma nova regra chamada `everyMinute`, do tipo "Expressão de programação", com a expressão `rate(1 minute)`.
5. No painel de código, abrir `lambda_function.py`, apagar o código existente e colar a implementação.

O código utilizado é:

```
import boto3

region = '<REPLACE_WITH_REGION>'
instances = ['<REPLACE_WITH_INSTANCE_ID>']
ec2 = boto3.client('ec2', region_name=region)

def lambda_handler(event, context):
    ec2.stop_instances(InstanceIds=instances)
    print('stopped your instances: ' + str(instances))
```

O marcador de região deve ser substituído pelo código da região realmente utilizada, obtido no canto superior direito do console. Por exemplo, o código da Região Leste dos EUA (Norte da Virgínia) é `us-east-1`, e deve permanecer entre aspas simples. O marcador do identificador da instância é substituído pelo ID real, copiado na guia de detalhes da instância chamada `instance1` no console do EC2 (o identificador começa com `i-`), também entre aspas simples.

Depois de salvar e escolher **Implantar**, a função passa a tentar interromper a instância a cada minuto. Na aba **Monitorar**, um dos gráficos mostra quantas vezes a função foi invocada, e outro exibe a contagem de erros e a taxa de sucesso em porcentagem. Esse exercício ilustra bem a lógica serverless: código curto, disparado por evento, sem servidor a administrar, e observável por métricas nativas.

3.6 Estudo de caso: EC2 via AWS CLI e CloudFormation

O segundo estudo de caso monta uma arquitetura completa a partir de um repositório público, `https://github.com/PAlejandroQ/Aws_Arquitecture`, exigindo a instalação do **AWS CLI** e do **VS Code**.

A configuração de credenciais é feita criando uma pasta `.aws` na raiz do usuário (por exemplo, `C:\Users\vitor\.aws` no Windows ou `/Users/name-user/.aws` no macOS e Linux), contendo dois arquivos sem extensão: `credentials` e `config`. O arquivo `config` tem o seguinte formato:

```
[default]
region = us-east-1
output = json
```

O arquivo `credentials` recebe a chave do AWS CLI obtida em "AWS Details" no laboratório. Alternativamente, executa-se `aws configure`.

No Windows, com PowerShell, os slides indicam preparar o ambiente com:

```
Set-ExecutionPolicy RemoteSigned -Scope Process
$PSVersionTable
```

E então executar os scripts na ordem: `.\cleanup-cfn.ps1` (caso já tenha sido executado antes), `deploy-cfn.ps1` e `.\ssh-connect.ps1`.

Em Linux ou macOS, o fluxo equivalente é:

```
cleanup-cfn.sh
deploy-cfn.sh
chmod 400 RAG-Key-CFN.pem
```

A conexão à instância provisionada usa a saída da pilha do CloudFormation para descobrir o IP público:

```
PUBLIC_IP=$(aws cloudformation describe-stacks \
  --stack-name RAG-Stack-CFN \
  --query 'Stacks[0].Outputs[?OutputKey==`PublicIP`].OutputValue' \
  --output text)

ssh -i RAG-Key-CFN.pem ubuntu@$PUBLIC_IP
```

Esse exercício reúne três ideias importantes: **infraestrutura como código** (a pilha CloudFormation descreve os recursos), **automação por CLI** (a consulta ao stack extrai dinamicamente o IP) e **acesso seguro** (a chave `.pem` com permissão restrita via `chmod 400`).

4 Programação em Python - Parte-A

As aulas de Python são ministradas pela Profa. Manoela Kohler e cobrem, em duas partes, o mesmo conjunto de slides introdutórios. Esta primeira parte concentra os fundamentos: filosofia da linguagem, conceitos básicos, tipos de dados e operadores.

4.1 Lógica e algoritmos

O material parte de uma reflexão conceitual. A lógica é a parte da filosofia que trata das formas do pensamento em geral e das operações intelectuais que visam determinar o que é verdadeiro ou não. No universo da tecnologia da informação, a lógica é a organização e o planejamento das instruções e assertivas em um **algoritmo**, a fim de viabilizar a implantação de um programa. Programar, portanto, **nada mais é do que a organização coesa de uma sequência de instruções voltadas à resolução de um problema de forma lógica**.

4.2 Filosofia e características do Python

A escolha do Python é justificada por um conjunto de atributos: totalmente gratuito, com boa usabilidade, orientado a objetos, poderoso, com ferramentas gráficas poderosas, flexível e apoiado por uma vasta comunidade. Complementarmente, os slides destacam que a linguagem é **open-source**, tem boa **compatibilidade**, é usada mundialmente pela academia e pela indústria, e está sempre atualizada com novas bibliotecas de uso livre.

Quanto a ambientes de desenvolvimento, são citados Jupyter, Spyder, PyCharm e VS Code. A conclusão do material sobre qual escolher é pragmática: **a melhor IDE é aquela em que você se sente mais à vontade ao utilizar**.

4.3 Comentários

Quando um programa cresce e se torna mais complicado, fica difícil de ler e manter. Por isso, é boa prática inserir documentação ou notas no código, chamadas de **comentários**. O comentário em Python começa com o sinal de hash ou tralha (`#`) e continua até o final da linha. O interpretador Python ignora comentários ao interpretar o código.

O material distingue três formas de uso: comentário em bloco, comentário em linha e comentário para documentação de funções, módulos, pacotes e classes.

```
# Comentario em bloco: explica o trecho seguinte
x = 10  # comentario em linha

def area(raio):
    """Comentario de documentacao (docstring) da funcao."""
    return 3.14159 * raio ** 2
```

4.4 Indentação

Python usa **indentação** para delimitar blocos, em vez de chaves. Tabs e espaços são suportados. Essa é uma característica que torna a estrutura visual do código inseparável de sua semântica.

4.5 Tipos de dados

O tipo **numérico** representa dados com valor numérico, podendo ser inteiro, número flutuante ou número complexo:

- **Integers**: representados pela classe `int`, contendo números inteiros positivos ou negativos, sem fração ou decimal.
- **Float**: representados pela classe `float`, número real com representação de ponto flutuante, especificado por um ponto decimal.
- **Números complexos**: representados pela classe complexa, especificados como parte real mais parte imaginária seguida de `j`.

O tipo **booleano** possui um de dois valores internos, Verdadeiro ou Falso, indicado pela classe `bool`.

As **sequências** são coleções ordenadas de tipos de dados semelhantes ou diferentes, permitindo armazenar vários valores de forma organizada e eficiente. Existem três tipos principais:

- **Strings**: matrizes de bytes que representam caracteres. Uma string é uma coleção de um ou mais caracteres entre aspas simples, duplas ou triplas, representada pela classe `str`. É possível acessar elementos individuais por índice.
- **Listas**: semelhantes a matrizes de outras linguagens, mas não precisam ser homogêneas, podendo conter inteiros, strings e objetos na mesma estrutura. São **mutáveis**, ordenadas, têm contagem definida e são representadas pela classe `list`.
- **Tuplas**: coleção ordenada de objetos Python, semelhante à lista, com valores de qualquer tipo indexados por inteiros. A diferença importante é que **tuplas são imutáveis**. Representadas pela classe `tuple`.

O **set** é uma coleção não ordenada, iterável, mutável e **sem elementos duplicados**. A ordem dos elementos é indefinida. A principal vantagem em relação à lista é possuir um método altamente otimizado para verificar se um elemento específico está contido no conjunto.

O **dicionário** é uma coleção não ordenada de valores de dados, usada para armazenar dados como um mapa. Diferentemente de outros tipos que mantêm apenas um valor único como elemento, o dicionário mantém um par **chave:valor**. A chave e o valor são separados por dois pontos, enquanto cada par é separado por vírgula.

```
inteiro = 6
flutuante = 2.5
booleano = True
texto = "The Shining"
lista = [4.5, 4.0, 5.0]
tupla = (1, 2, 3)
conjunto = {1, 2, 3}
dicionario = {"filme": "The Shining", "scores": [4.5, 4.0, 5.0]}
```

4.6 Operadores, type casting e interação com o usuário

Além dos tipos, a aula cobre **operadores** (aritméticos, de comparação, lógicos), **type casting** (conversão explícita entre tipos) e **interação com o usuário** por meio da função `input()`.

4.7 Exercícios propostos

Os exercícios reforçam esses conceitos:

1. Criar uma variável com valor 6 (inteiro) e outra com valor 2 (inteiro), dividir a primeira pela segunda salvando em nova variável e observar o tipo do resultado, tentando entender o motivo do tipo obtido. O ponto pedagógico aqui é que a divisão em Python retorna `float` mesmo entre inteiros.

