



Técnicas para o Processamento do Big Data

Bootcamp Cientista de Dados

Túlio Philipe Ferreira e Vieira

2021

Técnicas para o Processamento do Big Data

Túlio Philipe Vieira

© Copyright do Instituto de Gestão e Tecnologia da Informação.

Todos os direitos reservados.

Sumário

Capítulo 1. Introdução	5
Qual é a importância do Big Data?	5
Fluxo analítico para o processamento do Big Data.....	7
Pilha para o processamento do Big Data.....	8
Arquiteturas para o processamento do Big Data	12
Aplicações do processamento do Big Data	16
Estudo de Caso: Airbnb	19
Capítulo 2. Streaming de Dados.....	24
O que é o processamento em Streaming?	24
Kafka.....	26
Apache Spark Streaming	27
Capítulo 3. Algoritmos de Machine Learning Aplicados ao Pré-Processamento do Big Data.....	30
Identificação de anomalias.....	30
K-means.....	33
Spark MLlib	36
Técnicas para Preparação dos Dados.....	38
Capítulo 4. Algoritmos de Machine Learning Aplicados ao Processamento do Big Data.....	45
Análise de regressão	45
Regressão linear.....	46
KNN	52
Árvore de Decisão	56
SVM	58

Redes Neurais Artificiais e Deep Learning	60
Capítulo 5. Sistemas de Recomendação.....	65
Sistemas de recomendação.....	65
Capítulo 6. Teoria dos Grafos.....	75
Introdução.....	75
Conceitos e terminologias.....	77
Algoritmos para grafos.....	85
Neo4j e Cypher	88
Criando um banco de dados grafo.....	91
Realizando consultas sobre o banco de dados.....	96
Referências.....	102

Capítulo 1. Introdução

Neste capítulo serão estudados alguns conceitos fundamentais para o entendimento de todo o curso. Sua finalidade é mostrar como o processamento do Big Data é essencial para qualquer organização ou instituição que visa analisar dados e obter insights para se tornar uma organização orientada a dados.

Quando as organizações passam a tomar decisões a partir de uma análise acurada de um conjunto de dados provindos de diferentes fontes, é possível que os processos, produtos e serviços sejam otimizados. Desse modo, essas organizações podem obter vantagens competitivas e antecipar as mudanças e oscilações do mercado.

Como vocês já sabem, o Big Data pode ser definido como uma coletânea de base de dados de fontes **variadas**, que possuem grande **volume** de informação e são geradas em alta **velocidade**. O conjunto dessas características constituem os 3V's do Big Data. Existem outras definições que adicionam mais 2 ou 4V's, entretanto, a definição de alto volume, variedade e velocidade resume as principais características que tornam a análise do Big Data essencial e, ao mesmo tempo, desafiadora para as organizações.

Compreender essas características e como elas devem ser tratadas é primordial para transformar essa enorme quantidade de dados em informações. São essas informações que possibilitam a construção dos insights e são os insights que geram o poder transformador do Big Data.

Qual é a importância do Big Data?

O sucesso das organizações não está contido apenas em realizar as tarefas do “core business” com qualidade. Na era da informação as empresas precisam saber analisar os dados de maneira eficaz, pois conhecer o cliente, a própria empresa, os competidores e traçar estratégias é fundamental para qualquer empresa que deseja alcançar sucesso no mercado atual.

Nas palavras de Geoffrey Moore:

“Sem a análise do Big Data, as empresas estão cegas e surdas, vagando pela web como cervos em uma estrada.”

Através do Big Data é possível fazer uma análise social, realizar análises comparativas, proceder análise sobre as campanhas de marketing, escolher o público alvo e identificar os níveis de satisfação dos clientes (GARY, 2017).

Através da análise social é possível identificar o que as pessoas ou clientes estão falando sobre a empresa. Por meio da análise de dados das redes sociais pode-se identificar qual é a imagem que a empresa está passando para os consumidores. Como várias empresas possuem sites de vendas de produtos e serviços, é possível acessar esses dados e realizar uma análise cruzada com os dados de sua empresa e identificar as estratégias que mais agradam os clientes. Através da análise das estratégias de marketing é possível observar quais campanhas podem afetar positivamente o crescimento da empresa. Utilizando a ferramenta Google Analytics, por exemplo, podem ser identificados os “movimentos” realizados pelos consumidores em potencial. A análise do Big Data, através das redes sociais, por exemplo, permite identificar clientes ou grupos de clientes em tempo real, o que, por meio de campanhas de marketing, podem alavancar as vendas de produtos e serviços. Outra importante característica que demonstra a importância do Big Data para as empresas é a análise de satisfação dos clientes. Devido ao grande sucesso das redes sociais, vários consumidores ou grupo de clientes utilizam desse meio para compartilhar experiências sobre determinado produto ou serviço adquirido. Como apenas as duas maiores redes sociais do mundo (Facebook e YouTube) possuem mais de 4 bilhões de usuários (STATISTA, 2019), obter dado, em tempo real, sobre a opinião dos usuários torna-se uma tarefa menos custosa.

Figura 1 - A importância dos dados segundo Tim Berners-Lee, inventor da World Wide Web.



Fonte: Team (2019).

Fluxo analítico para o processamento do Big Data

Para alcançar todo o poder transformador gerado pelo Big Data, é necessário que todo esse volume e variedade de dados sejam processados. Só através do processamento adequado desses dados que é possível transformar o Big Data em informações úteis para as empresas.

Nesse sentido, é necessário definir um fluxo de atividades que possa ser seguido, a fim de conduzir o processamento do Big Data à construção de informação e geração de insights. Segundo Bahga e Madisetti (2016), são necessárias cinco etapas para a realização do processamento do Big Data. A Figura 2 apresenta esse conjunto de fases.

Na etapa de coleta de dados são definidas as fontes de dados a serem utilizadas para a geração de conhecimento. Essas fontes de dados podem ser, por exemplo, bancos de dados SQL, NoSQL, Frameworks de publish-subscribe etc.

A etapa seguinte corresponde à preparação dos dados. Como os dados do Big Data possuem uma grande variedade de fontes e formatos, é necessário que sejam aplicadas técnicas que permitam “limpar” dados ruidosos, encontrar anomalias, preencher lacunas presentes e torná-los aptos a serem utilizados na etapa de análise.

A etapa de análise consiste em determinar o tipo de análise a ser inserida nesse conjunto de dados e quais os algoritmos mais indicados para esse fim. É nesta etapa que são definidos quais os algoritmos de Machine Learning que devem ser empregados, a fim de se alcançar o resultado desejado.

A penúltima etapa é a definição do modo como os dados devem ser analisados. É nessa etapa que é indicado como será realizada a análise dos dados. Nessa etapa que é decidido se deve ser empregada uma análise em tempo real, através de dados históricos ou por meio de interações com o usuário. A escolha pelo modo em tempo real, ocorre quando a aplicação exige que a cada momento em que os dados sejam coletados, eles devam ser analisados. Quando a análise dos dados ocorre após um grande intervalo de tempo, por exemplo, a cada mês, pode ser utilizada o modo de batelada (dados históricos), ao passo que a análise interativa ocorre devido a maior flexibilidade de interações com os usuários.

A última etapa corresponde à visualização. Nessa fase é definido o modo como os dados devem ser exibidos ao usuário da aplicação. Por exemplo, essa visualização pode ser estática, dinâmica ou interativa.

As definições de cada uma dessas etapas são fundamentais para a escolha correta dos algoritmos e ferramentas necessárias para a construção de conhecimento através do Big Data. Desse modo, a escolha e interação correta entre cada uma dessas ferramentas e algoritmos gera o poder transformador da análise do Big Data.

Figura 2 – Fases para o processamento do Big Data.



Pilha para o processamento do Big Data

A ferramenta mais utilizada para o processamento inicial do Big Data é o Hadoop. O Hadoop consiste em um framework em código aberto, que é utilizado para

o armazenamento e processamento distribuído de um grande volume de dados. Esse armazenamento é realizado de maneira eficiente através do *Hadoop Distributed File System* (HDFS). O HDFS é encarregado de manter a integridade e a recuperação da informação. O processamento do Big Data é realizado através do MapReduce. O MapReduce realiza o processamento paralelo dos dados contidos no HDFS. Operações simples como cálculo da média, valores mínimos e máximos, ordenação e *join*, são realizados diretamente pelo MapReduce.

Quando é necessária uma análise mais apurada do Big Data, por exemplo, através de operações que envolvam a utilização de algoritmos de Machine Learning, é essencial a adoção de outras ferramentas. A Figura 3 mostra a pilha de ferramentas e frameworks utilizados para a criação de aplicações que processem o Big Data.

A fonte primária de dados indica de onde os dados são coletados. Esses dados podem ser, por exemplo:

- Logs: dados coletados pelos webserver a fim de armazenar informações sobre as interações dos usuários com um determinado site.
- Mídias sociais: são dados coletados através das interações de usuários com as redes sociais.
- Dados de sensores: são dados coletados por sensores, principalmente através da IoT.
- Dados de sistemas de vigilância: são dados coletados por câmeras e sensores de um sistema de segurança.

Os conectores de acesso são as ferramentas necessárias para que os dados das fontes primárias possam ser introduzidos em um ambiente de processamento, e a escolha do tipo de conector deve ser realizada com base na fonte de dados. Alguns exemplos desses conectores são:

- Gerenciadores de mensagem do tipo publish-subscribe: alguns exemplos são o Kafka e o Amazon Kinesis.

- Conectores de bancos de dados como o Apache Sqoop.
- Filas de mensagens como o RabbitMQ e Amazon SQS.

O armazenamento é o sistema responsável por garantir a resiliência dos dados coletados. Devido à característica dos dados do Big Data, normalmente, esses dados são armazenados através de ferramentas que suportam os Bancos NoSQL. Diferentemente dos bancos relacionais, os NoSQL não necessitam que os dados coletados possuam características definidas e que permitam ser armazenados como uma tabela.

A análise por meio de batelada (batch) ocorre quando uma porção dos dados armazenados são utilizados. Esses dados já estão presentes no banco de dados e não existe a necessidade de proceder o processamento em tempo real. Algumas ferramentas empregadas são:

- Hadoop – MapReduce.
- Pig: um framework de mais alto nível, que possibilita a escrita de códigos que são traduzidos para a análise através do MapReduce.
- Spark: é um framework em código aberto, que permite a utilização de várias ferramentas para a análise do Big Data.

A análise em tempo real pressupõe que os dados devam ser coletados e analisados e, em um curto intervalo de tempo, preferencialmente, esses dados devem ser continuamente coletados e processados. Alguns frameworks utilizados são o Apache Storm e o Spark Streaming.

As consultas interativas permitem ao usuário da aplicação criar consultas no formato SQL. Desse modo a interação ocorre de maneira mais amigável. Algumas ferramentas que permitem a construção desse tipo de consulta são:

- Spark SQL: é uma das ferramentas existentes no Apache Spark e permite ao usuário construir consultas utilizando a linguagem SQL, para obter dados do Big Data.

- Hive: é um framework construído no top do ecossistema Hadoop, que permite realizar consultas através da Hive Query Language.
- Amazon Redshift: ferramenta desenvolvida pela Amazon, que permite ter acesso a dados de diferentes fontes utilizando consultas do tipo SQL.

A visualização dos resultados também é de fundamental importância para a construção do conhecimento, pois é através da exibição desses resultados que os tomadores de decisão podem obter os insights. Alguns exemplos de ferramentas utilizadas com essa finalidade, são:

- Seaborn: biblioteca escrita em Python, que permite a construção de gráficos.
- Lightning: framework utilizado para criar uma visualização interativa dos dados.
- Tableau: software que também permite a criação de visualizações de dados de maneira interativa.

Figura 3 – Pilha de ferramentas e frameworks para o processamento do Big Data.



O conhecimento sobre os frameworks existentes e suas respectivas características, são fundamentais para a construção e aplicação que almejam promover, de melhoria nas organizações através da análise de dados.

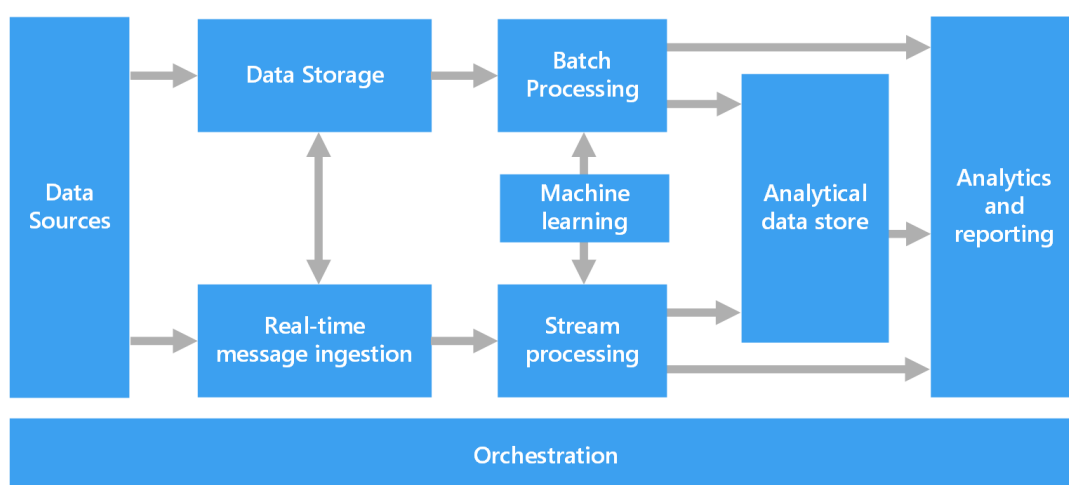
Arquiteturas para o processamento do Big Data

Definir a arquitetura também é importante para definir como os dados devem ser processados e disponibilizados para os usuários. A escolha sobre qual arquitetura utilizar, permite:

- Tratar grandes volumes de dados.
- Realizar o processamento dos dados em alta velocidade.
- Garantir tolerância a falhas.
- Criar aplicações que possam ser escaláveis.

Uma arquitetura geral que contemple todas as etapas de construção de uma aplicação de geração de conhecimento, a partir da análise do Big Data, deve conter os elementos presentes na Figura 4.

Figura 4 – Elementos de uma arquitetura do Big Data.



Fonte: Microsoft (2019).

Nessa figura é possível ver oito diferentes componentes, sendo eles:

- Fonte de dados: local de onde os dados são obtidos.
- Armazenamento: local onde os dados são armazenados.
- Processamento em batelada: processamento dos dados já armazenados.
- Ingestão de mensagens em tempo real: responsável por capturar os dados recebidos e manter a base de dados atualizada.
- Processamento em streaming: realiza o processamento contínuo dos dados coletados.
- Análise dos dados armazenados: permite a realização de consultas sobre o conjunto de dados já armazenados.
- Análise e reporting: permite que os resultados das análises possam ser acessados e exibidos para o usuário.
- Gerenciamento (orchestration): é o componente responsável por gerenciar todas as tarefas a serem executadas durante o processamento do Big Data.

A partir desse modelo geral de processamento, são criadas arquiteturas que permitem a análise de dados em tempo real e que garantem a tolerância a falhas. Principalmente quando existe a necessidade de realizar o processamento em tempo real, adotar todos os componentes presentes na Figura 4 pode ocasionar o aumento da latência no sistema. Nesse sentido, foram propostas arquiteturas alternativas para o processamento do Big Data, sendo as principais a Lambda e a Kappa.