2. Dois amigos dividem o lucro obtido através de um site de consultoria para projetos de IA. O lucro mensal é de 8 mil; o amigo 1 tem direito a 30% e o restante pertence ao amigo 2. Calcular o lucro total do ano e o lucro de cada um.
3. Prever mentalmente o resultado da expressão `5 + 3 * 10 / 3 == 15` e confirmar em Python. O exercício testa a compreensão da precedência de operadores e da divisão em ponto flutuante.

Um segundo bloco usa dados de um filme:

```
movieName = "The Shining"
Actors = ["Jack Nicholson", "Shelley Duvall", "Danny Lloyd",
          "Scatman Crothers", "Barry Nelson"]
scores = [4.5, 4.0, 5.0]
comments = ["Best Horror Film I Have Ever Seen",
            "A truly brilliant and scary film from Stanley Kubrick",
            "A masterpiece of psychological horror"]
```

Pede-se criar uma lista com os quatro elementos, imprimir a string com o nome do primeiro ator e imprimir a melhor avaliação do filme (score e comentário). As dicas fornecidas são `max(lista)`, que retorna o maior valor de uma lista, e `lista.index(valor)`, que retorna o índice desse valor. Ao final, pede-se refazer o exercício salvando as variáveis em um dicionário em vez de uma lista.

Um terceiro exercício pede um script que pergunte a idade do usuário com `input()`, imprima a idade, o tipo e o ano de nascimento (considerando que a pessoa já fez aniversário no ano), funcionando para qualquer ano em que o script venha a ser executado. A dica aponta o pacote `datetime`:

```
from datetime import datetime

datetime.now()        # data e hora corrente
datetime.now().year   # ano corrente
```

5 Programação em Python - Parte-B

A segunda parte do módulo de Python retoma o mesmo conjunto de slides e avança para as estruturas de controle e para as funções, encerrando com uma sequência de exercícios centrada na Lei dos Grandes Números.

5.1 Estruturas condicionais

As estruturas condicionais permitem que o fluxo do programa siga caminhos distintos conforme o resultado de um teste lógico. Combinadas aos operadores de comparação e lógicos vistos na parte A, elas são a base da tomada de decisão em qualquer algoritmo.

```
idade = 20

if idade < 18:
    print("Menor de idade")
elif idade < 65:
    print("Adulto")
else:
```

```
    print("Idoso")
```

5.2 Estruturas de repetição

Duas construções são apresentadas: o **while loop** e o **for loop**. O `while` repete enquanto uma condição permanece verdadeira; o `for` percorre os elementos de uma sequência.

```
# for loop
for i in range(25):
    print(i)

# while loop
i = 0
while i < 25:
    print(i)
    i = i + 1
```

5.3 Exercícios com estruturas de repetição

A lista de exercícios é extensa e cuidadosamente construída para exercitar tanto a solução iterativa quanto a solução vetorizada com NumPy:

1. Imprimir uma sequência de 25 números consecutivos, usando uma estrutura de repetição e também sem usar estrutura de repetição.
2. Imprimir a sequência acima duas vezes, com a dica de aninhar um `for` dentro de outro `for` (ou `while` dentro de `while`).
3. Na lista `[12, 23, 11, 34, 13, 56, 102, 101, 13]`, imprimir somente os números pares. A dica é o operador de resto da divisão, `%` (por exemplo, `10 % 2 = 0`), combinado ao operador de comparação `==`.
4. Simular a Mega Sena: seis números entre 1 e 60 em um sorteio, com e sem estrutura de repetição. A dica sem repetição é `np.random.randint()` do NumPy. Há aqui uma pergunta importante: uma estrutura de repetição sem nenhum tratamento e `np.random.randint` sem nenhum tratamento podem gerar algum problema? A resposta implícita é a possibilidade de números repetidos, e a dica para contorná-la é `np.random.choice()`.
5. Simular o resultado de um dado de seis faces jogado sete vezes, com e sem estrutura de repetição.
6. Calcular o fatorial de um número qualquer, com e sem estrutura de repetição, usando também o pacote `math`. A definição relembrada é que o fatorial de 5 é 5 x 4 x 3 x 2 x 1 = 120.

O exemplo de uso apresentado é `np.random.randint(1, 61, 1)`, que sorteia um número entre 1 e 60.

5.4 A Lei dos Grandes Números

Um exercício conceitualmente rico fecha a seção. A **Lei dos Grandes Números** afirma que a frequência relativa observada converge para a probabilidade teórica conforme o número de experimentos cresce. Os slides ilustram isso com resultados de lançamentos:

- 10 experimentos: 7 contra 3, isto é, 70% e 30%
- 100 experimentos: 52 contra 48, isto é, 52% e 48%
- 1000 experimentos: 502 contra 498, isto é, 50,2% e 49,8%

A tendência é clara: quanto maior o número de experimentos, mais próxima a proporção observada fica do valor esperado de 50%.

O exercício propõe testar a lei para N números aleatórios gerados com distribuição normal de média 0 e desvio padrão 1. Cria-se um script que conta quantos desses números caem entre -1 e 1 e divide por N. Sabe-se que o valor esperado $E(X)$ é 68,2%, e verifica-se que a média amostral tende a esse valor conforme N cresce.

```
import numpy as np

def lei_grandes_numeros(n, inicio=-1, fim=1):
    amostra = np.random.normal(0, 1, n)
    dentro = sum(1 for x in amostra if inicio < x < fim)
    return dentro / n

print(lei_grandes_numeros(10))
print(lei_grandes_numeros(1000))
print(lei_grandes_numeros(1000000))
```

5.5 Funções

As funções são apresentadas com três componentes: o **retorno**, o nome da **função** e os **parâmetros de entrada**. Uma regra operacional importante é enunciada: **a função deve ser declarada antes de sua chamada**.

O material explica a motivação: as funções são o primeiro passo para a reutilização de código. Elas permitem definir um bloco de código reutilizável que pode ser usado repetidamente em um programa. O Python fornece várias funções internas, como `print()` e `len()`, mas também é possível definir funções próprias.

5.6 Exercícios com funções

1. Criar uma função que calcule o fatorial de um número qualquer.
2. Usar a função `factorial` do pacote `math` para calcular o fatorial.
3. Criar uma função que receba o raio de um círculo e retorne a área, utilizando o pacote `math`. As dicas são `math.pi` e o operador de potência `**` ou `math.pow`.
4. Alterar a função anterior para retornar, além da área, o diâmetro e o perímetro, com valor padrão de raio igual a 5 e retorno via dicionário.

```
import math

def circulo(raio=5):
    return {
        "area": math.pi * raio ** 2,
        "diametro": 2 * raio,
        "perimetro": 2 * math.pi * raio,
```

```
    }

print(circulo())
print(circulo(3))
```

O exercício final integra funções e a Lei dos Grandes Números: criar uma função que receba como parâmetro o número de experimentos e calcule a média dos experimentos no intervalo fixo de -1 a 1, testando para n igual a 10, 1000 e 1000000. Após a validação, incluir um ou dois **parâmetros opcionais** de intervalo, com valores padrão -1 e 1, e testar outros intervalos.

6 Análise Exploratória - Ciência de Dados Começa

Esta aula é a ponte entre a infraestrutura e a modelagem. Ela trata dos fundamentos estatísticos e gráficos necessários para entender um conjunto de dados antes de modelá-lo.

6.1 Dataset como amostra

O ponto de partida é epistemológico: **um dataset é uma amostra do universo de dados que se quer estudar**. Por ser uma representação parcial, está sujeito a variações aleatórias que podem distorcer a percepção do comportamento geral dos dados.

Duas estratégias de amostragem são citadas: a **amostragem simples** e a **amostragem estratificada**. Apesar de o dataset representar apenas uma amostra do universo real, a análise exploratória fornece insights valiosos sobre distribuição, variabilidade, correlações e possíveis vieses presentes nos dados.

6.2 Função densidade de probabilidade e intervalo de confiança

A **função densidade de probabilidade** descreve como a probabilidade se distribui ao longo dos valores possíveis de uma variável contínua, sendo a base para raciocinar sobre variabilidade amostral.

O **intervalo de confiança** de (100 menos alfa) por cento para um parâmetro consiste em um intervalo aleatório que possui a propriedade de conter o valor real desse parâmetro com probabilidade de (100 menos alfa) por cento. A ênfase na aleatoriedade do intervalo, e não do parâmetro, é a interpretação correta do conceito.

6.3 Análises gráficas

As bibliotecas usadas são **matplotlib** e **seaborn**. Os tipos de gráfico cobertos são:

- **Boxplot**
- **Histogramas** e gráficos de **densidade**
- **Gráficos em barra**
- **Gráficos de dispersão**
- **Matriz de correlação**

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("Islander_data.csv")

df["Diff"].hist(bins=30)
plt.show()

sns.boxplot(x="Drug", y="Diff", data=df)
plt.show()

sns.heatmap(df.corr(numeric_only=True), annot=True)
plt.show()
```

6.4 Correlação e covariância

A **correlação indica a força e a direção do relacionamento entre dois atributos**. É uma medida da relação entre dois atributos, mas o material faz uma advertência central: **correlação não implica causalidade**. Duas variáveis podem estar altamente correlacionadas sem que exista relação de causa e efeito entre elas.

Tecnicamente, a correlação mede o grau e o sentido da relação **linear** entre duas variáveis numéricas. O coeficiente mais usado é o **coeficiente de correlação de Pearson**, definido a partir da covariância entre X e Y dividida pelo produto dos desvios-padrão de X e Y. A **covariância** mede como duas variáveis variam juntas.

O resultado varia de -1 a +1:

- **+1**: correlação linear perfeita positiva, isto é, as variáveis crescem juntas.
- **-1**: correlação linear perfeita negativa, isto é, uma cresce enquanto a outra diminui.
- **0**: sem relação linear.

6.5 Redução de dimensionalidade

Uma forma comum de aprendizado não supervisionado é a **redução de dimensionalidade**, na qual se aprende uma função de mapeamento do espaço visível de alta dimensão para um **espaço latente** de baixa dimensão.

Esse mapeamento pode ser:

- **Paramétrico**, no qual existe uma função $z = f(x)$ que pode ser aplicada a qualquer entrada;
- **Não paramétrico**, no qual se calcula um vetor de embedding apenas para cada ponto do conjunto de dados, mas não para outros pontos fora dele.

A abordagem não paramétrica é usada principalmente para **visualização de dados**, enquanto a paramétrica também pode ser utilizada como etapa de **pré-processamento** para outros algoritmos de aprendizado.

A forma mais simples e amplamente utilizada de redução de dimensionalidade é a **análise de componentes principais (PCA)**.

O **t-SNE** é apresentado como alternativa. A ideia básica do SNE é **converter as distâncias euclidianas de alta dimensão em probabilidades condicionais que representam semelhanças**. Isso permite preservar a estrutura de vizinhança local ao projetar os dados em duas ou três dimensões para visualização.

6.6 Teste de hipótese nula e nível de significância

Para verificar se há diferença nas médias de dois modelos ou grupos, assume-se inicialmente que **não há diferença real**. Essa é a **hipótese nula**: qualquer diferença observada seria explicada apenas pelo acaso.

Se a hipótese nula for verdadeira, ou seja, se não houver diferença entre as amostras, pode-se considerar que ambas vêm da mesma população. Nesse cenário, a maioria das diferenças de médias ficará próxima de zero, mas diferenças maiores podem ocorrer ao acaso, caindo na região improvável da distribuição.

Essa região improvável é chamada **região crítica**, e geralmente corresponde a 5%. Esse é o **nível de significância** do teste. Outros valores poderiam ser usados, como 1% ou 10%, mas 5% é o mais comumente adotado.

A regra de decisão é direta:

- Se a diferença das médias cair na região crítica, **rejeita-se a hipótese nula**, pois a chance de isso ocorrer ao acaso é baixa, indicando que a diferença é **estatisticamente significativa**.
- Se a diferença não estiver na região crítica, **não se rejeita a hipótese nula** de igualdade das médias. Considera-se que a diferença observada foi devida ao acaso e, portanto, não é estatisticamente significativa.

6.7 Valor-p

O **valor-p unilateral superior** indica a probabilidade de se obter, supondo que a hipótese nula é verdadeira, um valor de diferença de médias igual ou superior ao valor observado no estudo. Em um teste bilateral, o valor-p é o dobro do valor-p unilateral superior (ou do unilateral inferior, o que for menor).

A regra final: **se o valor-p for menor ou igual ao nível de significância, a hipótese nula é rejeitada; caso contrário, ela não é rejeitada**.

6.8 Regressão logística e linear

A **regressão logística** é um modelo de classificação **discriminativo** amplamente utilizado, que modela a probabilidade condicional do rótulo dado o vetor de entrada e os parâmetros. Nessa formulação, x é um vetor de entrada de dimensão fixa, y é o rótulo da classe pertencente a um conjunto de C classes possíveis, e os parâmetros são estimados a partir dos dados.

- Se C igual a 2, o modelo é conhecido como **regressão logística binária**. A combinação linear das entradas é dada por $a = w^T x + b$, sobre a qual se aplica a função logística.

- Se C maior que 2, é conhecido como **regressão logística multinomial**, ou regressão logística multiclasse.

Tanto no caso binário quanto no multinomial, e também na **regressão linear**, define-se uma **função objetivo** (objective function) a ser otimizada, o que conecta esta aula diretamente à discussão de otimizadores que aparece mais adiante no curso.

6.9 Estudo de caso: medicamento para ansiedade

O primeiro estudo de caso usa dados de um experimento sobre os efeitos de medicamentos para ansiedade e seu impacto em grupos que têm majoritariamente lembranças felizes ou tristes. As drogas e dosagens são:

- **A - Alprazolam** (Xanax, longo prazo): 1 mg, 3 mg, 5 mg
- **T - Triazolam** (Halcion, curto prazo): 0,25 mg, 0,5 mg, 0,75 mg
- **S - Sugar Tablet** (placebo): 1, 2 ou 3 comprimidos

A tarefa é carregar a base `Islander_data.csv` e fazer, livremente, análises gráficas para entendimento dos dados utilizando matplotlib e seaborn.

6.10 Estudo de caso: Tendências Mundiais

O segundo estudo de caso pede carregar a base `tendenciasMundiais.csv` em um dataframe e mostrar graficamente a relação entre **Expectativa de Vida** e **Taxa de Fertilidade** por país para os anos de 1960 e 2013, em um mesmo gráfico, utilizando a função `scatter`. Pede-se comparar e avaliar os dois anos, incluindo legenda, título e nomes dos eixos, e explorar livremente outros gráficos aprendidos na aula, como boxplot, violinplot e histograma.

7 Introdução à Inteligência Artificial

Esta aula, ministrada pela Profa. Manoela Kohler, oferece a fundamentação conceitual e histórica da área. As notas dos slides vêm parcialmente do curso "Artificial Intelligence 501" formulado pela Intel.

7.1 Objetivos de aprendizagem

Ao final da aula, espera-se ser capaz de: definir Inteligência Artificial, incluindo Machine Learning e Deep Learning; explicar como o Deep Learning ajuda a resolver limitações clássicas de ML; explicar desenvolvimentos históricos importantes e os ciclos de inverno e hype da IA; diferenciar IA moderna de IA clássica; entender possibilidades de aplicação da IA; compreender o fluxo de um projeto de ML; e reconhecer equívocos comuns na área.

7.2 Definições encaixadas

A estrutura conceitual é de círculos concêntricos: **Inteligência Artificial** é o conjunto mais amplo; dentro dele está o **Aprendizado de Máquina** (Machine Learning); dentro deste, as **Redes Neurais**; e, no núcleo, a **Aprendizagem Profunda** (Deep Learning).

7.3 Pombos como especialistas em arte

Um experimento marcante ilustra o que significa aprender. No estudo de Watanabe sobre discriminação de pinturas por pombos e humanos, pombos foram colocados em caixas de Skinner e apresentados a pinturas de dois artistas, Chagall e Van Gogh, recebendo recompensa ao bicar o botão quando a pintura era de Van Gogh.

Os resultados: os pombos discriminaram os quadros com **95% de acurácia** quando apresentados a pinturas usadas no treinamento, e a discriminação ficou em **85% de sucesso** para pinturas nunca vistas antes.

A conclusão é o cerne da ideia de aprendizado: os pombos não simplesmente memorizam as pinturas; eles extraem e reconhecem padrões, no caso, o estilo; e são capazes de **generalizar** a partir do que já foi visto para tomar decisões. Essa é uma capacidade humana e da inteligência artificial, que um computador convencional não possui.

7.4 Machine Learning e suas limitações

A definição operacional de Machine Learning é: **estes algoritmos aprendem ao ver repetidamente um conjunto de dados, em vez de serem explicitamente programados por seres humanos**.

Um exemplo clássico de terminologia é a base de classificação de espécies de flores, na qual cada linha é uma observação e cada coluna é uma característica (feature).

O exemplo aplicado é a identificação de transações fraudulentas de cartão de crédito. Seria necessário identificar características para que o algoritmo aprenda que combinações delas sugerem atividade incomum:

- Hora da transação
- Valor da transação
- Localização
- Categoria da compra