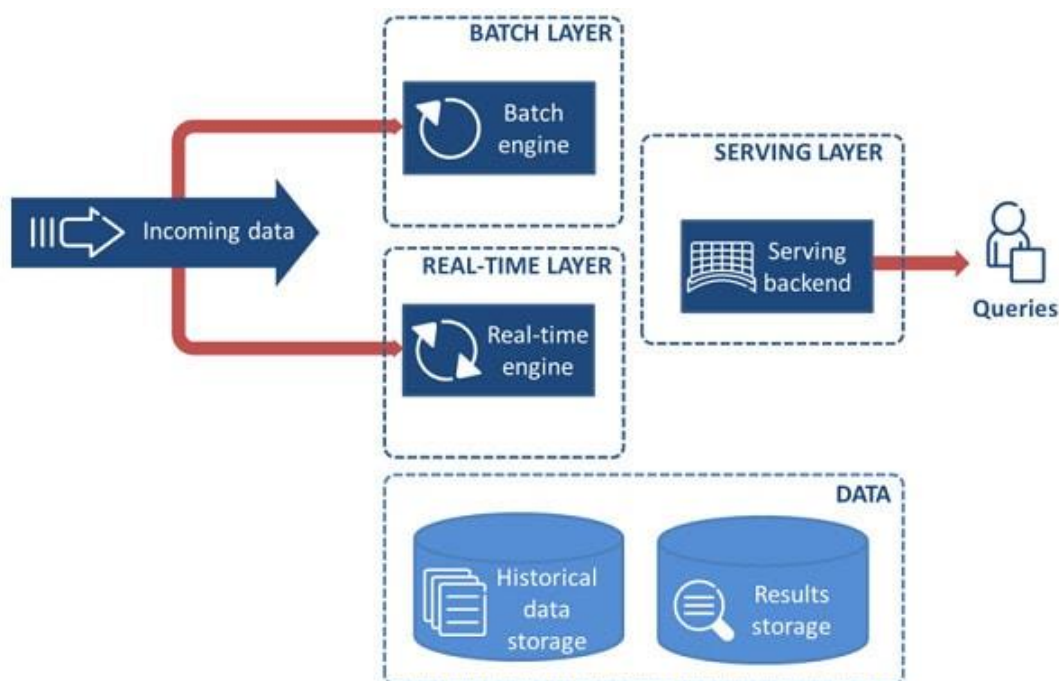
A arquitetura Lambda propõe a adoção de três diferentes camadas. Essas camadas são:

- Batch Layer: responsável por armazenar os dados recebidos. Os dados armazenados por essa camada são imutáveis e possuem um timestamp.

- Speed Layer: responsável por reduzir a latência entre o armazenamento e o processamento dos dados. Essa camada não garante a tolerância a falhas.
- Serving Layer: responsável por processar e realizar as consultas sobre os dados, que estão armazenados em batch.

A Figura 5 apresenta o modelo da arquitetura Lambda. Como pode ser visto, os dados coletados são enviados tanto para o Batch Layer quanto para o Speed/Real time Layer. Desse modo, a camada Speed impõe restrições quanto aos limites de latência tolerados, assim consegue entregar o processamento em tempo real. Entretanto, devido à essas restrições, não existe a garantia de tolerância a falhas. Por outro lado, a camada Batch não impõe tais restrições, assim é possível garantir a tolerância a falhas.

Figura 5 – Arquitetura Lambda do Big Data.



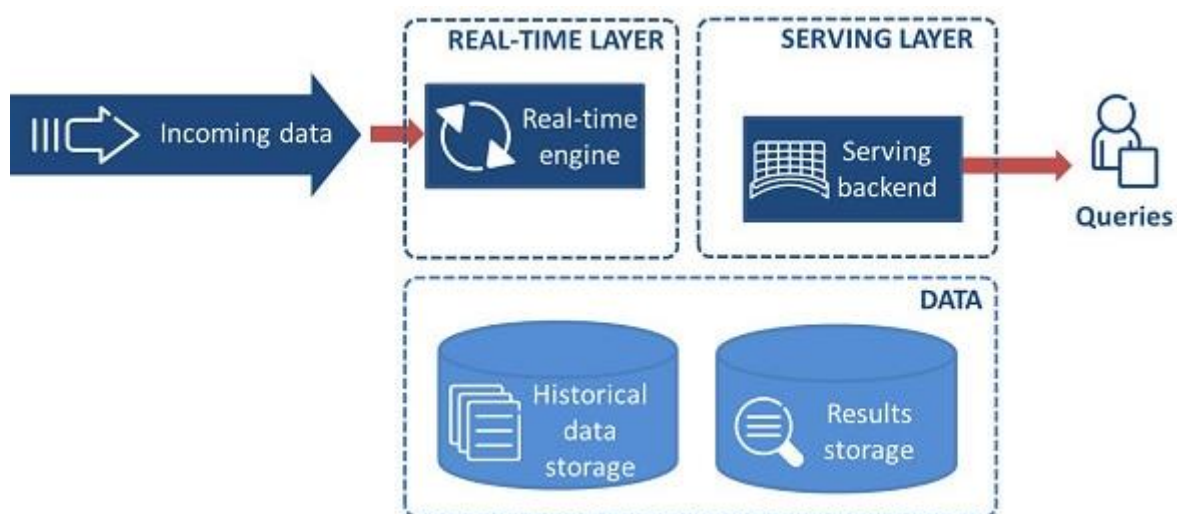
Fonte: Ericsson (2017).

Devido à necessidade de dois diferentes frameworks a serem utilizados para o processamento dos dados (Hot path e Cold path), a arquitetura Lambda exige maior complexidade e consome um maior número de recursos (hardware e software). Uma

arquitetura alternativa que visa reduzir a complexidade do sistema e do processamento do Big Data é a Kappa. A Figura 6 apresenta o modelo de arquitetura Kappa.

Por meio dessa figura é possível perceber que não existe a camada Batch. Desse modo os dados são processados apenas em tempo real. O reprocessamento dos dados ocorre apenas quando novos dados são introduzidos.

Figura 6 – Arquitetura Kappa do Big Data.



Fonte: Ericsson (2017).

A escolha sobre qual das arquiteturas utilizar para realizar o processamento do Big Data deve ser realizada com base na aplicação. Por exemplo, aplicações que demandem alta velocidade de processamento, possuem fonte reduzida de recursos (hardware e software) e podem lidar com uma menor resiliência do sistema, devem adotar a arquitetura Kappa. Entretanto, caso os recursos sejam ilimitados e a aplicação necessita de uma maior robustez, deve-se utilizar a arquitetura Lambda.

Aplicações do processamento do Big Data

Análise da Web

A análise da web consiste em coletar um conjunto de interações realizadas pelos usuários através da internet. Essas interações podem ser, por exemplo, através da análise de logs. Esses arquivos de logs podem conter a data e o horário de acessos, tipo de requisição enviada e o endereço IP da requisição. Após o processamento desse conjunto de dados, podem ser construídos sistemas capazes, por exemplo, de identificar as preferências dos usuários, quais itens oferecidos possuem maior probabilidade de ser aceito pelo cliente e monitorar as diferentes mudanças do consumidor.

Um outro meio de coletar esses dados é através dos “Cookies”. Os cookies representam uma estratégia utilizada para monitorar as interações que os usuários realizam através de uma determinada página Web. A partir desse monitoramento é possível obter dados sobre quais páginas foram visitadas, o tempo em que o usuário ficou em cada uma dessas páginas, número de vezes que um mesmo usuário visitou uma determinada página e um outro grande número de informações. Como é de se esperar, esse grande número de dado gera a possibilidade de realizar o “rastreamento” sobre o público alvo de um determinado produto ou serviço. Por meio do processamento desses dados, as empresas podem gerar informações valiosas e obter vantagens competitivas.

Recomendação de Conteúdo

A recomendação de conteúdo ou produtos consiste em coletar dados implícitos ou explícitos sobre as preferências dos usuários e oferecer produtos ou serviços que sejam mais adequados às necessidades dos clientes e, assim, possuem maior probabilidade de serem aceitos. Os dados implícitos são aqueles que utilizam fontes alternativas para inferir as preferências do usuário. Se um usuário assistiu um determinado vídeo até o fim, esse é um exemplo de dado implícito obtido; enquanto dados explícitos obtidos podem ser, por exemplo, o número de estrelas atribuído a

um determinado filme. Nos próximos capítulos vamos falar com um pouco mais de detalhes sobre os sistemas de recomendação.

Análise de Risco

Bancos, instituições financeiras e corretoras de seguros utilizam análise do perfil dos seus clientes para tentar medir qual seria o risco de oferecer crédito aos clientes ou mensurar os valores de apólice de seguro. Os dados para modelar essa análise são coletados de diferentes fontes, como histórico de créditos oferecidos, balanço financeiro de contas bancárias, padrões de consumo e número de transações bancárias. No Brasil existe a possibilidade de inclusão e consulta de clientes no cadastro positivo que registra os consumidores com as características de “bom pagador”. Todos esses dados são utilizados para criar os “scores” dos clientes e, assim, conseguir realizar a análise sobre qual deve ser o risco de emprestar dinheiro ao consumidor, seja através de papel moeda ou por meio de cartão de crédito.

Detecção de Fraude

Instituições bancárias e financeiras também podem utilizar sistemas de big data para detectar fraudes, como fraudes em cartão de crédito, lavagem de dinheiro e fraudes em reivindicação de seguro. As estruturas de análise de big data em tempo real podem ajudar na análise de dados de fontes diferentes e rotular, continuamente, transações que indiquem fraudes no sistema. Os modelos de aprendizado de máquina podem ser construídos para detectar anomalias nas transações e detectar atividades fraudulentas. As estruturas de análise em batelada (lote) podem ser usadas para analisar dados históricos de transações dos clientes, a fim de procurar padrões que indiquem fraude

Healthcare

Healthcare ou cuidado com a saúde também é uma área com grande potencial de utilização do processamento do Big Data. Isso ocorre pois existe uma abundância de dados que podem ser obtidos de pacientes e, através do processamento desses dados, prover diagnósticos mais precisos, maior conforto aos pacientes e ajudar na universalização ao acesso à saúde. O processamento do Big

Data pode ser utilizado, por exemplo, para realizar o monitoramento remoto de pacientes. Vários diferentes sensores como termômetros, frequencímetros, monitores de pressão arterial e da glicose sanguínea. Esses dados podem, por exemplo, ser enviados para aplicativos de smartphones que, através de algoritmos de aprendizado de máquina, podem realizar a análise contínua desses dados e, caso identifiquem alguma anomalia, podem enviar esses dados diretamente para as centrais médicas. Desse modo, medidas que visem assegurar a promoção da saúde do paciente podem ser tomadas em tempo real.

Outra área que utiliza o processamento do Big Data em prol da melhoria da qualidade de vida das pessoas e promoção da saúde, é a vigilância e controle de epidemias. Os sistemas de vigilância epidemiológica estudam a distribuição e os eventos relacionados à saúde da população. Desse modo, é possível aplicar esses estudos para o diagnóstico e controle de problemas relacionados à saúde pública. Esses sistemas de vigilância criam o Big Data através de resultados de laboratório em nível individual, dados de diagnóstico, tratamento e demográficos. Um exemplo para a aplicação desse processamento é que ocorre com a Dengue. Dados sobre locais de maior incidência, características da população e estrutura demográfica são coletados pelas secretarias de saúde, a fim de gerar o conhecimento necessário para a criação de políticas públicas que visem reduzir incidência dessa doença.

Internet das Coisas

A Internet das Coisas pode ser vista como a integração e comunicação dos objetos do nosso cotidiano (mesas, cadeiras, máquinas de lavar, geladeiras etc.) através da internet. Quando os objetos do nosso cotidiano apresentam a capacidade de comunicação autônoma entre si, temos a possibilidade de integração entre o mundo físico e o virtual. Desse modo, esses objetos podem coletar dados de diferentes aplicações e, através da internet, pode existir a interação remota com o ambiente. Por exemplo, diferentes sensores (temperatura, presença, fumaça etc.) e atuadores (alarmes, reguladores de temperatura etc.), podem ser adicionados em uma residência. Esses dados coletados são enviados para uma central que, através de algoritmos de aprendizado de máquina, constrói perfis de consumo de cada um

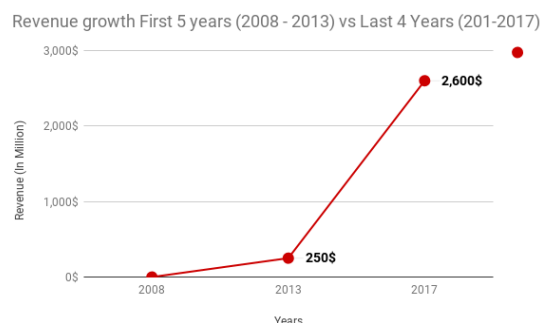
dos moradores. Assim, é possível que equipamentos como ar-condicionado e aquecedores sejam ligados apenas em horários específicos, a fim de gerar maior conforto e redução do consumo de energia.

Estudo de Caso: Airbnb

A plataforma Airbnb (Air, Bed and Breakfast) foi criada em 2008, pelos então estudantes de design Nathan Blecharczyk, Brian Chesky e Joe Gebbia. Como várias outras histórias de startups que começaram para a resolução de um problema, começou a Airbnb. Sem dinheiro para pagar o aluguel do apartamento onde moravam em São Francisco, Califórnia, e à procura de um novo projeto e da "grande ideia" para iniciar seu empreendimento, os três amigos acharam a oportunidade ideal para começar seu negócio: uma conferência de designers na cidade. Como grande parte dos hotéis da região estavam com lotação completa, decidiram alugar alguns espaços dentro do apartamento deles como a sala, cozinha e quarto dos fundos (MORALES, 2012).

A partir dessa ideia inicial eles perceberam que esse serviço de hospedagem, não tradicional, poderia ser uma opção viável aos hotéis e pousadas, uma vez que esse tipo de serviço torna o contato entre os proprietários e os hóspedes algo mais casual e diferenciado. Em apenas uma noite eles construíram a primeira versão do site de Airbnb. Quatro anos após o início das atividades, em 2012, passaram a integrar o grupo das empresas conhecidas como unicórnios (denominação atribuída a um grupo de startups que passam por um crescimento exponencial, alcançando a avaliação de US\$ 1 bilhão por uma agência de capital de risco). Existem, aproximadamente, 200 empresas que integram esse grupo de “unicórnios” no mundo. A Figura 7 mostra o crescimento da Airbnb durante os anos.

Figura 7 – Crescimento da Airbnb.



Fonte: Agriya (2017).

Em 2011, a Airbnb contratou o primeiro cientista de dados para compor a equipe de desenvolvimento e marketing da empresa. Em apenas 5 anos a empresa cresceu 43000%. A ideia central foi buscar responder a seguinte pergunta: como conectar os cientistas de dados a outras áreas? Para responder a esse questionamento foram adotadas outras três perguntas (NEWMAN, 2015):

1. Como nós (Airbnb) caracterizamos os cientistas de dados?
2. Como isso (análise de dados) está envolvido no processo de tomada de decisão?
3. Como podemos escalar o nosso negócio para levarmos o Airbnb para todos os lugares do mundo?

1 - Dados não são números, são pessoas.

Ao invés de tratar os dados e os cientistas como entidades que respondem perguntas, exclusivamente, sobre estatísticas e modelos matemáticos, eles passaram a utilizar a visão de que “os dados são a voz dos clientes”. Os dados históricos dos clientes passaram a ser empregados para compreender o porquê de o cliente ter tomado uma decisão. Assim, é possível aprender a opinião do cliente sem que ela seja, propositalmente, exposta.