A limitação aparece no exemplo seguinte: suponha que se queira determinar se uma imagem é de um gato ou de um cão. Que características usar? Não há uma lista óbvia de atributos tabulares que separe as duas classes. **É aí que entra o Deep Learning**, capaz de aprender as próprias representações em **diferentes níveis de abstração**, dispensando a engenharia manual de características.

7.5 História da IA

1950, os primórdios. Em 1950, Alan Turing desenvolveu o teste de Turing para avaliar a capacidade das máquinas de apresentar comportamento inteligente. Em 1956, a Inteligência Artificial foi aceita como campo na Conferência de Dartmouth. Em 1957, Frank Rosenblatt inventou o algoritmo **perceptron**, precursor das redes neurais modernas. Em 1959, Arthur Samuel publicou um algoritmo para um programa de damas usando aprendizado de máquina.

O primeiro inverno da IA. Em 1966, o comitê ALPAC avaliou técnicas de IA para tradução de máquina e determinou que o retorno não compensou o investimento. A partir de 1969, Marvin Minsky publicou um livro sobre as limitações do algoritmo perceptron, que retardou a pesquisa em redes neurais. Em 1973, o relatório Lighthill destacou o fracasso da IA. Os dois relatórios levaram a cortes no financiamento governamental para pesquisa em IA, produzindo o primeiro **AI Winter**.

O boom dos anos 1980. Surgem os **sistemas especialistas**, sistemas com regras programadas concebidas para imitar especialistas humanos. Rodavam em mainframes com linguagens de programação especializadas, por exemplo LISP. Foram as primeiras tecnologias de IA amplamente utilizadas, com dois terços das empresas da Fortune 500 usando sistemas especialistas em seu pico. Em 1986, o algoritmo de **backpropagation** mostrou-se capaz de treinar perceptrons de múltiplas camadas, o que trouxe novos sucessos e interesse à pesquisa em redes neurais artificiais.

Outro AI Winter, final dos anos 1980 e início dos 1990. O progresso dos sistemas especialistas na resolução de problemas de negócios diminuiu. Esses sistemas passaram a ser fundidos em aplicativos de negócios gerais, como SAP e Oracle, que podiam funcionar em PCs em vez de mainframes. As redes neurais não escalaram para problemas maiores e mais complexos, e o interesse em IA nos negócios diminuiu.

Aprendizado de máquina clássico, final dos anos 1990 e início dos 2000. Avanços no algoritmo **SVM** levaram-no a se tornar o método de escolha. Soluções em IA tiveram sucesso em reconhecimento de voz, diagnóstico médico, robótica e outras áreas. Algoritmos de IA foram integrados em sistemas maiores e se tornaram úteis em toda a indústria. O Deep Blue derrotou o campeão mundial de xadrez Garry Kasparov, e o motor de busca do Google foi lançado usando inteligência artificial.

A ascensão do Deep Learning, a partir de 2006. Em 2006, Geoffrey Hinton publicou um artigo sobre pré-treinamento sem supervisão que permitiu treinar redes neurais mais profundas, e o foco de estudo migrou para a aprendizagem profunda. Em 2009, o banco de dados **ImageNet** de imagens rotuladas por humanos foi apresentado na conferência CVPR. Em 2010, algoritmos competiram em tarefas de reconhecimento visual na primeira competição ImageNet.

IA moderna, de 2012 até o presente. Em 2012, o Deep Learning bateu benchmarks anteriores na competição ImageNet. Em 2013, foi usado para entender o significado conceitual de palavras. Em 2014, avanços semelhantes apareceram na área de tradução, o que levou a progressos em pesquisa na web, pesquisa em documentos, resumo de documentos e tradução de máquina; no mesmo ano, algoritmos de visão computacional passaram a descrever fotos. Em 2015, o framework **TensorFlow** foi desenvolvido. Em 2016, o **AlphaGo** da DeepMind, desenvolvido por Aja Huang, derrotou Lee Se-dol, o maior campeão de Go. Em 2023, a xAI foi fundada por Elon Musk, em março.

Para 2024 e 2025, o material lista os temas dominantes: **LLM**, **SLM**, **VLM**, modelos multimodais, **Foundation Models**, **agentes de IA**, grande evolução de modelos proprietários, DeepSeek,

iniciativas open-source, arquiteturas **MoE** (mistura de especialistas), além de **ética** e **regulamentação**.

7.6 Fluxo de um projeto de Machine Learning

Os slides apresentam um fluxograma do trabalho típico em projetos de ML, encadeando as etapas desde a definição do problema até a produção. A discussão sobre esse fluxo prepara o terreno para os equívocos comuns apresentados a seguir.

7.7 Seis equívocos comuns

Equívoco 1: o unicórnio. Cientistas de dados que são especialistas em todas as áreas são chamados de unicórnios. Na prática, equipes bem-sucedidas contêm pessoas de diversas áreas e formações, com várias habilidades: alguns se distinguem em comunicação, outros se sobressaem em estatística. Equipes bem-sucedidas têm especialistas em três áreas principais: **negócios, ciência e engenharia**.

Equívoco 2: foco apenas em pesquisa e algoritmos. Equipes de ciência de dados não podem focar apenas em pesquisa e algoritmos. Boas equipes conseguem identificar problemas, comunicar suas descobertas e trabalhar com a engenharia para achar o melhor método de produção.

Equívoco 3: a solução mais complicada é a melhor. Equipes tendem a ser mais bem-sucedidas quando começam do básico e depois passam para técnicas mais complexas de modelagem. Modelos complexos podem ser mais precisos, mas são menos interpretáveis, mais propícios a falhar de forma imprevisível e mais difíceis de sustentar. Começar do básico também garante que o projeto se alinhe às necessidades do negócio.

Equívoco 4: técnicas são iguais entre indústrias. É necessário **conhecimento do domínio** para entender quais são os dados e problemas mais relevantes. As técnicas usadas para limpar e armazenar dados, assim como extrair insights e fazer modelagem, são similares, mas a aplicação exige contexto.

Equívoco 5: projetos começam bem definidos. Projetos de ciência de dados normalmente são exploratórios e experimentais. Geralmente não é claro o quão difícil será a solução do problema até que se explorem os dados. Gerentes de produto devem trabalhar com a equipe e demais envolvidos para gerenciar expectativas.

Equívoco 6: os melhores modelos de previsão são os mais avançados. Há mais desafios envolvidos na escolha de um modelo do que sua capacidade de previsão. Alguns modelos podem ser lentos ou complicados demais para incluir em produção, e alguns podem não ser interpretáveis, tornando investidores mais relutantes.

7.8 Recomendações de estudo

O material indica fontes para acompanhamento contínuo da área: Kaggle, Medium, KDnuggets, Papers with Code e arXiv.

8 Estimativa de Custos e Criação de Endpoint

Esta aula une dois temas complementares: um modelo de aprendizado supervisionado baseado em árvores e o modelo de cobrança da AWS, culminando na elaboração de um orçamento realista para um projeto de IA de três anos.

8.1 Árvores de regressão

A referência clássica citada é Breiman, Leo, et al., **Classification and regression trees**, CRC Press, 1984.

O princípio de funcionamento é a partição recursiva do espaço de atributos. A cada divisão, ou **split**, o algoritmo **minimiza a soma do erro quadrático para os lados direito e esquerdo** da partição. Os splits são escolhidos a partir de valores presentes na base de dados e de forma a minimizar a soma dos erros quadráticos.

O exemplo dos slides constrói a árvore progressivamente. O primeiro split ocorre no valor 20 de uma variável; o segundo split, em 170 de outra; o terceiro, em 200; e o quarto, em 40. Cada região resultante recebe como predição a **média da variável dependente y** dos pontos que caem nela. No exemplo, as folhas assumem valores como 65,7; 300,5; 1023; -64,1 e 0,7.

Essa construção deixa claro por que uma árvore de regressão é um aproximador constante por partes: dentro de cada retângulo do espaço de atributos, a predição é única e igual à média local.

8.2 Random Forest

O **Random Forest** estende a árvore individual por meio de um comitê de árvores. O algoritmo, conforme apresentado, opera em quatro passos:

- **Passo 1**: escolher o número de árvores que se deseja construir e o espaço de atributos S, repetindo os passos 2 e 3 para cada árvore.
- **Passo 2**: escolher aleatoriamente K dados e S atributos do conjunto de treinamento.
- **Passo 3**: construir a árvore de decisão associada àqueles K dados.
- **Passo 4**: para cada dado novo, fazer com que cada árvore da floresta infira um valor da variável resposta. A resposta do comitê será, por exemplo, a **média aritmética** da inferência de todas as árvores.

A aleatorização dupla, nos dados e nos atributos, é o que descorrelaciona as árvores e reduz a variância do comitê em relação a uma árvore isolada.

O treinamento é conduzido no SageMaker com scikit-learn, seguindo a documentação oficial em `https://sagemaker.readthedocs.io/en/stable/frameworks/sklearn/using_sklearn.html`.