Essas opiniões foram utilizadas para caracterizar o crescimento da comunidade, desenvolvimento de produtos e serviços, e priorização de recursos. Através da análise desses dados foi possível traduzir a voz do cliente em estratégias que contribuíram para a melhor tomada de decisão. A estatística foi empregada para compreender as experiências individuais dos consumidores e agregar essas experiências para identificar tendências. Por meio dessa análise foi possível ajustar o foco do negócio a partir de uma análise apurada do mercado.

2 - Parceria proativa vs. Coleta de estatísticas reativa.

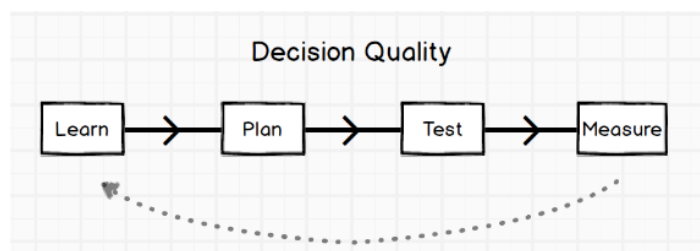
Na Airbnb foi adotada uma estratégia de que a diferença entre uma boa ideia e uma grande ideia é o impacto que ela gera. Assim, as decisões são tomadas através dos insights obtidos pela análise dos dados e são acompanhadas para que obtenham o efeito desejado.

Para assegurar que as decisões gerem os resultados esperados, é necessário que o responsável pela tomada de decisão compreenda a dimensão do insight obtido através dos dados, caso isso não ocorra, o valor obtido pelo insight será perdido. A estratégia utilizada pela Airbnb foi aproximar os cientistas de dados dos responsáveis por tomarem decisões. Com isso, foi possível levar a parceria de um cenário em que as ações eram tomadas a partir das demandas (reativa) para uma parceria em que é possível gerar ações que visem melhorar o desenvolvimento da empresa.

3 - Decisões dirigidas pelos clientes.

Uma vez que existem dados e ferramentas disponíveis para realizar uma análise sobre o comportamento dos clientes, a questão passa a ser a de descobrir como e quando utilizar a voz do cliente para que as decisões sejam assertivas. Para encontrar o tempo e a forma ideal dessa decisão, foi adotado um processo de 4 passos, a Figura 8 mostra esses passos.

Figura 8 – Processo empregado pelo Airbnb para a tomada de decisão.



Fonte: Newman (2015).

- **Learn:** aprender sobre o contexto do problema, agrupando os resultados e pesquisas passadas com as oportunidades.
- **Plan:** transformar o aprendizado em um plano. Para isso, é utilizada uma análise preditiva avaliando os possíveis caminhos e os resultados (previstos) para cada uma dessas opções.
- **Test:** à medida que o planejamento evolui, é criado um experimento controlado para avaliar a qualidade do plano escolhido.
- **Measure:** por ultimo, os resultados dos experimentos são medidos identificando os impactos causados pelas estratégias escolhidas. Caso os resultados sejam satisfatórios, o plano é colocado em prática, caso contrário o ciclo recomeça.

4 - Democratização da Ciência de dados.

Até o início de 2011 a Airbnb era uma empresa que funcionava apenas nos Estados Unidos e existia apenas 1 escritório, na cidade de San Francisco, e 3 cientistas de dados. Em menos de 6 meses foram abertos mais de 10 escritórios ao redor do mundo. Assim, para replicar o sucesso da análise de dados obtido pelo escritório central, foi necessário criar uma rede de compartilhamento de informações que pudesse ser interpretada e replicada por todas as demais sedes. Para isso, foi necessário investir em tecnologia (serviços de compartilhamento, sistemas de análise de dados e treinamento).

Com a adoção dessas estratégias, os cientistas de dados e toda a empresa conseguiram alcançar a escala mundial de serviços. Hoje a Airbnb possui um valor de mercado superior aos 40 bilhões de dólares (SCHLEIFER, 2019). Assim, é possível perceber o valor que a análise de dados pode gerar para uma empresa.

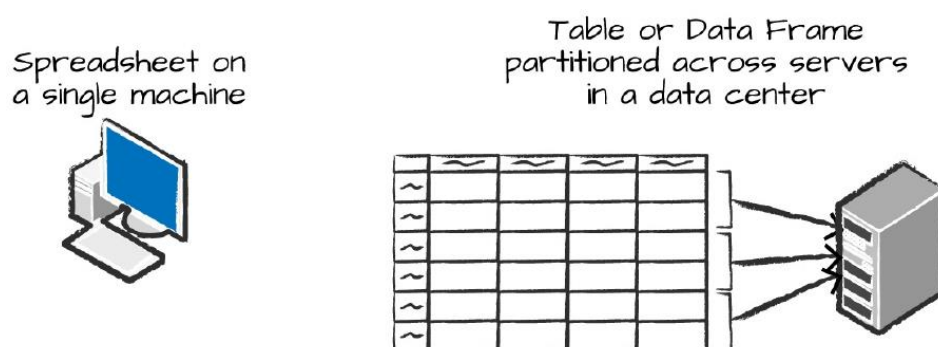
Capítulo 2. Streaming de Dados

Neste capítulo serão discutidos um dos modelos de processamento mais utilizados: o Streaming de Dados. Várias aplicações exigem que os dados coletados sejam analisados em tempo real, assim, para esse tipo de aplicação, é necessário que o sistema utilizado seja capaz de lidar com o processamento contínuo de dados.

O que é o processamento em Streaming?

A fim de explicar o que é o processamento em streaming é necessário indicar o que é o processamento através de lote de dados. A análise de dados por meio de lote (batch) consiste em examinar todo o conjunto de dados já armazenados ou grande parte dele em uma única etapa. Assim, por exemplo, dados históricos sobre vários exames de um paciente podem ser analisados a fim de identificar a progressão de uma doença, ou um conjunto de dados sobre clientes de uma grande loja podem ser utilizados para encontrar grupos de clientes com características semelhantes. A Figura 9 ilustra o modelo de processamento via lote de dados.

Figura 9 - Processamento via lote de dados.

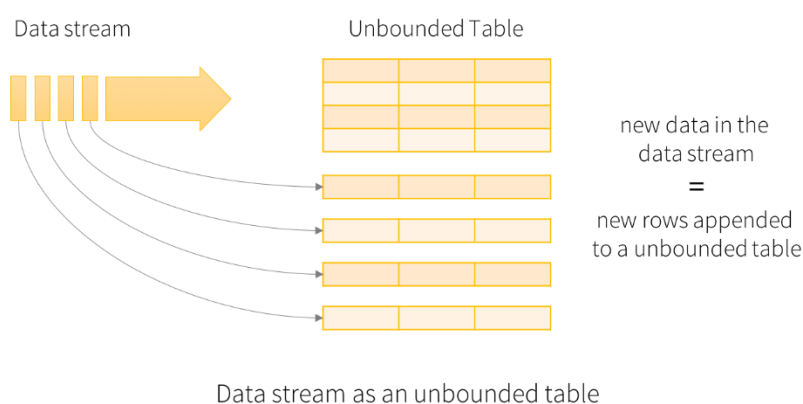


Fonte: Chambers e Zaharia (2018).

No processamento via stream de dados, como o nome sugere, os dados armazenados são, constantemente, atualizados. Cada novo dado que é adicionado à base histórica modifica esse conjunto de dados. Concomitantemente, a cada atualização é realizado um novo processamento. Desse modo, o processamento

dinâmico pode ser visto como uma análise de dados sobre um conjunto “infinito”, já que a cada momento novos dados são inseridos na aplicação. Esse tipo de processamento ocorre, por exemplo, no monitoramento remoto da saúde de paciente. Diferentes tipos de sensores (temperatura, batimentos cardíacos, pressão arterial etc.) enviam, constantemente, dados para um sistema que analisa esses dados e, caso alguma anomalia seja identificada, podem ser emitidos alertas para o médico responsável. A Figura 10 mostra a representação do processamento através do streaming de dados.

Figura 10 - Processamento via streaming.



Fonte: Chambers e Zaharia (2018).

Como pode ser visto, existem diferenças significativas entre o processamento via streaming e o realizado por meio dos lotes de dados. Essas diferenças estão resumidas na tabela 1.

Tabela 1 - Processamento via lote x Streaming.

	Processamento em Lotes	Processamento em Streams
Escopo dos dados	Consultas ou processamento de todos ou a maioria do conjunto de dados	Consultas ou processamento de dados a cada nova atualização
Tamanho dos dados	Grandes lotes de dados	Registros individuais ou micro lotes
Performance	Latências de minutos a horas	Latências da ordem de segundos ou milissegundos
Análise	Dados analíticos complexos	Métricas mais simples, agregações e rotações

Fonte: Adaptado de Chambers e Zaharia (2018).

Kafka

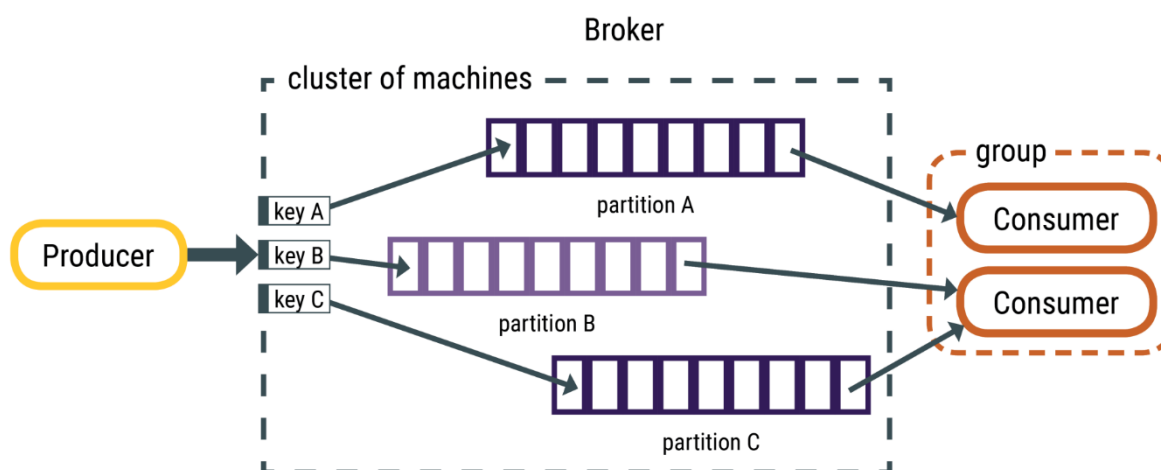
Apache Kafka é uma plataforma open-source de processamento de streams, desenvolvida pela Apache Software Foundation, escrita em Scala e Java. O projeto tem como objetivo fornecer uma plataforma unificada, de alta capacidade e baixa latência, para tratamento de dados em tempo real. Sua camada de armazenamento é, essencialmente, uma "fila de mensagens de publishers/subscribers maciçamente escalável, projetada como um log de transações distribuído", tornando-o altamente valioso para infraestruturas corporativas que processam transmissão de dados e que utilizam a IoT como meio de obtenção de dados. Além disso, Kafka se conecta a sistemas externos (para importação/exportação de dados) através do Kafka Connect e ainda fornece o Kafka Streams, uma biblioteca Java de processamento de fluxos.

O Kafka é baseado no modelo de troca de mensagens através da arquitetura publish/subscriber. Sendo assim, existem elementos que são responsáveis por publicar uma mensagem (normalmente sensores) e elementos que vão consumir essas mensagens (normalmente, aplicações de ML). Quando um sensor obtém um dado de temperatura, por exemplo, ele entrega esse dado para o Broker, que encaminha a medida lida para o elemento que se inscreveu, para ler os dados de

temperatura desse sensor. O Broker é o coração do ecossistema do Kafka. Um Kafka Broker é executado em uma única instância em sua máquina. Um conjunto de Brokers entre diversas máquinas formam um Kafka Cluster.

Uma das principais características do Kafka é a escalabilidade e resiliência que ele oferece. Você pode rodar o Kafka local na sua máquina, onde sua própria máquina teria um Kafka Broker formando um Kafka Cluster, como pode subir “n” instâncias de Kafka Brokers e todas estarem no mesmo Kafka Cluster. Com isso é possível escalar sua aplicação, e replicar os dados entre os Brokers. A Figura 11 apresenta um modelo de arquitetura para o Kafka.

Figura 11 - Modelo de arquitetura para o Kafka.



Fonte: Adaptado de Chambers e Zaharia (2018).

Apache Spark Streaming

O Spark Streaming é empregado para realizar o processamento de dados de maneira dinâmica. A programação do Spark Structured Streaming ocorre, basicamente, da mesma forma como em um modelo que utiliza o processamento estático do Spark.

Para construir uma aplicação que utiliza o Spark Structured Streaming, é necessário seguir os seis passos apresentados. Além disso, é necessário também realizar três etapas intermediárias que possibilitam a análise de dados em tempo real, sendo elas:

1. Definir a fonte de dados (de onde os dados são coletados);
2. Definir o processamento (o que fazer com os dados);
3. Definir como e onde os dados devem ser enviados após o processamento.

Esses passos definem a estrutura básica para o processamento dos dados através de streaming. A Figura 12 mostra como é definido o primeiro passo apresentado, utilizando a linguagem python.

Figura 12 - Etapas 1 e 2 para a construção de uma aplicações que utiliza o Spark Streaming.

Definindo o Processo de Streaming

```

1 #Inicia a construção do "pipeline" (define a fonte)
2 inputDF = ( spark
3     .readStream
4     .schema(jsonSchema) #esquema definido para a coleta dos dados presentes em JSON
5     .option("maxFilesPerTrigger", 1) #mantém a leitura de apenas um arquivo por batch, para manter mais lenta a coleta
6     .option("badRecordsPath", bad_records_path) #define o modo de leitura para os dados ruins"
7     .json(sensor_path) #define o local a ser pesquisado para obter os dados
8     .withColumn("INPUT_FILE_NAME", input_file_name()) #cria a coluna para armazenar o nome do arquivo o qual o dado foi lido
9     .withColumn("PROCESSED_TIME", current_timestamp()) #adiciona o tempo em que o dado foi processado
10    .withWatermark("PROCESSED_TIME", "1 minute") #adiciona a janela de tempo para a leitura (marca d'água)
11 )

```

Indica de onde o dado será obtido

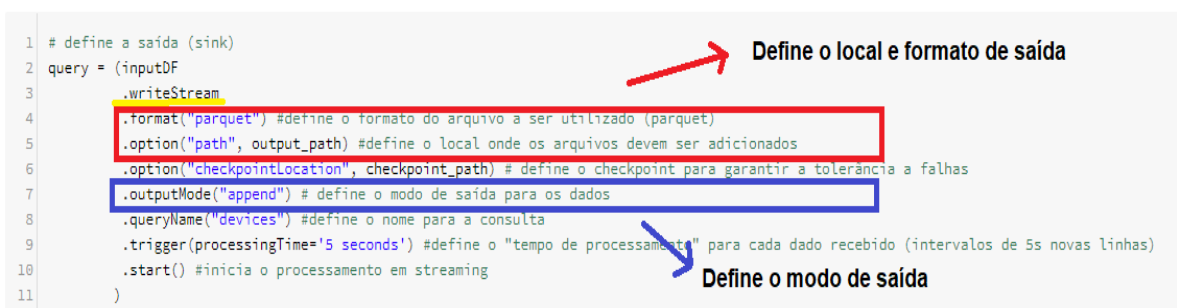
Indica o que será feito com os dados

inputDF: pyspark.sql.dataframe.DataFrame = [timestamp: timestamp, deviceId: long ... 4 more fields]

Command took 0.78 seconds -- by tulio.automacao@gmail.com at 07/11/2019 20:17:39 on Trabalho Pratico

Nessa figura é possível ver que os dados são obtidos de um caminho e que possuem o formato JSON. Além disso, é possível notar que são adicionadas mais duas colunas ao Dataframe (INPUT_FILE_NAME e PROCESSED_TIME). A Figura 13 apresenta como é utilizado o passo 3, para a construção de uma aplicação que utiliza o streaming.

Figura 13 - Construção da última etapa para a aplicação do Spark Streaming.



```

1 # define a saída (sink)
2 query = (inputDF
3     .writeStream
4     .format("parquet") #define o formato do arquivo a ser utilizado (parquet)
5     .option("path", output_path) #define o local onde os arquivos devem ser adicionados
6     .option("checkpointLocation", checkpoint_path) # define o checkpoint para garantir a tolerância a falhas
7     .outputMode("append") # define o modo de saída para os dados
8     .queryName("devices") #define o nome para a consulta
9     .trigger(processingTime='5 seconds') #define o "tempo de processamento" para cada dado recebido (intervalos de 5s novas linhas)
10    .start() #inicia o processamento em streaming
11 )
  
```

Define o local e formato de saída

Define o modo de saída

Para a entrada dos dados podem ser utilizados dados que estão armazenados em pastas distribuídas (HDFS), conectar, diretamente, o Kafka ou Kinesis, e através de conexões do tipo socket.

Para os modos de saída podem ser empregados três diferentes formas.

1. Modo Completo (Complete Mode);
2. Modo de Adição (Append Mode) - não pode ser utilizada com agregações;
3. Modo de Atualização (Updated Mode).

Esses modos indicam como os dados devem ser exibidos na saída após a realização de cada uma das consulta e operações.

Capítulo 3. Algoritmos de Machine Learning Aplicados ao Pré-Processamento do Big Data

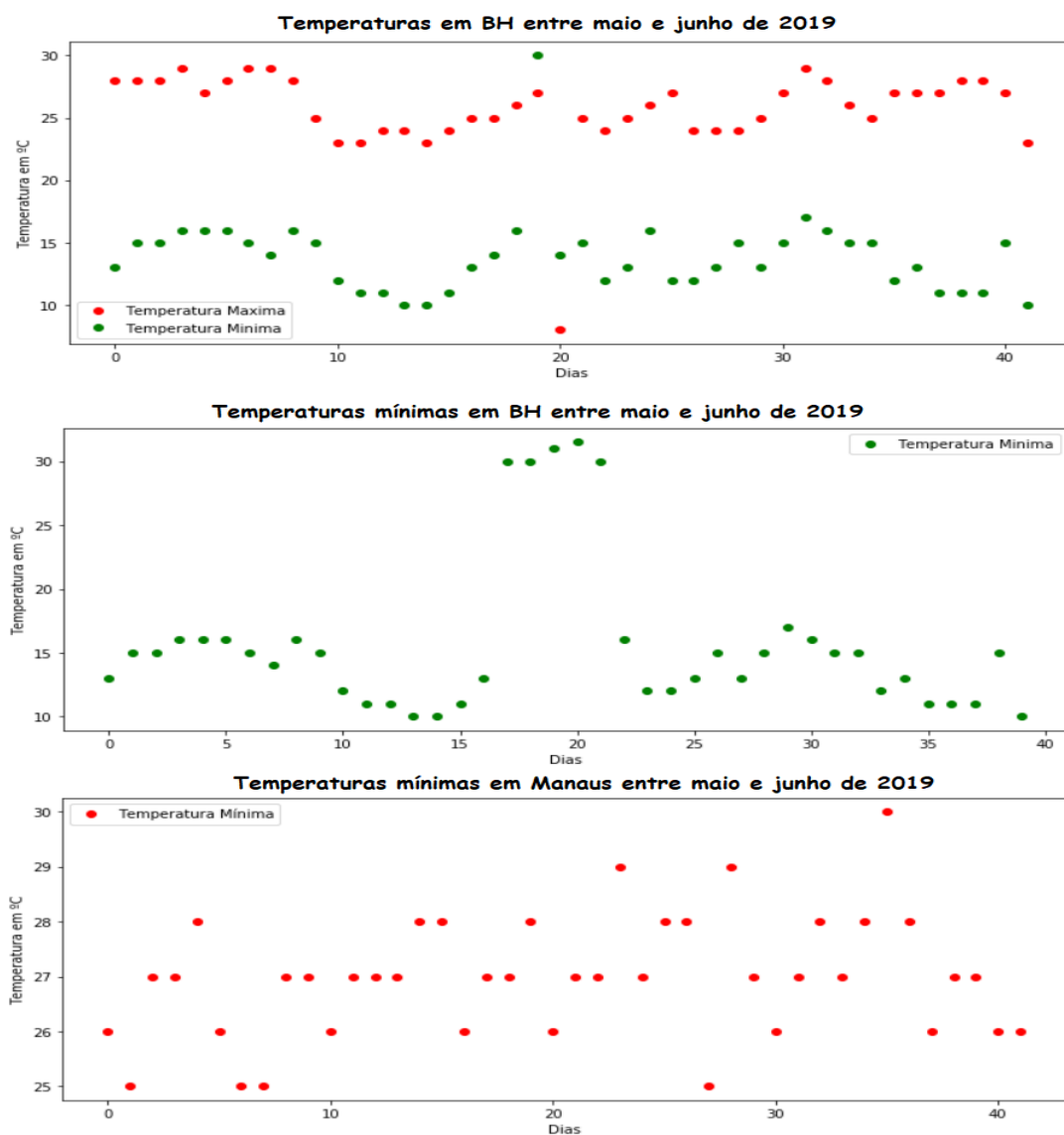
Neste capítulo são apresentados alguns conceitos sobre algoritmos de Machine Learning aplicados ao pré-processamento de dados e algumas aplicações que utilizam esses algoritmos.

Identificação de anomalias

As anomalias correspondem a dados ou conjunto de dados que possuem um comportamento diferente do esperado em um dataset. Esse comportamento fora do padrão normal dos dados pode ser um indicativo de problemas na medição ou representar um caso particular que merece ser analisado.

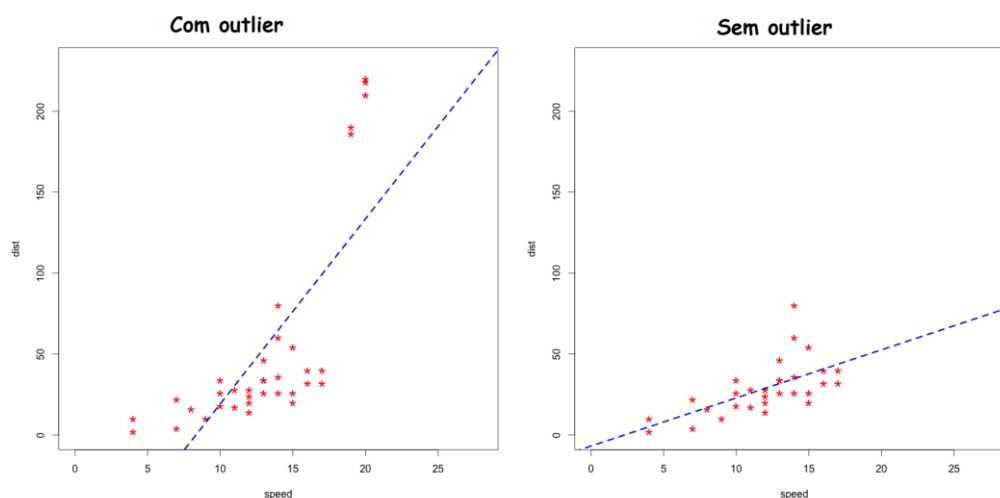
Existem, basicamente, três tipos de anomalias. Anomalias em um ponto, coletivas e de contexto. As anomalias em um ponto ocorrem quando existe apenas um valor discrepante na sequência de dados. Anomalias coletivas ocorrem quando um conjunto de dados em sequência possui um comportamento anormal e anomalias de contexto ocorrem quando uma anomalia é identificada em valores que estão fora do contexto do problema. A Figura 14 mostra exemplos desses tipos de anomalias.

Figura 14 – Exemplos de anomalias.



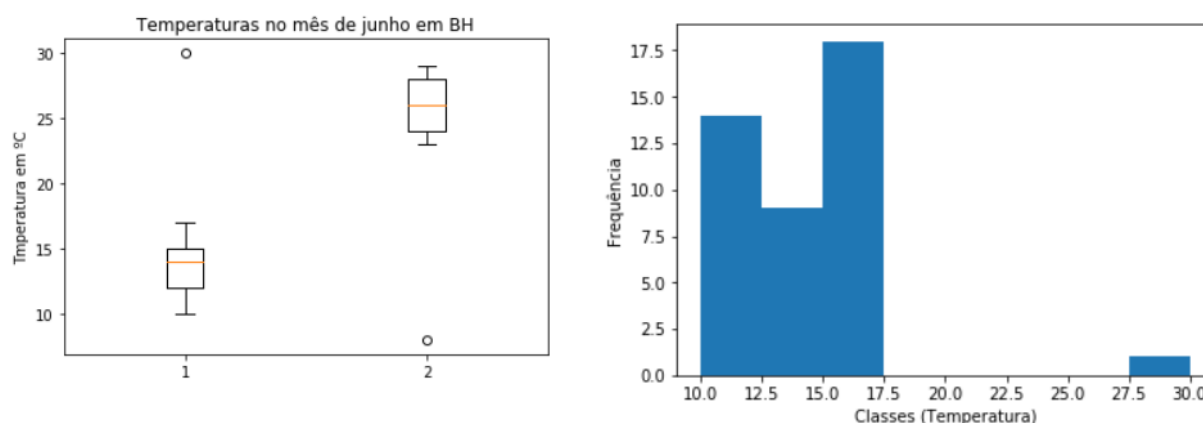
As anomalias devem ser corretamente identificadas. Isso ocorre, pois elas podem influenciar nos resultados obtidos pela análise de dados. Por exemplo, em um processo de regressão linear, as anomalias podem provocar grande variação nos valores encontrados para os coeficientes “a” e “b”. Esses valores alterados podem levar a previsões equivocadas sobre o comportamento de um sistema. A Figura 15 mostra o “erro” de previsão introduzido por outliers no processo de regressão linear.

Figura 15 – Erros introduzidos por outlier.



Existem vários processos utilizados para tentar identificar anomalias nos dados. Dentre os mais utilizados temos os métodos gráficos. Esses métodos utilizam uma análise visual para identificar possíveis valores que estejam fora de um comportamento padrão do restante do conjunto de dados. Exemplos desses métodos são o Boxplot e o histograma. A Figura 3.5 mostra a utilização desses métodos para as anomalias presentes na Figura 16. À esquerda da Figura temos o boxplot e à direita a representação do histograma.

Figura 16 – Métodos gráficos para identificação de anomalias.



Pela Figura 16 temos que no boxplot existem pontos que estão fora das “caixinhas”. Esses pontos representam os possíveis outliers no conjunto de dados.

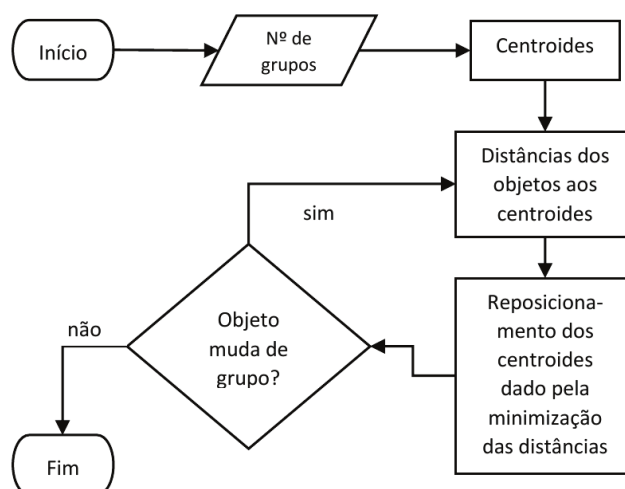
Pelo histograma é possível ver que existem dados que estão muito distantes do restante. Esse comportamento anormal sugere que esses dados representam algum tipo de anomalia nos dados.

Outros métodos mais elaborados utilizados para a detecção de anomalias utilizam alguns algoritmos de aprendizado de máquina para encontrar dados ou grupos de dados com comportamento anormal. Por exemplo, podem ser utilizados algoritmos não supervisionados para realizar o agrupamento dos dados e encontrar “clusters” de anomalias. Um exemplo de algoritmo não supervisionado, utilizado para a identificação de anomalias em dados de sensores, é o K-means.

K-means

O algoritmo K-means, como apresentado na seção anterior, é utilizado para encontrar grupos em um conjunto de dados. Para isso, o usuário deve definir o número de grupos que devem ser encontrados no Dataset. O número de grupos é definido através da quantidade de centroides escolhidos. Os centroides correspondem ao elemento responsável por realizar o agrupamento dos dados. Por exemplo, se for definido que existem dois centroides a serem encontrados no conjunto de dados, ao final da execução do K-means, devem existir dois grupos diferentes de dados.

Figura 17 – Fluxograma para a determinação dos agrupamentos pelo K-means.



Fonte: Coelho et al. (2013)

Inicialmente são escolhidos, aleatoriamente, pontos que representam os centroides no conjunto de dados. Após esse processo, são calculadas as distâncias desses centroides a cada um dos dados do dataset. Essas distâncias são utilizadas para definir os grupos iniciais para os dados. A próxima etapa consiste em reposicionar os centroides para cada um dos grupos encontrados. Depois de reposicionados, ocorre o cálculo das distâncias entre esses novos centroides e cada um dos dados. Essas distâncias são utilizadas para encontrar, novamente, os elementos que compõem os grupos. Esse processo é repetido até que os novos centroides não sofram um reposicionamento.

Para ilustrar esse processo, vamos utilizar um banco de dados contendo duas variáveis X e Y. Essas variáveis foram criadas aleatoriamente. Todos os códigos apresentados foram desenvolvidos para serem utilizados em Python 3. A Figura 17 apresenta o banco de dados desenvolvido.

Figura 17 – Banco de dados criado para o K-means.

```
#cria dados aleatórios
dados = {'x': [25,34,22,27,33,33,31,22,35,34,67,54,57,43,50,57,59,52,65,47,49,48,35,33,44,45,38,43,51,46],
        'y': [79,51,53,78,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,25,20,14,12,20,5,29,27,8,7]}
}
```

```
#cria o dataframe
df = DataFrame(dados, columns=['x', 'y'])
print (df.head())
```

```
   x  y
0  25 79
1  34 51
2  22 53
3  27 78
4  33 59
```

A partir desses dados, é utilizado o algoritmo k-means para identificar dois diferentes agrupamentos de dados nesse conjunto. A Figura 18 apresenta o procedimento para criar o objeto desse algoritmo e aplicá-lo. Todo esse procedimento foi construído para a utilização da biblioteca ScikitLearning.