```
from sklearn.ensemble import RandomForestRegressor

modelo = RandomForestRegressor(n_estimators=100, random_state=42)
modelo.fit(X_train, y_train)
predicoes = modelo.predict(X_test)
```

8.3 Fundamentos de cobrança da AWS

O modelo de cobrança da AWS repousa sobre três dimensões:

- **Compute**: cobrança por uso, em hora ou segundo, variando conforme o tipo de instância.
- **Storage**: cobrança por GB.
- **Data Transfer**: a saída de dados é agregada e cobrada por GB; a chegada de dados não tem cobrança.

Os três princípios de pagamento são: **pagar apenas pelo que foi usado**, **pagar menos ao reservar** e **pagar menos por mais uso e conforme a AWS cresce**.

8.4 On-premises versus nuvem

A comparação estruturada contrapõe a infraestrutura tradicional (equipamentos, recursos e administração, custo e contratos) à nuvem AWS (escalar para cima e para baixo, sem despesa inicial e pagando pelo uso, melhor tempo de chegada ao mercado e agilidade, infraestrutura self-service).

8.5 Benefícios concretos e indiretos

Como cientista de dados com conhecimento em nuvem, é preciso saber apontar as vantagens do ambiente em nuvem. O material as separa em duas categorias.

Benefícios concretos:

- Redução de gastos com computação, armazenamento, rede e segurança
- Reduções nas compras de hardware e software, isto é, redução de capex
- Reduções nos custos operacionais, de backup e de recuperação de desastres
- Redução na equipe de operações

Benefícios indiretos:

- Reutilização de serviços e aplicativos, que permite definir e redefinir soluções usando o mesmo serviço de nuvem
- Aumento da produtividade do desenvolvedor
- Maior satisfação do cliente
- Processos de negócios ágeis, capazes de responder rapidamente a oportunidades novas e emergentes
- Aumento do alcance global

8.6 AWS Pricing Calculator

A **AWS Pricing Calculator** é a ferramenta central para estimativa de custos. Ela permite:

- Estimar custos mensais
- Identificar oportunidades de reduzir custos mensais
- Modelar soluções antes de construí-las
- Explorar pontos de preço e os cálculos por trás da estimativa

- Encontrar os tipos de instância e termos de contrato disponíveis que atendem às necessidades
- Nomear a estimativa e criar e nomear grupos de serviços

Uma estimativa gerada é dividida em três partes: **total dos primeiros 12 meses**, **total inicial (upfront)** e **total mensal**.

A atividade proposta consiste em dividir a turma em grupos de quatro ou cinco pessoas para usar a AWS Pricing Calculator com especificações fornecidas e desenvolver uma estimativa de custo, reportando as descobertas à classe.

8.7 Do treino ao endpoint: o esquema clássico de Deep Learning

O esquema apresentado tem quatro estágios encadeados:

1. **Treino e Avaliação**: o modelo é treinado e avaliado.
2. **Deploy Model**: o modelo é implantado.
3. **API**: expõe-se o modelo por meio de uma interface programática.
4. Consumo pela aplicação final.

Esse encadeamento é exatamente o que torna a estimativa de custos não trivial: cada estágio consome recursos diferentes, com perfis de cobrança distintos.

8.8 Exercício integrador: orçamento de três anos

O exercício final propõe um cenário realista. Uma empresa deseja construir um sistema de visão computacional que identifica objetos em imagens enviadas por usuários. A empresa exige duas fases: **Desenvolvimento (2 anos)** e **Produção (1 ano)**.

Deve-se montar um orçamento detalhado que cubra:

- Treinamento do modelo ao longo de 2 anos
- Armazenamento de dataset e modelos
- Endpoint em produção por 1 ano
- Custos de inferência
- Transferência de dados
- Vida útil total do projeto de 3 anos

8.9 Gabarito da estimativa de custo

O gabarito orienta a decomposição do orçamento em duas frentes principais.

Na frente de **Storage**, usa-se o **Amazon S3** para armazenamento do dataset e do modelo, com atenção à possibilidade de novos dados e múltiplos modelos ao longo do tempo. Um ponto frequentemente esquecido e destacado no gabarito é **incluir o custo de operações PUT no S3**, referentes ao salvamento dos dados enviados pelos usuários. A calculadora específica está em `https://calculator.aws/#/createCalculator/S3`.

Na frente de **Compute**, usa-se o **EC2** com modelos de instância equipados com **GPU**, apropriados para treinamento e inferência de modelos de visão computacional. Toda a solução é elaborada via

SageMaker AI. A referência para escolha de instâncias é a página de tipos de instância de computação acelerada da AWS, em `https://aws.amazon.com/ec2/instance-types/accelerated-computing/`.

9 Introdução à Redes Neurais Artificiais

Esta aula, também da Profa. Manoela Kohler, constrói a teoria de redes neurais desde a inspiração biológica até o Multilayer Perceptron.

9.1 Inteligência computacional

O objetivo da inteligência computacional é o desenvolvimento de paradigmas ou algoritmos que requeiram máquinas para realizar tarefas cognitivas. Um sistema de inteligência computacional deve ser capaz de fazer três coisas:

- Armazenar conhecimento
- Aplicar o conhecimento armazenado para resolver problemas
- Adquirir novo conhecimento através da experiência

9.2 Inspiração na natureza

Diversas técnicas de inteligência computacional têm inspiração em fenômenos naturais:

Técnica	Inspiração
Sistemas especialistas	Inferência humana
Lógica Fuzzy	Processamento linguístico
Redes Neurais	Neurônios biológicos
Algoritmos Genéticos	Evolução biológica
Particle Swarm	Comportamento de pássaros e peixes
Sistemas Híbridos	Aspectos combinados

9.3 Redes neurais biológicas

Cada neurônio recebe estímulos de outros neurônios, e as **sinapses** são utilizadas para a comunicação. O efeito de cada estímulo no neurônio é controlado por um **peso sináptico**, que pode ser positivo ou negativo. Esse peso se adapta para que a rede como um todo aprenda: reconhecimento de objetos, entendimento de outras línguas, controle do próprio corpo. O processamento é **altamente paralelo**.

O experimento dos pombos como especialistas em arte reaparece aqui, agora com a leitura voltada às redes: a capacidade de extrair padrões e generalizar é característica das redes neurais, biológicas e artificiais, e não de um computador convencional.

9.4 Definição de rede neural artificial

Redes Neurais Artificiais são sistemas inspirados nos neurônios biológicos e na estrutura massivamente paralela do cérebro, com capacidade de adquirir, armazenar e utilizar conhecimento experimental.

A semelhança com o cérebro se dá em dois aspectos:

- O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de **aprendizado**.
- As forças de conexão entre neurônios, conhecidas como **pesos sinápticos**, são utilizadas para armazenar o conhecimento adquirido.

O objetivo do campo é estudar a teoria e a implementação de sistemas massivamente paralelos que possam processar informações com eficiência comparável à do cérebro.

A correspondência entre os dois mundos pode ser resumida assim:

Redes Neurais Biológicas	Redes Neurais Artificiais
Neurônio biológico	Neurônio artificial
Rede de neurônios	Estrutura em camadas
Bilhões de neurônios	Dezenas a milhões de neurônios
Conexões sinápticas	Pesos e bias (parâmetros)

9.5 Características básicas

Procura paralela e endereçamento pelo conteúdo. O cérebro não possui endereço de memória e não procura a informação sequencialmente.

Aprendizado. A rede aprende por experiência, não sendo necessário explicitar os algoritmos para executar uma determinada tarefa.

Associação. A rede é capaz de fazer associações entre padrões diferentes. Exemplos citados: foto para pessoa, sintomas para doença, leitura de sensores para falha, características do cliente para fraude.

Generalização. Redes neurais são capazes de generalizar seu conhecimento a partir de exemplos anteriores, inclusive para exemplos nunca vistos, com habilidade de lidar com ruídos e distorções e responder corretamente a padrões novos.

Abstração. Capacidade de abstrair a essência de um conjunto de entradas, isto é, a partir de padrões ruidosos extrair a informação do padrão sem ruído.

Robustez e degradação gradual. A perda de um conjunto de elementos processadores ou de conexões sinápticas não causa o mau funcionamento da rede neural.

9.6 História das redes neurais

- **1943**: o neurônio de **McCulloch e Pitts**, modelo computacional para o neurônio artificial, sem capacidade de aprendizado.