Figura 18 – Construção do algoritmo k-means através do sklearn.

```
#adiciona as bibliotecas para construir o algoritmo
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=2) # cria o objeto de para o algoritmo k-means para encontrar 2 clusters
kmeans.fit(df) #aplica o algoritmo
centroids = kmeans.cluster_centers_ #encontra as coordenadas dos centroids
print(centroids)
```

```
[[38.75      61.625    ]
 [47.07142857 22.14285714]]
```

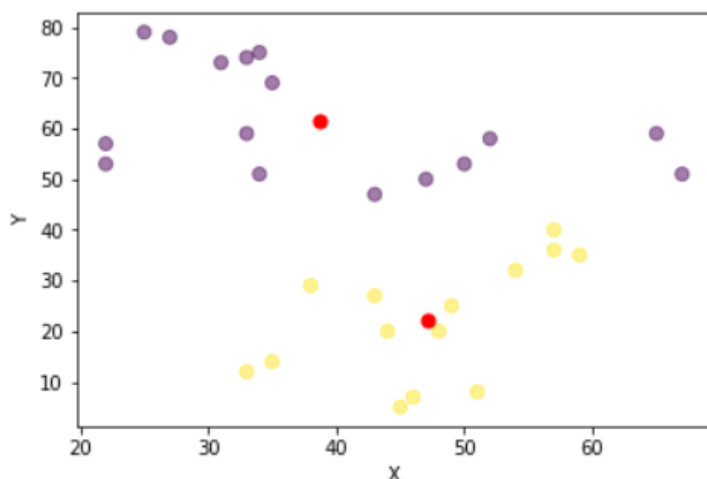
A Figura 19 mostra o resultado final após a aplicação do agrupamento realizado pelo K-means. Os códigos para essa aplicação podem ser acessados através do link abaixo:

- https://drive.google.com/open?id=1Dt3e9_rYwgn1AUXJWsT1qLIITjn94sK7.

Figura 19 – Resultado do agrupamento realizado pelo K-means.

```
#realiza o plot do gráfico da saída
plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroides[:, 0], centroides[:, 1], c='red', s=50)
plt.xlabel("X")
plt.ylabel("Y")
```

Text(0, 0.5, 'Y')



Spark MLlib

Spark MLlib é a biblioteca de aprendizado de máquina do Spark que consiste em algoritmos de aprendizagem, incluindo a classificação, regressão, clustering, filtragem colaborativa e redução de dimensionalidade. Essa biblioteca permite aplicar todo o pipeline da ciência de dados para a análise de dados, utilizando sistemas distribuídos. A Figura 20 apresenta a aplicação do pipeline da análise de dados através da MLlib.

Figura 20 - Pipeline de aplicação do Aprendizado de Máquina utilizando a MLlib.



Para a construção e uma aplicação utilizando a MLlib é necessário que sejam realizados seis diferentes passo:

- 1) Construir e iniciar a seção SPARK;
- 2) Implementar o carregamento dos dados para o Spark: carregar o arquivo, especificar o formato desejado e ler os dados como um Dataframe do Spark;
- 3) Identificar as características a serem utilizadas para treinamento e teste do modelo;
- 4) Instanciar as classes e os objetos dos algoritmos a serem utilizados;
- 5) Utilizar o método fit() para realizar o treinamento do modelo;
- 6) Avaliar o modelo.

O Apache Spark utiliza o conceito de Dataframes para realizar a análise dos dados. Um Dataframe pode ser visto como uma “planilha do Excel”. Nessa “planilha” cada uma das colunas representa um tipo de dado utilizado. Todos as linhas de uma mesma coluna possuem o mesmo formato utilizado, por exemplo, string, inteiro, float etc. No Spark, uma vez que esses dados são armazenados no sistema, eles não

podem ser mais modificados. Para realizar operações com esses dados, são realizadas “transformações”. Essas transformações indicam quais devem ser os procedimentos a serem realizados com os dados selecionados. Quando uma transformação é empregada, nenhum “job” é executado.

Técnicas para Preparação dos Dados

A preparação dos dados é fundamental para a confecção de qualquer análise. É através da preparação dos dados que boa parte dos problemas enfrentados pelos algoritmos de Aprendizado de Máquina conseguem ser solucionados. A maior parte do tempo necessário para construção de uma aplicação que utilize todo o pipeline da ciência de dados é consumida pelo processo de preparação dos dados. Portanto, sempre é importante realizar uma preparação minuciosa de todo o dataset, pois essa tarefa, quando executada corretamente, garante uma análise precisa e eficiente.

Existem várias técnicas que podemos utilizar para a preparação dos dados. Nesta seção vamos estudar as seguintes técnicas:

- Aquisição dos dados com Spark.
- Consolidação dos dados com Spark.
- Tratamento de dados faltosos.
- Tratamento de outliers.
- Tratamento de valores duplicados.
- Transformação de dados.
- Correlações.
- Redução da dimensionalidade.

A aquisição de dados corresponde à primeira etapa para a construção do Pipeline da Ciência de Dados. Nesse processo é necessário indicar o local e o formato dos dados (extensão dos arquivos), que devem ser introduzidos para o processamento da informação.

Após os dados serem inseridos no sistema e armazenados de maneira distribuída, é necessário que os dados de diferentes fontes possam ser agrupados. Dessa forma, estamos criando o Big Data que será utilizado para encontrar as informações necessárias na construção do conhecimento a partir da análise dos dados.

A partir do momento em que os dados de diferentes fontes já estão armazenados, é de extrema importância iniciar o processo de preparação. Esse processo é fundamental para que a análise dos dados ocorra de maneira eficiente. Uma das primeiras etapas para a preparação dos dados consiste em tratar dados faltosos. Como normalmente os dados responsáveis pela construção do Big Data representam aplicações reais, é comum encontrar situações em que os dados coletados e armazenados possuam um grande conjunto de variáveis. Nesse conjunto de variáveis é corriqueiro deparar com linhas ou conjuntos de linhas que apresentam valores faltosos. Esses valores aparecem, normalmente, nos sistemas representados pela sigla NaN (Not a Number). Existem vários procedimentos que podem ser utilizados para tratar os valores NaN.

- Eliminar as linhas que contém os valores NaN.
- Substituição dos valores NaN (valor zero, média dos demais valores, valores fixos ou aplicar modelos de regressão).

A substituição dos valores por um valor fixo, consiste em adotar um valor qualquer que irá apresentar todos os valores NaN existentes em uma mesma coluna ou em todo o dataset. A escolha desse valor deve ser realizada através do conhecimento prévio sobre o dataset e sobre qual é o objetivo da aplicação da análise de dados. A aplicação dos modelos de regressão consiste em utilizar os dados

presentes em uma mesma coluna e aplicar os modelos de regressão para prever qual deve ser o valor que não está presente no dataset e que deve substituir o valor NaN.

Dados duplicados também representam inconsistências que devem ser tratadas antes da aplicação dos algoritmos para o processamento dos dados. Dados duplicados podem gerar um processamento desnecessário ou adicionar um viés às análises de dados. Um procedimento comum quando são encontrados dados duplicados consiste em simplesmente eliminar as linhas que contém as mesmas informações.

Transformação dos dados também é um procedimento bastante utilizado para facilitar a visualização e análise dos dados. Colunas podem ser agrupadas, valores com unidades diferentes podem ser transformados (ex.: **m/s** para **km/h**), dados categóricos podem ser transformados em numéricos (ex.: categoria **ruim** para **0** e categoria **bom** para **1**) ou colunas podem ser transformadas em linhas (transposição de matrizes). Todas essas transformações visam deixar o procedimento de análise dos dados mais eficiente.

A correlação é uma técnica estatística utilizada para identificar se uma determinada variável está linearmente relacionada à outra. Além disso, através da correlação é possível dimensionar o quanto essas variáveis estão relacionadas (MINGOTI, 2005).

Essa correlação pode ser expressa através do coeficiente de correlação. As Equações 1 e 2 apresentam diferentes fórmulas utilizadas para o cálculo do coeficiente de correlação.

Equação 1

$$r_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} = \frac{\sigma_{ij}}{\sigma_i\sigma_j}$$

Equação 2

$$r = \frac{n \sum_{i=1}^n x_i y_i - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{\sqrt{(n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2)(n \sum_{i=1}^n y_i^2 - (\sum_{i=1}^n y_i)^2)}}$$

Em que $-1 \leq r_{ij} \leq 1$, $i, j = 1, 2, \dots, p$. Quando $i = j$, a expressão torna-se igual a 1. Os valores σ_{ij} correspondem à covariância entre os dados, e σ_i representa o desvio padrão de uma variável. Assim, é possível verificar que existem apenas três possibilidades para coeficiente de correlação.

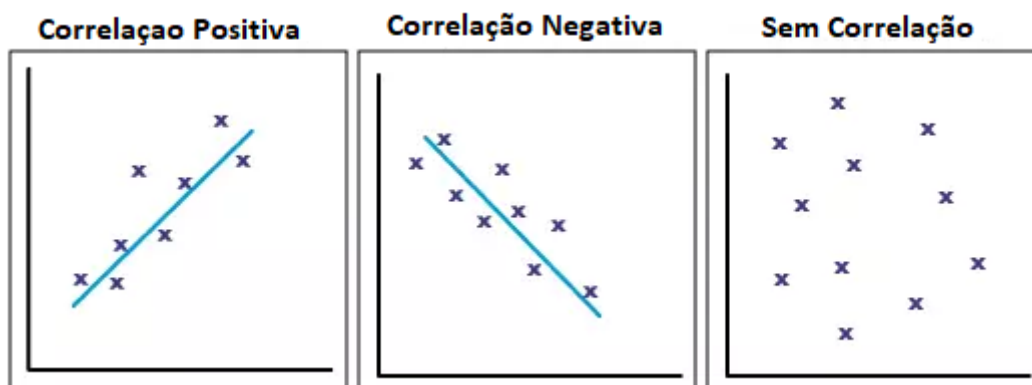
Correlação positiva: indica que o relacionamento entre duas variáveis acontece em uma mesma direção, ou seja, caso uma variável aumente, a outra também deve aumentar; e caso uma diminua, a outra também será reduzida. Um exemplo pode ser o relacionamento entre o peso e altura de uma pessoa.

Correlação negativa: indica que o relacionamento linear entre duas variáveis ocorre em direções opostas, ou seja, caso uma variável seja aumentada, a outra deve diminuir e vice-versa. Por exemplo, um aumento da velocidade, mantendo a distância constante, faz com que o tempo de viagem diminua.

Correlação zero: demonstra que não existe uma correlação linear entre as variáveis. Por exemplo, relacionamento entre gostar de café e comprar um carro.

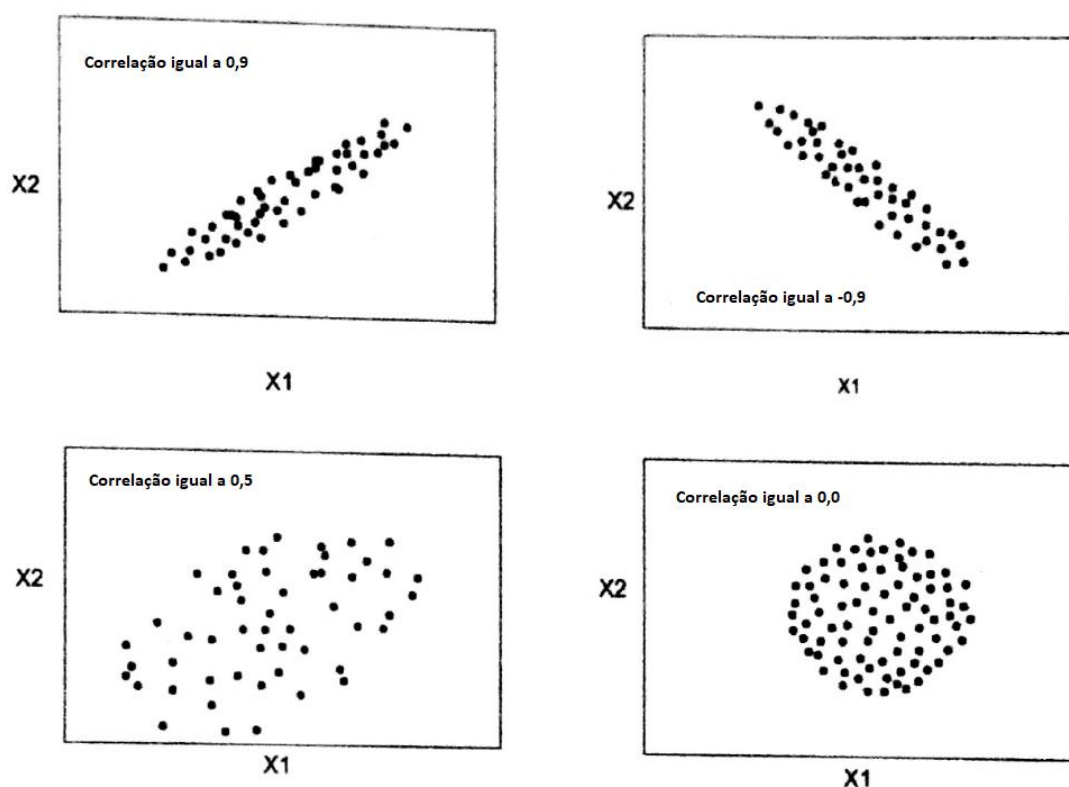
A Figura 21 mostra exemplos gráficos desses casos relatados.

Figura 21 – Tipos de correlação.



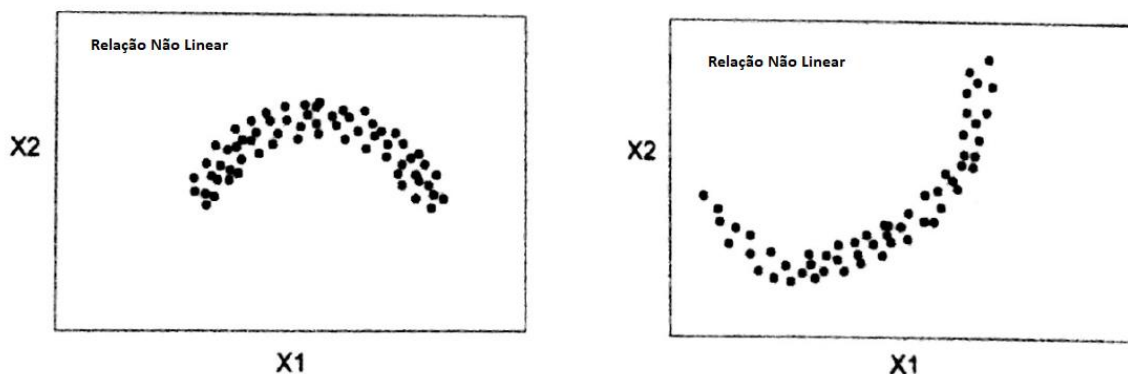
É importante ressaltar que o coeficiente de correlação apenas expressa a correlação linear entre variáveis. As Figuras 22 e 23 apresentam valores de correlação para alguns gráficos de dispersão.

Figura 22 – Diferentes valores de coeficientes de correlação.



Fonte: Mingoti (2005).

Figura 23 – Dados não linearmente relacionados.