- **1949**: Hebb, em The Organization of Behavior, apresenta a primeira formulação explícita de uma regra de aprendizado, a **Hebbian Learning Rule**.
- **1958**: o **perceptron** de Rosenblatt, método de aprendizado supervisionado.
- **1969**: Minsky e Papert publicam o livro sobre perceptrons, provando as limitações do perceptron de uma camada.
- **1982**: Hopfield faz a pesquisa na área voltar a crescer; Kohonen apresenta os **mapas auto-organizáveis**.
- **1983**: Barto trabalha com aprendizado por reforço e controle.
- **1986**: Rumelhart apresenta o **backpropagation**, método mais popular para treinamento de perceptrons de múltiplas camadas e modelos de aprendizado profundo.
- **Atualmente**: interpretações probabilísticas, redes **spiking** e Deep Learning.

9.7 O problema do OU-EXCLUSIVO

A limitação provada por Minsky e Papert é ilustrada pela tabela verdade do XOR:

Ponto	X1 e X2	Saída
A0	0 e 0	0
A1	0 e 1	1
A2	1 e 0	1
A3	1 e 1	0

Um perceptron de uma camada resolve **somente funções linearmente separáveis**, e o XOR não é linearmente separável: não existe uma única reta capaz de separar os pontos A1 e A2 dos pontos A0 e A3. Essa constatação foi a causa direta de um dos invernos da IA.

9.8 O neurônio artificial

O neurônio artificial, ou **elemento processador**, é composto por: entradas, sinapses (pesos), soma ponderada, bias e função de ativação.

Sobre as **entradas**: são atributos independentes, sujeitos a **normalização**, e representam uma única observação. Sobre a **saída da rede neural**: pode ser contínua, binária ou categórica, representa a resposta de uma única observação, e pode haver mais de uma saída.

Sobre os **pesos**, o material é enfático: são o ponto crucial de uma rede neural, é o que de fato representa a rede, e **a criação de uma rede neural é basicamente o processo de ajuste destes pesos**.

Formalmente, o combinador linear calcula a soma ponderada das entradas, e a saída do neurônio é obtida aplicando a função de ativação à soma ponderada acrescida do bias. Em notação simplificada, o campo local induzido é a soma dos produtos entre pesos e entradas, e a saída é $y_k = \varphi(u_k + b_k) = \varphi(net)$.

9.9 Funções de ativação e bias

As funções de ativação apresentadas são: **degrau ou limiar**, **logística**, **linear**, **tangente hiperbólica (tanh)** e **ReLU**.

O **bias** tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, deslocando o ponto de operação do neurônio.

9.10 Como a rede neural funciona

O funcionamento é ilustrado com um problema de avaliação de imóveis. As entradas são metragem, distância ao centro e número de quartos; elas alimentam uma **camada escondida**, que por sua vez alimenta a **saída**, produzindo a previsão.

Um detalhe conceitual importante é destacado: podemos ter pesos com valores zero. Nem todos os atributos são importantes para todos os neurônios; **a conexão ainda existe, mas não tem valor na saída do neurônio**. Isso significa que cada neurônio da camada escondida pode se especializar em um subconjunto de atributos.

9.11 Topologias

Duas grandes famílias são definidas:

- **Redes Feed-Forward**: redes de uma ou mais camadas de processadores cujo fluxo de dados é sempre em uma única direção, isto é, não existe realimentação.
- **Redes Recorrentes**: redes com conexões entre processadores da mesma camada ou com processadores de camadas anteriores, ou seja, com realimentação.

Dentro das feed-forward, há a rede com uma única camada de saída e a rede de múltiplas camadas, com uma camada escondida e uma camada de saída, que é o **MLP**.

9.12 Perceptron de uma camada

O perceptron de uma camada é a forma mais simples de rede neural usada para a classificação de padrões linearmente separáveis. O **teorema da convergência do perceptron** garante que, se os padrões usados no treinamento forem retirados de classes linearmente separáveis, o perceptron converge e posiciona a superfície de decisão na forma de um **hiperplano** entre as duas classes.

9.13 Multilayer Perceptron

O problema do OU-EXCLUSIVO pode ser solucionado adicionando-se uma camada intermediária de processadores, dando origem ao **Multi-Layer Perceptron**. A solução clássica decompõe o XOR em funções mais simples: **OR**, **NAND** e **AND**.

Concretamente, os slides mostram que as retas separadoras associadas aos neurônios da camada escondida podem ser escritas a partir das combinações lineares $20x_1 + 20x_2 - 10$, que corresponde

à reta $x_2 = -x_1 + 0{,}5$, e $-20x_1 - 20x_2 + 30$, que corresponde à reta $x_2 = -x_1 + 1{,}5$. Duas retas paralelas delimitam a faixa em que a saída do XOR é 1, o que uma única reta jamais conseguiria.

A definição formal do MLP: é uma extensão do perceptron proposto por Rosenblatt, composta de várias camadas de neurônios. É a arquitetura de rede neural clássica mais utilizada e contém três tipos de camadas: **camada de entrada**, **camada ou camadas escondidas (intermediárias)** e **camada de saída**. Qualquer neurônio de uma camada pode interligar-se com outro neurônio da camada seguinte.

O treinamento de um MLP é realizado de maneira **supervisionada** com o algoritmo **backpropagation**, ou retropropagação do erro, baseado na regra de aprendizado por correção de erro. O desafio que ele resolve é achar um algoritmo de aprendizado para a atualização dos pesos das **camadas intermediárias**. Nesse algoritmo, a determinação do sinal de erro é um processo recursivo que se inicia nos neurônios da camada de saída e vai até os neurônios das camadas intermediárias. Esse foi um dos motivos para o ressurgimento da área de redes neurais.

9.14 Processamento neural: aprendizado e recall

O processamento de uma rede neural pode ser dividido em duas fases:

- **Aprendizado**: processo de atualização de pesos sinápticos para aquisição do conhecimento, ou seja, **aquisição da informação**.
- **Recall**: processo de cálculo da saída da rede dado um certo padrão de entrada, ou seja, **recuperação da informação**.

O material recomenda o site `http://playground.tensorflow.org`, um playground interativo de redes neurais, como ferramenta de intuição.

9.15 Estudos de caso

Predição da faixa de preço de celulares. A base tem 2000 observações, 20 atributos e 4 classes correspondentes a faixas de preço. Os atributos incluem `battery_power` (energia total que a bateria pode armazenar, em mAh), `blue` (tem bluetooth ou não), `clock_speed` (velocidade de execução de instruções pelo microprocessador), `dual_sim`, `fc` (megapixels da câmera frontal), `four_g`, `int_memory` (memória interna em gigabytes), `m_dep` (profundidade do aparelho em cm), `mobile_wt` (peso), `n_cores` (número de núcleos), `pc` (megapixels da câmera principal), `px_height` e `px_width` (resolução em pixels), `ram` (memória RAM em megabytes), `sc_h` e `sc_w` (altura e largura da tela em cm), `talk_time` (maior duração de uma única carga de bateria), `three_g`, `touch_screen` e `wifi`.

Câncer de mama. Base do University of Wisconsin, Clinical Sciences Center, com 30 atributos mais classe e identificador. Entre os atributos estão o **raio** (distância média do centro a pontos no perímetro do tumor), a **textura** (desvio padrão dos valores em escala de cinza), o **perímetro** e a **área**. A base tem **569 instâncias**, sendo **357 benignas** e **212 malignas**.

10 Algoritmos de Aprendizado e Otimizadores

Esta aula aprofunda o mecanismo pelo qual os pesos de uma rede neural são efetivamente ajustados.

10.1 Recapitulação e definições importantes

A aula começa retomando o problema do OU-EXCLUSIVO, o neurônio artificial com suas sinapses, entradas e bias, as funções de ativação (degrau, logística, linear, tanh e ReLU) e a topologia MLP feed-forward.

Em seguida, define dois conceitos operacionais indispensáveis.

Época. É a apresentação de todos os registros da base de dados e a retropropagação do erro para ajuste dos pesos. Em uma época podem ocorrer inúmeras **iterações**, e a cada iteração é feito o ajuste dos pesos.

A relação entre época, iteração e **batch size** é ilustrada com uma base de dados de quatro registros:

- Com **batch size igual a 4**, é necessária 1 iteração para apresentar todos os registros, resultando em 1 ajuste de peso por época.
- Com **batch size igual a 2**, são necessárias 2 iterações para apresentar todos os registros, resultando em 2 ajustes de pesos por época.

Em ambos os casos, 50 épocas significam que todos os registros serão apresentados à rede 50 vezes.

Loss (função custo). É a função para cálculo do erro da rede, ou seja, a função que calcula o erro entre o valor real e o valor previsto. O exemplo dos slides compara um vetor previsto com valores como 0,7; 0,2 e 0,1 contra um vetor real com 1,0 e 0,0.

10.2 Propriedades do processo de aprendizado

Duas propriedades são de importância primordial para uma rede neural: aprender a partir de seu ambiente e melhorar o seu desempenho através da aprendizagem.

O processo de aprendizado se dá assim: a rede neural aprende acerca de seu ambiente através de um **processo iterativo de ajustes aplicados a seus pesos sinápticos e níveis de bias**. Idealmente, a rede se torna mais instruída sobre o seu ambiente após cada iteração (época) do processo.

10.3 Variedade de algoritmos de aprendizado

Um algoritmo de aprendizado é um conjunto preestabelecido de regras bem definidas para a solução de um problema de aprendizado. Existe uma variedade deles:

- Por correção de erro
- Baseada em memória
- Hebbiana

- Competitiva
- Boltzmann
- Backpropagation, incluindo Gradiente Descendente e Levenberg-Marquardt

10.4 Backpropagation

O algoritmo backpropagation opera em **dois passos**.

Feed Forward (propagação). Um padrão é apresentado à camada de entrada da rede e propagado em direção à camada de saída. A saída obtida é comparada com a saída desejada para aquele padrão particular, e então o erro é calculado.

Feed Backward (retropropagação). O erro é propagado a partir da camada de saída até a camada de entrada. Os pesos das conexões dos neurônios das camadas internas vão sendo atualizados conforme o erro é retropropagado. Para os neurônios das camadas intermediárias, onde não existem saídas desejadas, o sinal do erro é determinado **recursivamente** em termos dos sinais dos erros dos neurônios diretamente conectados a eles e dos pesos dessas conexões.

Vantagens do backpropagation: é simples de implementar e apresenta boa capacidade de generalização.

Desvantagens: custo computacional significativo e baixa velocidade de aprendizado.

10.5 A função de erro e o gradiente descendente

Deve-se minimizar o erro de todos os processadores da camada de saída, para todos os padrões, por meio da **soma dos erros quadráticos**. Na notação dos slides, o erro ESSE é metade do somatório, sobre todos os padrões p e todos os processadores k da camada de saída, do quadrado da diferença entre a saída desejada e o estado de ativação. Aqui, `s_pk` é o estado de ativação do processador k ao apresentar o padrão p, e `t_pk` é a saída desejada para o processador k ao apresentar o padrão p.

A minimização desse erro se dá pelo método do **Gradiente Descendente**: cada peso sináptico i do elemento processador j é atualizado proporcionalmente ao **negativo da derivada parcial do erro** desse processador com relação ao peso, escalado pela taxa de aprendizado.

10.6 Otimizadores

Três variantes do gradiente descendente são apresentadas:

- **Gradiente Descendente** (batch completo): usa todos os registros da base para calcular o gradiente antes de cada atualização.
- **Gradiente Descendente Estocástico**: atualiza os pesos a cada registro apresentado.
- **Gradiente Descendente em Mini Batch**: atualiza os pesos a cada lote de registros, sendo o compromisso entre as duas abordagens anteriores.

Essas três variantes correspondem exatamente às escolhas de `batch size` discutidas no início da aula, o que fecha o raciocínio: escolher o tamanho do lote é escolher o otimizador.

10.7 Levenberg-Marquardt

A principal crítica ao backpropagation é o **longo tempo de treinamento**. Uma proposta de variação para acelerar o processo de aprendizado é o algoritmo de **Levenberg-Marquardt (LM)**.

Trata-se de um método mais rápido para o treinamento de redes neurais feed-forward de **tamanho moderado**, até algumas centenas de pesos sinápticos. Como se baseia em matrizes, as implementações em Matlab, R e Python ficam ainda mais eficientes.

O raciocínio por trás dele é o seguinte. O backpropagation tradicional é baseado na **derivada de primeira ordem** da saída em relação aos pesos. Para acelerar o treinamento, seria desejável considerar **derivadas de segunda ordem**. O problema é que o cálculo da **matriz Hessiana**, que contém as derivadas de segunda ordem, é muito custoso computacionalmente.

O algoritmo de Levenberg-Marquardt foi projetado para atingir velocidades de treinamento de segunda ordem **sem a necessidade de calcular explicitamente a matriz Hessiana**. Ele se baseia na **matriz Jacobiana**, que contém as primeiras derivadas parciais dos erros em relação aos diversos pesos sinápticos.

10.8 Estudos de caso

Expectativa de Vida. Base da World Health Organization (WHO), com 20 atributos mais a classe, incluindo ano, status, mortalidade em adultos e consumo de álcool, totalizando **2938 registros**.

KDD Cup. Base da terceira competição internacional de ferramentas de Knowledge Discovery and Data Mining, associada à KDD-99. O problema é a **detecção de intrusão** em ambiente militar, especificamente uma LAN da U.S. Air Force. A base foi preparada e gerenciada pelo MIT Lincoln Labs, e seus atributos são dados de tráfego da rede a ser protegida. A base original tem **5 milhões de registros, 24 classes e 41 atributos**.

Câncer de mama. A mesma base do University of Wisconsin descrita na aula anterior, com 569 instâncias, 357 benignas e 212 malignas.

11 DeepLearning Overview

A aula final revisita os conceitos centrais do curso e apresenta um panorama amplo das aplicações de aprendizado profundo.

11.1 Revisão para o quiz

A primeira metade da aula é uma revisão sistemática. Retoma-se a definição de computação em nuvem, as seis vantagens (economias de escala, fim da adivinhação de capacidade, velocidade e agilidade, fim dos gastos com data centers, alcance global em minutos), os tipos de computação em nuvem (IaaS, PaaS e SaaS) e o Amazon S3 como armazenamento em nível de objeto. Também se revisitam a redução de dimensionalidade com PCA e t-SNE, as árvores de regressão com seus valores de folha, e as redes neurais.

11.2 Definição de Deep Learning

Deep Learning é uma abordagem de aprendizado de máquina baseada em redes neurais profundas, capazes de aproximar funções complexas por meio de representações hierárquicas. Esse campo engloba uma variedade de técnicas voltadas para regularização e melhoria na generalização do modelo, sendo amplamente utilizado em tarefas como visão computacional, processamento de linguagem natural e séries temporais.

11.3 Visão computacional

Visão Computacional é a área da inteligência artificial que estuda métodos e algoritmos capazes de permitir que máquinas interpretem, analisem e extraiam informações de imagens e vídeos. A área utiliza técnicas de processamento de imagens, aprendizado de máquina e, mais recentemente, Deep Learning, com destaque para as **redes neurais convolucionais (CNNs)**, que aprendem representações hierárquicas diretamente dos dados visuais. Essas representações permitem capturar padrões como **bordas, texturas, formas e objetos complexos**.

A operação fundamental dessas redes é a **convolução**, que aplica filtros deslizantes sobre a imagem para extrair mapas de características.

11.4 Detecção de objetos

Os fundamentos da área de detecção de objetos são cobertos junto a um catálogo de arquiteturas:

- Família **YOLO**, inclusive a **YOLOv12**
- **Mask R-CNN**
- **RetinaNet**
- **DETR**
- Família **Fast R-CNN**
- **Tracking** (rastreamento de objetos ao longo do vídeo)

As aplicações incluem detecção de objetos em imagens e em vídeo, **OCR** (reconhecimento óptico de caracteres), monitoramento de velocidade de trânsito e cenários de segurança, entre outros. Os arquivos de apoio da aula incluem diversos vídeos que demonstram esses casos.

11.5 Modelos generativos e outras frentes

Além da detecção, a aula apresenta um conjunto de tópicos que compõem a fronteira da área:

- **Diffusion Models** e **GANs**, para geração de imagens.
- **Reconhecimento facial**.
- **Similarity Learning**, no qual o modelo aprende uma representação que **aproxima entradas similares e afasta as diferentes**. Esse princípio é a base de sistemas de busca por similaridade e de verificação de identidade.