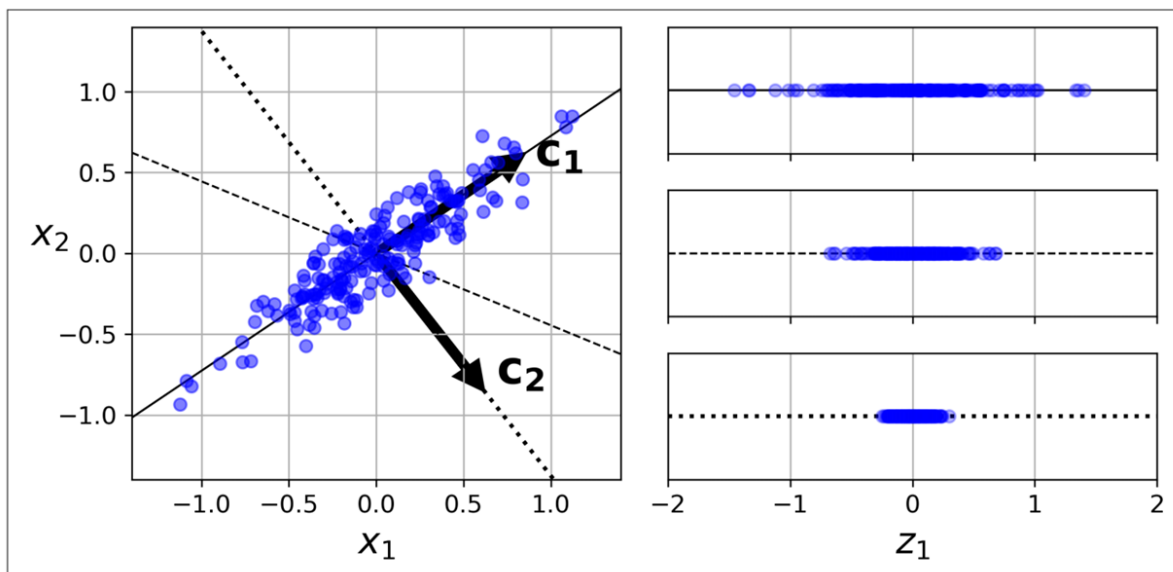


Fonte: Mingoti (2005).

Redução da dimensionalidade consiste em reduzir a quantidade de variáveis presentes em um dataset. Essa redução visa manter as características (variância) do banco de dados, mas com um conjunto menor de variáveis. Uma vez que são eliminados elementos que não contribuem ou pouco tem influência sobre a análise de dados, o processo de aplicação dos algoritmos de mineração (aprendizado de máquina) será mais rápido e eficiente sem que seja ocasionado algum prejuízo à solução do problema.

A técnica mais utilizada para a redução da dimensionalidade dos dados é conhecida como Análise de Componentes Principais (PCA, do inglês *Principal Component Analysis*). A ideia central do PCA é reduzir a dimensionalidade de um conjunto de dados, no qual existe um grande número de variáveis que estão relacionadas entre si, ao mesmo tempo que procura reter ao máximo a variação presente no conjunto de dados. Essa redução é alcançada pela transformação das variáveis em um novo conjunto, as componentes principais, as quais são descorrelacionadas e ordenadas a fim de que as primeiras componentes principais retenham o máximo da variação presente em todas as variáveis originais. Portanto, as k componentes principais, de uma coleção de n variáveis aleatórias ($k < n$), são combinações lineares especiais das mesmas e trazem consigo a maior parte da informação contida nas n variáveis originais. A Figura 24 apresenta uma representação gráfica dessa técnica.

Figura 24 – Representação gráfica do PCA.



Fonte: adaptado de Peixoto et al. (2012).

Capítulo 4. Algoritmos de Machine Learning Aplicados ao Processamento do Big Data

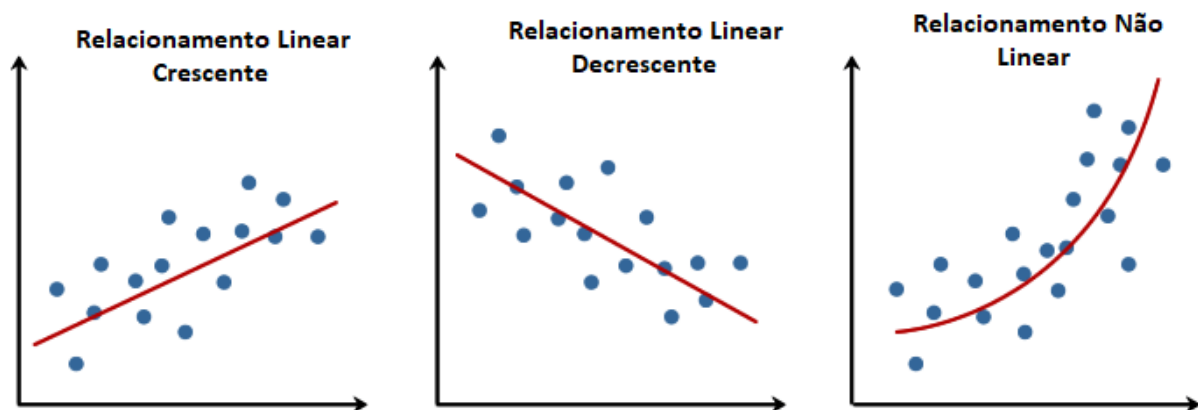
Neste capítulo serão apresentados algoritmos e técnicas empregadas para a construção de modelo capazes de realizar o processamento de um conjunto de dados e possibilitar a construção do conhecimento. Para isso, são apresentadas técnicas utilizadas para a previsão de valores, algoritmos empregados para a classificação, clusterização e regressões. A aplicação dessas técnicas tem por objetivo mostrar como os diferentes algoritmos de Aprendizado de Máquina podem ser utilizados para gerar conhecimento a partir da análise de um conjunto de dados.

Análise de regressão

As regressões são tipos de análise preditiva. Elas são utilizadas para estimar o relacionamento entre uma variável dependente e uma ou mais variáveis independentes. A variável independente é uma variável que representa uma quantidade que está sendo manipulada numa experiência. Já uma variável dependente representa uma quantidade cujo valor depende da forma como a variável independente é manipulada. As regressões são empregadas para modelar a relação entre esses tipos de variáveis.

A análise de regressão inclui vários tipos de regressões como linear, linear múltipla e não linear. As regressões lineares são, normalmente, empregadas para realizar a modelagem de sistemas simples, cuja variável dependente está relacionada apenas a uma variável independente. Já na regressão linear multivariada, a variável dependente está relacionada a mais de uma variável. Na regressão não linear, como o nome sugere, são encontradas relações não lineares para modelar a interação entre diferentes variáveis. A Figura 25 exibe alguns modelos de regressão.

Figura 25 – Exemplos de modelos de regressão.



Fonte: adaptado de Mingoti (2005).

Regressão linear

A regressão linear visa modelar o relacionamento entre duas variáveis através do ajuste de uma função linear. Uma variável é considerada a variável independente e a outra é a variável dependente. Variações na variável independente afetam, linearmente, a variável dependente. A Equação 3 mostra o modelo de regressão linear.

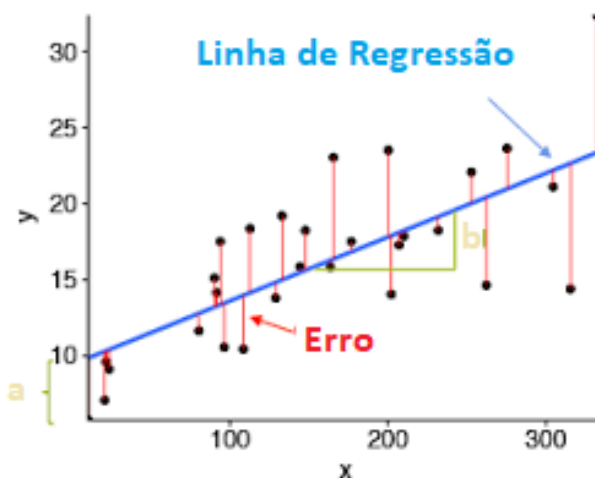
Equação 3

$$Y = a + bX$$

Onde: **X** é a variável independente e **Y** a variável dependente. O coeficiente angular dessa reta é representado por “**b**” e “**a**”, é o intercepto do eixo das ordenadas (y quando x=0).

O método mais comum para encontrar os parâmetros presentes na Equação 3 é conhecido como Mínimos Quadrados. Esse método calcula os valores que minimizam o quadrado da diferença vertical entre o valor real (medida) e a reta (prevista). A Figura 26 mostra o valor de erro (média - prevista). O quadrado da soma de todos os erros (linhas verticais em vermelho) é a função a ser minimizada.

Figura 26 – Erro minizado para a regressão.



A partir da minimização desse erro podem ser encontrados os parâmetros (“a” e “b”), que geram a reta com o menor erro médio quadrado. Essa reta é única, pois só existe uma combinação de parâmetros (inclinação e intercepto) que gera esse menor erro.

Após encontrar a reta que mais se adequa aos pontos do experimento, é necessário realizar uma análise sobre essa regressão. Essa análise normalmente é realizada através do coeficiente de determinação, sendo este coeficiente aquele que expressa o quanto a variância da variável resposta (dependente) é explicada pela variável preditora (independente). A equação 8 apresenta essa relação.

Equação 4

$$R^2 = \frac{\text{Variação explicada}}{\text{Variação total}}$$

O coeficiente de determinação também pode ser encontrado como o quadrado do coeficiente de correlação de Pearson. Desse modo, o coeficiente de determinação possui valores $0 \leq R^2 \leq 1$.

Agora vamos aplicar os conceitos de regressão linear para a solução de um problema. O problema que vamos tratar consiste em encontrar a equação da reta que relaciona a idade de um trabalhador com o salário anual recebido. A Tabela 1 apresenta os valores utilizados nesse problema.

Tabela 1 – Dados utilizados para aplicação da regressão linear.

Medida	Idade	Salário Anual
1	18	R\$ 15000
2	25	R\$ 29000
3	57	R\$ 68000
4	45	R\$ 52000
5	26	R\$ 32000
6	64	R\$ 80000
7	37	R\$ 41000
8	40	R\$ 45000
9	24	R\$ 26000
10	33	R\$ 33000

Inicialmente vamos definir as bibliotecas a serem utilizadas. A Figura 27 apresenta essa definição.

Figura 27 – Definição das bibliotecas para a regressão.

```
#regressão utilizando o otimizador
#Definindo as bibliotecas
import numpy as np #biblioteca necessária para trabalhar com os vetores e matrizes
import scipy #biblioteca necessária para obter as funções de treinamento
import matplotlib.pyplot as plt #biblioteca utilizada para construir os gráficos
from scipy.optimize import curve_fit # biblioteca necessária para realiza a otimização dos MSE
```

O segundo passo consiste em criar o banco de dados a ser utilizado pela regressão (variável dependente e independente). A Figura 28 mostra como foram definidas as variáveis presentes na Tabela 7.

Figura 28 – Definição das variáveis para a regressão.

```
#definindo as variáveis
idade=[18,25,57,45,26,64,37,40,24,33] # variável independente
salarioAnual=[15000,29000,68000,52000,32000,80000,41000,45000,26000,33000] #variável dependente

xData = np.array(idade) #transformando a lista em array
yData = np.array(salarioAnual) #transformando a lista em array
```

Como a regressão é linear, temos que definir a equação linear que deve ser ajustada aos pontos desse banco de dados. A Figura 29 mostra como é realizada a definição da equação linear a ser otimizada.

Figura 29 – Definição da equação linear.

```
#define a função a ser otimizada (regressão simples)
def equacaoLinear(x, a, b):
    return a * x + b
```

Como demonstrado é necessário que sejam gerados pontos iniciais para a construção do processo de otimização. Esses pontos são os responsáveis por gerar uma reta inicial. A partir dessa reta inicial é aplicado o processo de otimização para proceder o ajuste dos coeficientes da equação linear. A Figura 30 mostra os valores iniciais aplicados aos parâmetros “a” e “b” da reta.

Figura 30 – Valores iniciais para os parâmetros da reta.

```
#gera os parâmetros iniciais para o otimizador
parametrosIniciais = np.array([1.0, 1.0])
```

Agora que já construímos o modelo, é necessário aplicar o processo de otimização dos parâmetros da reta. Para isso será utilizada a função “curve_fit” do sklearn. Ela é responsável por encontrar os parâmetros ótimos para a equação da reta, ou seja, ela encontra os valores de “a” e “b” que geram os menores erros médios quadráticos. A Figura 31 apresenta como deve ser realizado esse processo de otimização.

Figura 31 – Processo de otimização.

```
#realiza a otimização através do erro médio quadrado (MSE)
parametrosOtimizados, pcov = curve_fit(equacaoLinear, xData, yData, parametrosIniciais)
#parametrosOtimizados - contém os parâmetros de ajuste da curva
#pcov - contém a covariância dos parâmetros encontrados
```

Para encontrar o erro gerado (valores previstos – valores reais), vamos utilizar a equação linear com os parâmetros otimizados (valores previstos) e realizar a subtração dos valores reais. A Figura 32 mostra esse procedimento.

Figura 32 – Calculo do erro na previsão.

```
#realiza a previsão dos dados através do modelo (constroi a equação linear)
pervisaoModelo = equacaoLinear(xData, *parametrosOtimizados) #utiliza a função linear com os parâmetros otimizados
```

```
#encontra o erro absoluto (linhas verticais)
erroAbsoluto = pervisaoModelo - yData #(valor previsto - valor real)
```

De posse desses valores é possível realizar o procedimento do cálculo do erro médio quadrático (MSE) e do coeficiente de determinação. A Figura 33 apresenta os cálculos realizados.

Figura 33 – Calculo das métricas de desempenho da regressão.

```
#calcula o erro quadrado entre cada medida
SE = np.square(erroAbsoluto)
#calcula o MSE
MSE = np.mean(SE)
print('SE: ', SE)
print('MSE: ', MSE)
```

```
SE: [ 4587883.33064101  6834760.82563804  413069.18137435  634115.64144357
 18436761.23538558  4467167.3659006  1517955.21070226  1424802.59030848
 873982.13514337  15601891.57382987]
MSE: 5479238.909036714
```

```
#realiza o cálculo do coeficiente de determinação
Rsquared = 1.0 - (np.var(erroAbsoluto) / np.var(yData)) # numpy.var - encontra a variância entre os dados do vetor
print('Coeficiente de Determinação:', Rsquared)
```

```
Coeficiente de Determinação: 0.9846300347582353
```

```
#mostra os parâmetros da regressão
print('Y = {}X {}'.format(parametrosOtimizados[0], parametrosOtimizados[1]))
```

```
Y = 1320.5325666669085X -6627.651716729711
```

Por último, podemos realizar o “plot” dessa equação e regressão. A Figura 34 apresenta o gráfico com os pontos utilizados e a reta que minimiza o erro médio quadrático.

Figura 34 – Gráfico de saída para o modelo de regressão.

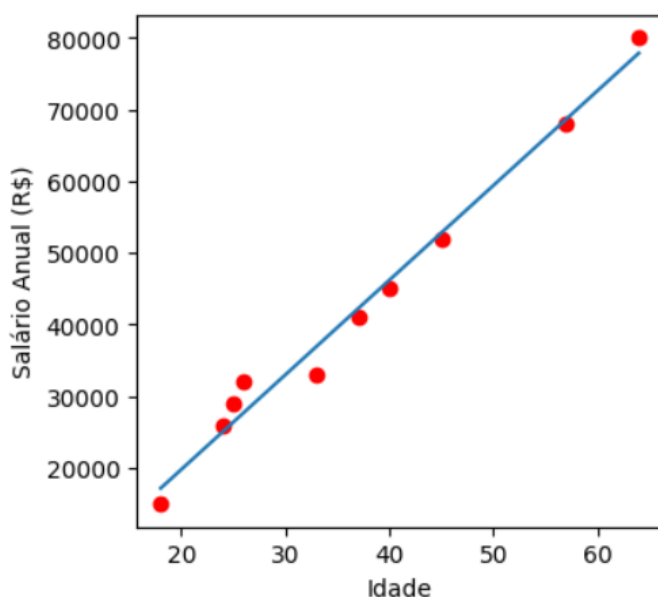
```
#realiza o plot da figura
f = plt.figure(figsize=(4, 4), dpi=100) #indica o tamanho da figura
axes = f.add_subplot(111) #cria os objetos para o subplot

# realiza o plot dos dados (pontos no gráfico)
axes.plot(xData, yData, 'ro')

# cria os dados para serem utilizados na construção da linha (equação)
xModel = np.linspace(min(xData), max(xData)) #encontra os valores máximos e mínimos da "linha"
yModel = equacaolinear(xModel, *parametrosOtimizados) # aplica a função com os parâmetros obtidos

# realiza o plot da "linha"
axes.plot(xModel, yModel)
plt.xlabel("Idade")
plt.ylabel("Salário Anual (R$)")
```

Text(0, 0.5, 'Salário Anual (R\$)')



Como pode ser visto, o coeficiente de determinação possui um valor próximo a 1. Assim, podemos ver que a variância dos salários pode ser explicada apenas pela variável idade.