- **Explainable AI**, voltada à interpretabilidade dos modelos.
- **Noisy Labels**, tratamento de rótulos ruidosos no conjunto de treinamento.
- **Identificação de Deep Fake**.

11.6 Processamento de Linguagem Natural

Processamento de Linguagem Natural (PLN) é a área da inteligência artificial que utiliza Deep Learning para modelar, compreender e gerar linguagem humana, por meio de representações distribuídas e modelos neurais, como redes recorrentes e arquiteturas Transformer, amplamente aplicadas em tarefas de tradução, classificação de texto e geração de linguagem.

O fio condutor da exposição é a evolução das **representações de palavras**. Para que modelos de aprendizado de máquina possam processar texto, é necessário transformar palavras em representações numéricas.

Métodos clássicos: Bag of Words. O BoW representa um texto como um vetor de contagem ou frequência de palavras, desconsiderando a ordem e o contexto em que elas aparecem. Embora simples e eficientes, essas representações são limitadas, pois não capturam relações semânticas entre palavras nem ambiguidades de significado.

Word embeddings. Com o avanço do Deep Learning surgiram os word embeddings, como o **Skip-Gram** e o **GloVe**, que representam palavras em vetores densos de dimensão fixa. Esses vetores são aprendidos a partir de grandes volumes de texto e capturam similaridades semânticas e sintáticas, permitindo que palavras com significados semelhantes tenham representações próximas no espaço vetorial.

Embeddings contextuais. Mais recentemente, modelos baseados em **Transformers** introduziram os embeddings contextuais, nos quais a representação de uma palavra depende do contexto em que ela aparece na frase. Arquiteturas como **BERT, GPT e T5** produzem vetores dinâmicos que resolvem ambiguidades semânticas e capturam relações complexas de longo alcance, representando o estado da arte em tarefas de PLN.

O exemplo escolhido para tornar isso concreto é a palavra "banco" em duas frases: "O banco aprovou o empréstimo para o cliente" e "Ele sentou no banco da praça para descansar". Em um modelo de word embedding estático, a palavra "banco" teria um único vetor em ambos os casos, misturando os dois sentidos. Em um modelo contextual, cada ocorrência recebe um vetor distinto, refletindo a instituição financeira no primeiro caso e o assento no segundo. Essa é a diferença qualitativa que os Transformers introduziram.

12 AWS na Prática - Quiz Final

Material de slides não disponível para esta aula.

O quiz final é o instrumento de avaliação da disciplina. Como se depreende da estrutura da aula de Deep Learning Overview, que dedica sua primeira metade explicitamente à "Revisão Quiz", os temas cobrados concentram-se nos fundamentos conceituais do curso: a definição de computação em nuvem, as seis vantagens do modelo, a distinção entre IaaS, PaaS e SaaS, os modelos de implantação (cloud, híbrido e on-premise), o Amazon S3 como armazenamento em nível de objeto e suas classes, a distinção entre armazenamento em bloco e em objeto, o papel do EC2 e dos serviços de computação relacionados, o modelo de cobrança da AWS por compute, storage e data transfer, além dos conceitos de redução de dimensionalidade, árvores de regressão e redes neurais.

13 Síntese da Disciplina

A disciplina constrói, camada por camada, o que significa fazer ciência de dados na nuvem. O percurso não é acidental: cada bloco resolve uma limitação deixada pelo anterior.

O bloco inicial responde à pergunta "onde o trabalho acontece". A resposta é a nuvem, definida como entrega sob demanda de recursos de TI com pagamento conforme o uso. As seis vantagens (troca de capex por opex, economias de escala, fim da adivinhação de capacidade, velocidade, foco no negócio em vez de data centers, e alcance global) justificam economicamente essa escolha, e a taxonomia IaaS/PaaS/SaaS organiza o que fica sob responsabilidade de quem. A prática correspondente é a criação do domínio SageMaker e a abertura de um JupyterLab hospedado.

O segundo bloco responde a "onde ficam os dados e onde roda o código". S3 e EC2 são os dois eixos. A distinção conceitual mais importante ali (bloco versus objeto, com suas consequências em desempenho, latência e custo) reaparece nas decisões de arquitetura de qualquer projeto. Os estudos de caso, com Lambda e com CloudFormation via AWS CLI, mostram que infraestrutura moderna é código, e que automação e observabilidade vêm de graça quando se adotam os serviços gerenciados.

O terceiro bloco, dedicado a Python, fornece o instrumento. Tipos, estruturas de controle e funções são o mínimo necessário, e a Lei dos Grandes Números serve como exercício integrador elegante: ela ensina programação e, ao mesmo tempo, prepara a intuição estatística de que amostras pequenas enganam e amostras grandes convergem.

Essa intuição é exatamente o ponto de partida do quarto bloco, sobre análise exploratória. Se um dataset é uma amostra sujeita a variação aleatória, então é preciso quantificar essa incerteza (intervalos de confiança, testes de hipótese, valor-p) antes de afirmar qualquer coisa. A advertência de que correlação não implica causalidade e o cuidado com o nível de significância de 5% são salvaguardas metodológicas que atravessam todo o restante do curso. PCA e t-SNE entram como ferramentas para ver o que está escondido em alta dimensão.

O quinto bloco enquadra historicamente o que se está fazendo. A história da IA, com seus dois invernos e seus ciclos de hype, ensina uma lição de sobriedade, reforçada pelos seis equívocos comuns: não existe unicórnio, a solução mais complexa não é necessariamente a melhor, projetos de dados são exploratórios por natureza, e a escolha de um modelo envolve muito mais do que sua acurácia. O experimento dos pombos, que atravessa duas aulas, fixa a definição operacional de aprendizado: extrair padrão e generalizar, não memorizar.

O sexto bloco une modelagem e economia. Árvores de regressão e Random Forest oferecem um modelo interpretável e robusto; simultaneamente, o modelo de cobrança da AWS (compute por hora ou segundo, storage por GB, transferência de saída por GB) e a Pricing Calculator ensinam a traduzir uma arquitetura em orçamento. O exercício de orçar três anos de um sistema de visão computacional (dois de desenvolvimento e um de produção, com treino, storage, endpoint, inferência e transferência) é possivelmente o exercício mais próximo da prática profissional em todo o curso.

O sétimo e oitavo blocos abrem a caixa-preta das redes neurais. Do neurônio de McCulloch e Pitts ao MLP, passando pela demonstração de que o XOR não é linearmente separável e pela solução via camada escondida, constrói-se a arquitetura. Depois, o backpropagation em seus dois passos, a minimização da soma dos erros quadráticos por gradiente descendente, e as variantes de otimização (batch completo, estocástico, mini-batch, Levenberg-Marquardt) explicam como os

pesos, que são a rede, chegam aos seus valores. A tríade época, iteração e batch size amarra teoria e implementação.

O bloco final amplia o horizonte. Deep Learning é definido como aproximação de funções complexas por representações hierárquicas, e essa definição se materializa em dois domínios. Em visão computacional, as CNNs aprendem bordas, texturas, formas e objetos, e sobre elas se constroem detectores (YOLO, Mask R-CNN, RetinaNet, DETR), geradores (GANs, modelos de difusão) e frentes de pesquisa como explicabilidade, rótulos ruidosos e detecção de deep fakes. Em PLN, a trajetória vai do Bag of Words, que ignora ordem e contexto, aos word embeddings densos (Skip-Gram, GloVe), e destes aos embeddings contextuais dos Transformers (BERT, GPT, T5), capazes de distinguir os dois sentidos da palavra "banco" conforme a frase.

A conexão entre todos os blocos é o ciclo completo mostrado no esquema de Deep Learning apresentado na aula de custos: treino e avaliação, deploy do modelo, exposição via API e consumo pela aplicação. Cada etapa desse ciclo tem um serviço da AWS correspondente, um custo associado e um conjunto de decisões técnicas. O que a disciplina entrega, ao final, não é apenas conhecimento de algoritmos nem apenas familiaridade com um console de nuvem, mas a capacidade de percorrer esse ciclo inteiro com consciência de custo, de método estatístico e de limitação de modelo.