A biblioteca sklearn possui funções que são muito mais simples de utilizar. No exemplo anterior foram utilizados mais passos para que fosse possível visualizar todo o processo de construção da regressão linear. Todos os códigos utilizados e, inclusive, com a utilização do sklearn, podem ser acessados através do link:

- <https://drive.google.com/open?id=1S59L4kQsdWYgNRtInMT4cx314fY2aN5g>.

KNN

O KNN é algoritmo supervisionado utilizado, normalmente, para realiza a classificação de instâncias em um conjunto de dados. O funcionamento desse algoritmo consiste em encontrar a distância entre os novos pontos adicionados e todo o conjunto de dados. A partir dessa distância é determinada a classificação desse novo elemento. A classificação é realizada através da associação dos K vizinhos mais próximos, encontrados para esse novo ponto. Os vizinhos mais próximos correspondem àqueles que possuem a menor distância para esse novo ponto. A Figura 35 mostra o cálculo das distâncias entre cada um dos dados de um dataset.

Figura 35 – Cálculo das distâncias em um conjunto de dados.

#	a1	a2	Classe
1	0.5	1	2
2	2.9	1.9	2
3	1.2	3.1	2
4	0.8	4.7	2
5	2.7	5.4	2
6	8.1	4.7	1
7	8.3	6.6	1
8	6.3	6.7	1
9	8	9.1	1
10	5.4	8.4	1
11	5	7	?

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

$$d_E(\#1, \#11) = \sqrt{(0.5 - 5)^2 + (1 - 7)^2}$$

$$d_E(\#1, \#11) = \sqrt{(-4.5)^2 + (-6)^2}$$

$$d_E(\#1, \#11) = \sqrt{20.25 + 36}$$

$$d_E(\#1, \#11) = \sqrt{56.25}$$

$$d_E(\#1, \#11) = 7.5$$

Fonte: Coelho et al. (2013).

Para a aplicação desse e dos próximos algoritmos, será utilizado o banco de dados conhecido como Iris. Esse banco de dados contém o comprimento e largura das sépalas e pétalas de um conjunto de espécies de Íris (Setosa, Virginica e Versicolor). A Figura 36 exemplifica esse banco de dados.

Figura 36 – Exemplo do banco de dados Iris.



```
#Converte o banco de dados iris para o dataframe
df_iris = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= iris['feature_names'] + ['target'])
```

```
print(df_iris.head())
```

	sepal length (cm)	sepal width (cm)	...	petal width (cm)	target
0	5.1	3.5	...	0.2	0.0
1	4.9	3.0	...	0.2	0.0
2	4.7	3.2	...	0.2	0.0
3	4.6	3.1	...	0.2	0.0
4	5.0	3.6	...	0.2	0.0

```
[5 rows x 5 columns]
```

Como esse é um algoritmo supervisionado, é necessário dividir o banco de dados em instâncias para treinamento e teste. A Figura 37 apresenta esse procedimento.

Figura 37 – Divisão do banco de dados entre treinamento e teste.

```
#transforma os dados em array
X = df_iris.iloc[:, :-1].values #dados de entrada
y = df_iris.iloc[:, 4].values # saídas ou target
```

```
#realiza a divisão dos dados entre treinamento e teste
from sklearn.model_selection import train_test_split # função que realiza a divisão do dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)# divide 20% para teste
```

Como dito no Capítulo 3, é necessário realizar um tratamento dos dados de entrada, pois os algoritmos de aprendizado de máquina, normalmente, possuem um comportamento mais natural quando os dados são tratados. Para isso, é realizada a normalização dos dados. Esse procedimento é apresentado na Figura 38.

Figura 38 – Procedimento de normalização dos dados.

```
# realiza o processo de normalização dos dados
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() #objeto que normaliza os dados
scaler.fit(X_train) #realiza a normalização dos dados

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Após o processo de normalização, é possível realizar a construção e treinamento do modelo. A Figura 39 apresenta esse processo utilizando a biblioteca sklearn. Para esse exemplo foram escolhidos cinco vizinhos para realizar a classificação dos dados de teste do modelo.

Figura 39 – Procedimento construção e treinamento do modelo.

```
#treina o modelo
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5) #utiliza a construção por meio de 5 vizinhos
classifier.fit(X_train, y_train) # aplica a classificação

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Após o processo de treinamento, pode ser aplicado o processo de previsão, ou seja, encontrar a classificação de um conjunto de dados que não foi utilizado para o treinamento do modelo. Essa previsão é realizada da forma como mostrada na Figura 40.

Figura 40 – Previsão realizada pelo modelo KNN treinado.

```
#realiza a previsão
y_pred = classifier.predict(X_test)
```

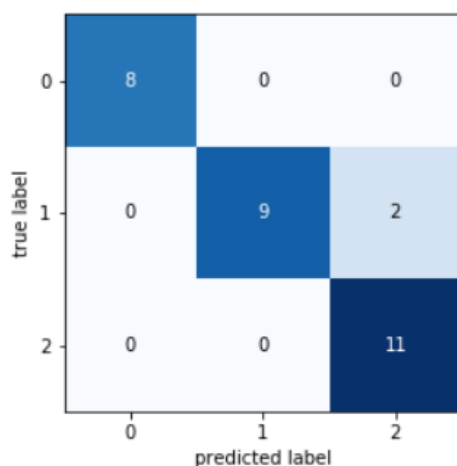
O resultado da previsão pode ser utilizado para medir o desempenho desse algoritmo. A métrica que será utilizada para comparar o resultado obtido pelo algoritmo de classificação é conhecida como matriz de confusão. A matriz de confusão proporciona uma excelente métrica de desempenho para processos de classificação, pois é possível visualizar os erros e acertos do processo para cada uma

das classes e instâncias do modelo. Assim, é possível ter acesso às taxas de classificação de cada uma das diferentes classes. A Figura 41 apresenta a matriz de confusão para o KNN.

Figura 41 – Matriz de confusão para o algoritmo KNN aplicado ao dataset Iris.

```
#realiza o plot da matriz de confusão
matriz_confusao = confusion_matrix(y_test, y_pred)
from mlxtend.plotting import plot_confusion_matrix

fig, ax = plot_confusion_matrix(conf_mat=matriz_confusao)
plt.show()
```



Pela matriz de confusão presente na Figura 41, é possível perceber, por exemplo, que o KNN classificou corretamente 11 instâncias como pertencentes à classe 2 e 2 instâncias foram, erroneamente, classificadas como pertencentes à classe 2, quando deveriam ter sido classificadas como pertencentes à classe 1. Portanto, a matriz de confusão apresenta-se como uma excelente métrica para comparar a classificação de modelos.

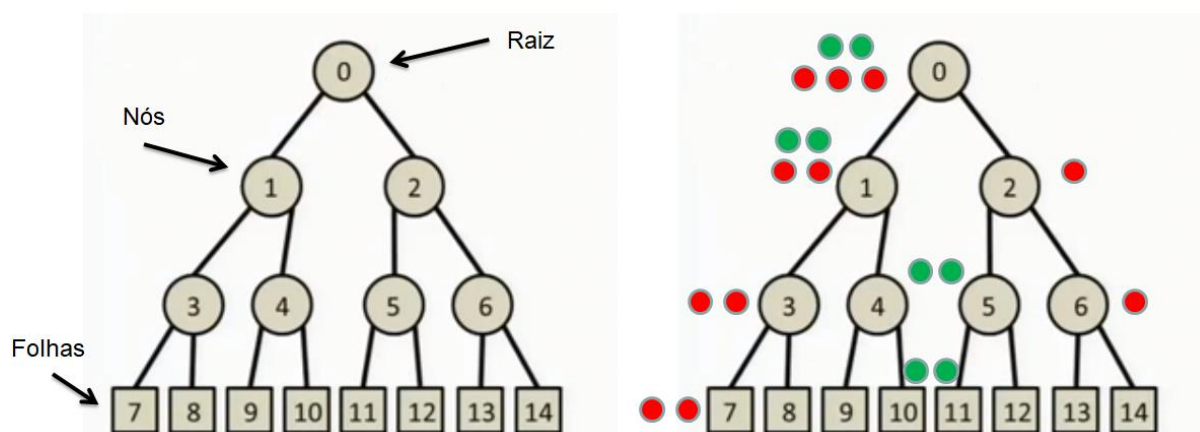
Os códigos para esse exemplo podem ser acessados através do link:

- https://drive.google.com/open?id=1Dt3e9_rYwgn1AUXJWsT1qLIITjn94sK7.

Árvore de Decisão

As árvores de decisão também são algoritmos supervisionados. Elas podem ser utilizadas tanto para resolver problemas de classificação quanto para problemas de regressão. As árvores de decisão são constituídas por um nó raiz, nós intermediários e por folhas. O nó raiz é o primeiro nó da árvore de decisão. É por esse nó que os dados são apresentados para o problema. A partir desse nó as instâncias são repassadas para os nós intermediários até que cheguem às folhas da árvore. A Figura 42 apresenta um exemplo de árvore.

Figura 42 – Árvore de decisão.



Os nós são os elementos responsáveis por realizar a análise dos dados, ou seja, é nos nós que ocorre o processo de decisão e separação do conjunto de dados. À medida que os dados descem pela árvore de decisão, vai ocorrendo uma maior separação dos dados até que, ao final do processo, nas folhas, o conjunto de dados possa ser separado.

Para o exemplo de utilização da árvore de decisão, também será empregado o banco de dados Iris. A Figura 43 apresenta como é criado e treinado o modelo de classificação utilizando árvores de decisão.

Figura 43 – Criação do algoritmo de árvore de decisão através do sklearn.

```
from sklearn.tree import DecisionTreeClassifier # importa o classificador árvore de decisão
from sklearn import metrics #importa as métricas para avaliação

# Cria o objeto de classificação através do
clf = DecisionTreeClassifier()

# Realiza o treinamento do classificador
clf = clf.fit(X_train,y_train)

#Realiza a previsão de classificação
y_pred = clf.predict(X_test)
```

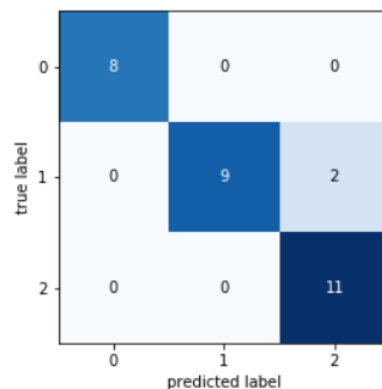
A Figura 44 mostra a matriz de confusão para a classificação do banco de dados Iris, utilizando o algoritmo árvore de decisão.

Figura 44 – Matriz de confusão para o algoritmo árvore de decisão.

```
#Avaliando o modelo

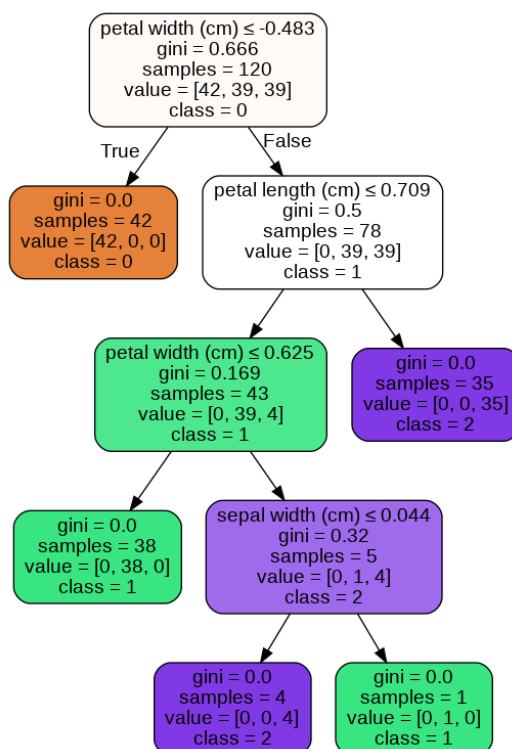
#realiza o plot da matriz de confusão
matriz_confusao = confusion_matrix(y_test, y_pred)
from mlxtend.plotting import plot_confusion_matrix

fig, ax = plot_confusion_matrix(conf_mat=matriz_confusao)
plt.show()
```



Pela Figura 44 é possível ver que a classificação através da árvore de decisão apresentou um desempenho similar ao encontrado pelo algoritmo KNN. A grande vantagem em se utilizar a árvore de decisão reside no fato de ser possível compreender todo o processo de separação realizado. A Figura 45 apresenta todo o processo de decisão realizado pelo algoritmo, até encontrar o resultado expresso na Figura 44.

Figura 45 – Processo de decisão realizado pela árvore de decisão.



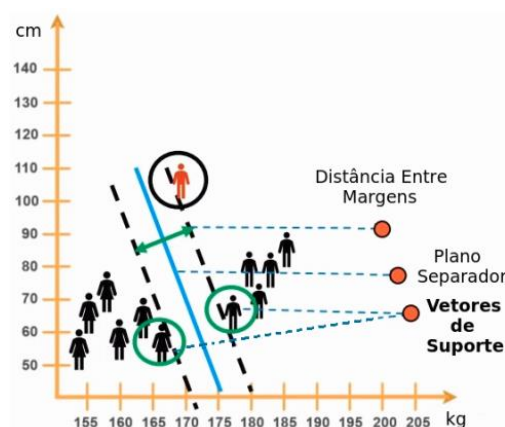
Os códigos para essa aplicação podem ser encontrados através do link:

- https://drive.google.com/open?id=1Dt3e9_rYwgn1AUXJWsT1qLIITjn94sK7.

SVM

Os Support Vector Machines (SVM) também pertencem à classe de algoritmos supervisionados. Esse são utilizados, normalmente, para realizar a classificação de um conjunto de elementos. O princípio de funcionamento desse algoritmo consiste em encontrar o hiperplano que garante a maior separação entre um conjunto de dados. Esse hiperplano é encontrado através dos vetores de suporte que, por meio do processo de otimização, encontram o hiperplano que garante a maior distância de separação entre os dados. A Figura 46 mostra, graficamente, o hiperplano (para duas dimensões é uma reta) de separação.

Figura 46 – Separação realizada pelo SVM.



Para a aplicação do SVM também é utilizado o banco de dados Iris. A Figura 47 mostra como deve ser construído, treinado e a previsão de classificação do algoritmo SVM através da biblioteca sklearn.

Figura 47 – Construção, treinamento e previsão do SVM utilizando o Sklearn.

```
#cria o objeto SVM
clf = SVC(gamma='auto') #escolhe o kernel linear

#realiza a classificação via SVM
clf.fit(X_train,y_train)

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

#Realiza a previsão de classificação
y_pred = clf.predict(X_test)
```

A Figura 48 apresenta a matriz de confusão para a aplicação do algoritmo SVM. Por meio dessa figura é possível verificar que os resultados também foram similares aos demais algoritmos de classificação. O SVM, em bancos de dados mais complexos, normalmente possui melhor resultado que os outros algoritmos demonstrados nas seções anteriores. Os códigos para essa construção podem ser encontrados através do link:

- https://drive.google.com/open?id=1Dt3e9_rYwgn1AUXJWsT1qLIITjn94sK7.

Figura 48 – Matriz de confusão para o SVM.



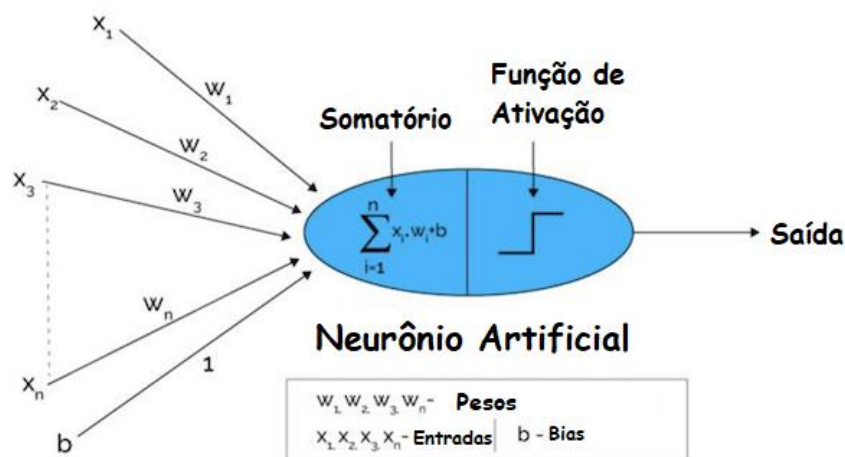
Redes Neurais Artificiais e Deep Learning

Redes neurais artificiais são sistemas de computação com nós interconectados que funcionam como os neurônios do cérebro humano. Usando algoritmos, elas podem reconhecer padrões escondidos e correlações em dados brutos, agrupá-los e classificá-los, e – com o tempo – aprender e melhorar continuamente.

A unidade fundamental de uma rede neural artificial é o perceptron. A Figura 49 apresenta um exemplo desse elemento. Os perceptrons são os responsáveis por armazenar as características de um conjunto de dados. Eles realizam essa função através do ajuste dos pesos existentes em cada conexão. Esses ajustes ocorrem durante o processo de treinamento da rede. Esse treinamento, normalmente, ocorre por meio de um processo conhecido como backpropagation. O backpropagation consiste em duas etapas. Na primeira etapa, os dados são apresentados à rede. Desse modo, ela realiza o processo do cálculo de todos os pesos e encontra uma saída estimada. Essa etapa é conhecida como feedforward, ou seja, o processamento ocorre no sentido da esquerda para direita. Na segunda etapa, é calculado o erro entre o valor encontrado no processo de feedforward e o valor real. Esse erro é

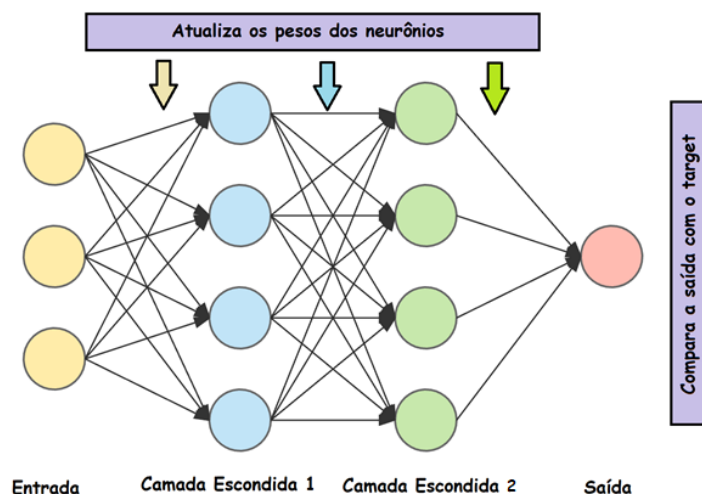
propagado no sentido inverso da rede. À medida que esse erro é propagado, é realizado um processo de otimização para que os erros sejam minimizados e os pesos sejam atualizados. Devido ao sentido inverso de propagação do erro, essa etapa recebe o nome de backpropagation.

Figura 49 – Exemplo de um perceptron.



Quando vários perceptrons estão conectados, temos a construção de uma rede neural artificial. A Figura 50 apresenta um exemplo de rede Perceptron Multicamadas (MLP). Nessa rede existem as camadas de entrada, intermediárias e saída. É no processo de treinamento que a rede neural atualiza os pesos e consegue “aprender” as características de um conjunto de dados.

Figura 50 – Exemplo de rede MLP.



Quando uma rede neural possui um grande número de camadas escondidas, temos a construção do chamado Deep Learning. A principal vantagem do Deep Learning é que existe um maior número de neurônios e, conseqüentemente, mais pesos para serem ajustados. Assim, é possível que a rede consiga “aprender” características mais complexas sobre o banco de dados. Entretanto, com isso, a complexidade computacional para o treinamento do modelo também será maior.

Para a construção da aplicação de rede neural artificial, será utilizada uma rede MLP. Essa rede será construída utilizando o Sklearn. Ao final desta apostila, no anexo B, existe um tutorial para o uso da API Keras que permite a construção de modelos utilizando o Deep Learning.

O primeiro passo para a construção do modelo consiste em definir a rede, indicando as entradas, a quantidade de camadas escondidas e neurônios em cada uma das camadas. A Figura 51 apresenta a forma como a MLP é criado no sklearn.

Figura 51 – Construção da rede MLP.

```
#definição da biblioteca
from sklearn.neural_network import MLPClassifier

#define a configuração da rede
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 5), random_state=1) #rede com 2 camadas escondidas com 5 neurônios cada
```

Como pode ser visto, foi construída uma rede com 2 camadas escondidas e 5 neurônios em cada uma das camadas. Após essa etapa é necessário realizar o procedimento de treinamento do modelo. A Figura 52 mostra como ele é realizado através do Sklearn.

Figura 52 – Procedimento para realizar o treinamento da rede MLP.

```
#realiza o fit do modelo
clf.fit(X_train,y_train)

MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

Após o treinamento do modelo, podem ser realizadas as previsões sobre a classificação. A previsão de classificação e a matriz de construção podem ser visualizadas através da Figura 53.

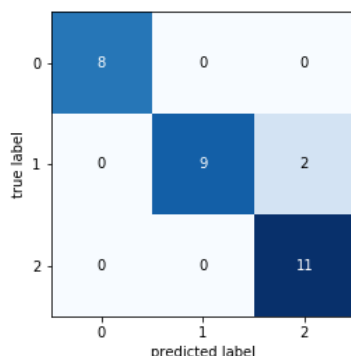
Figura 53 – Previsão e matriz de confusão para a rede MLP.

```
#realiza a previsão
y_pred=clf.predict(X_test)

#Avaliando o modelo

#realiza o plot da matriz de confusão
matriz_confusao = confusion_matrix(y_test, y_pred)
from mlxtend.plotting import plot_confusion_matrix

fig, ax = plot_confusion_matrix(conf_mat=matriz_confusao)
plt.show()
```



Como pôde ser visto, o resultado obtido é similar a todos os outros encontrados pelos demais algoritmos. As redes neurais artificiais são utilizadas em

sistemas mais complexos, em que as características do conjunto de dados necessitam de uma análise mais apurada. Os códigos para a construção dessa aplicação podem ser encontrados através do link:

- https://drive.google.com/open?id=1Dt3e9_rYwgn1AUXJWsT1qLIITjn94sK7.

Capítulo 5. Sistemas de Recomendação

Este capítulo tem como objetivo apresentar os sistemas de recomendação e mostrar a importância desses sistemas para a construção do conhecimento e melhoria das relações de consumo entre empresas e clientes.

Sistemas de recomendação

Atualmente, os consumidores possuem uma infinidade de produtos e serviços que podem ser acessados por diferentes meios. Podem realizar as compras através do telefone, presencial ou pela internet. Essa quantidade de produtos e meios de adquiri-los fazem surgir uma enormidade de opções que, muitas vezes, não são exploradas pelos consumidores.

Uma forma de apresentar novos produtos que possam interessar os clientes e que, às vezes, eles não conheciam, é por meio dos sistemas de recomendação. Os sistemas de recomendação filtram os dados de interação do usuário com o sistema de compras, utilizando diferentes algoritmos, e recomendam os itens mais relevantes para o usuário.

O primeiro passo para a construção dos sistemas de recomendação consiste na coleta dos dados. A coleta desses dados de preferência do usuário pode ocorrer de maneira explícita ou implícita. A coleta de dados explícita ocorre quando o usuário indica sua preferência ou grau de satisfação com determinado produto. Por exemplo, as notas ou estrelas atribuídas a um filme. A coleta implícita de dados é realizada através de outras variáveis que podem ser atribuídas como preferência do usuário. Esses dados podem ser, por exemplo, histórico de pesquisa por determinado produto, histórico de compras ou visualizações.

O segundo passo na direção de desenvolver o sistema de recomendação consiste em armazenar esses dados. Caso os dados coletados apresentem uma estrutura definida, normalmente são utilizados os bancos de dados SQL. Caso esses

dados sejam não estruturados, são empregados os bancos de dados NoSQL em que, muitas vezes, é realizado o armazenamento através de grafos.

Seguindo a estratégia de construção, o terceiro passo é a filtragem dos dados. É nesse passo que são identificadas as características de similaridade entre os itens ou usuários. Existem algumas formas de identificar as similaridades e recomendar os itens mais relevantes, vamos discutir duas delas:

- Filtro colaborativo baseado no usuário.
- Filtro colaborativo baseado no item.

No filtro colaborativo baseado no usuário é encontrado, inicialmente, a similaridade entre os usuários. Baseada nessa similaridade entre os consumidores, são recomendados itens que um determinado usuário gostou para outro usuário com características similares, mas que não conhece esse item. Por exemplo, veja a Figura 54, o usuário A assistiu e gostou dos filmes Central do Brasil, Cidade de Deus e Alto da Compadecida. O usuário B assistiu e gostou dos filmes Central do Brasil e Alto da Compadecida. Assim, utilizando o filtro colaborativo, é possível dizer os usuários A e B são similares. Como o usuário B não viu o filme Cidade de Deus que encontra-se no catálogo de filmes que o usuário A viu e gostou, esse filme será recomendado para o usuário B.

Figura 54 – Filtro colaborativo baseado no usuário.



A abordagem baseada nos usuários gera alguns problemas para o sistema que a implementa. Alguns desses problemas estão ligados à maior quantidade de usuário que itens no catálogo de produtos, a variação dos gostos dos usuários e a possibilidade de o usuário manipular, deliberadamente, o sistema de recomendação. Como a quantidade de usuário é, geralmente, maior que a de itens, encontrar a similaridade entre cada um dos usuários é uma tarefa, computacionalmente, mais cara. Como as preferências dos usuários variam com o tempo, a similaridade entre grande parte dos usuários deve ser constantemente atualizada, o que pode demandar um volumoso gasto computacional. Outra fragilidade desse método é que ele é mais vulnerável à manipulação de usuários maliciosos. Isso ocorre, pois, esses usuários podem criar perfis falsos e manipular as recomendações de itens através das mudanças de similaridades entre usuários.

Na abordagem por meio do filtro colaborativo baseado no item a lógica de escolha de similaridades passa do usuário para os itens. Nessa estratégia, inicialmente, são identificadas similaridades entre os itens e, posteriormente, são recomendados itens similares aos usuários. A Figura 55 mostra um cenário típico de recomendação baseado nos itens. Inicialmente, é encontrada a similaridade entre cada um dos filmes da lista. Por exemplo, gênero do filme, atores, notas dadas pelos usuários, classificação de audiência dentre outras. No exemplo da Figura 55, foi identificada grande similaridade entre os filmes Cidade de Deus e Central do Brasil. O usuário B assistiu apenas o filme Central do Brasil e gostou. Assim, será recomendado um filme similar para o usuário B, que nesse cenário, é o filme Cidade de Deus. Como pode ser visto nesse tipo de abordagem, os problemas relatados pelo filtro baseado nos usuários, são minimizados devido à similaridade estar relacionada aos itens e não aos usuários.

Figura 55 – Filtro colaborativo baseado nos itens.



Os sistemas de recomendação são muito importantes para o comércio atual. Através desses sistemas é possível melhorar a qualidade dos produtos e serviços oferecidos aos consumidores e, com isso, aumentar o percentual de vendas. Neste exemplo, vamos utilizar o banco de dados do Movie Lens. Esse dataset possui mais de 100.000 registros de filmes, usuários e notas atribuídas por esses usuários a cada um dos filmes. Para a implementação do sistema de recomendação será utilizada a biblioteca Pandas. Essa biblioteca possibilita manipular um grande volume de dados de maneira mais eficiente e amigável, através da utilização dos dataframes.

Para começar o desenvolvimento de uma aplicação, o primeiro passo é conhecer o banco de dados. Para isso, é utilizada a biblioteca Pandas para realizar o carregamento e visualização do dataset. A Figura 56 mostra esse passo inicial.

Figura 56 – Apresentação do dataset Movie Lens.

```
colunas_ratings=['user_id','movie_id','rating','time_stamp']
ratings = pd.read_csv(io.BytesIO(uploaded['u.data']),sep='t',names=colunas_ratings) # utilizado para importar o arquivo CSV que contém o banco de dados
colunas_movies=['movie_id','titulo']
movies = pd.read_csv(io.BytesIO(uploaded['u.item']),sep='|',names=colunas_movies, usecols=range(2), encoding='latin-1') # utilizado para importar o arquivo CSV que contém o banco de dados
```

[4] ratings.head()

	user_id	movie_id	rating	time_stamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

[5] movies.head()

	movie_id	titulo
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

Como os dados fornecidos pelo Movie Lens estão em dataframes separados, ou seja: um dataframe contém os usuários, o identificador dos filmes, a nota atribuída pelo usuário ao filme ao horário dessa ação, e o outro dataframe contém os identificadores dos filmes e o nome dos filmes. Para ter apenas um dataframe que contenha todas essas informações é necessário aplicar o método “merge”. Esse método realiza a união desses dois dataframes utilizando o identificador de cada um dos filmes. A Figura 57 mostra esse procedimento.

Figura 57 – Aplicação do método “merge” para o dataset Movie Lens.

```
[7] #agrupando os dataframe para ter a junção de cada usuário e filme que viu
movie_rating=pd.merge(movies,ratings, on='movie_id') #combina os dataframe por meio do id do filme
```

[8] movie_rating.head()

	movie_id	titulo	user_id	rating	time_stamp
0	1	Toy Story (1995)	308	4	887736532
1	1	Toy Story (1995)	287	5	875334088
2	1	Toy Story (1995)	148	4	877019411
3	1	Toy Story (1995)	280	4	891700426
4	1	Toy Story (1995)	66	3	883601324

De posse desse único dataframe é possível realizar algumas análises sobre os dados existentes. A primeira análise realizada corresponde à identificação dos filmes que possuem a maior média das avaliações. Para isso é utilizado o método

“groupby” do Pandas. Através desse método é possível agrupar os dados do dataframe utilizando uma das colunas. Para encontrar o valor médio das notas atribuídas é realizado o agrupamento pelos títulos dos filmes e aplicado o método “mean” sobre as notas. Assim, o método “mean” retorna o valor médio das notas atribuídas a cada filme. A Figura 58 ilustra o procedimento realizado.

Figura 58 – Aplicação do método “groupby” para o dataset Movie Lens.

```
[11] #encontrando o valor médio das notas atribuídas para cada filme
movie_rating.groupby('titulo')['rating'].mean().sort_values(ascending=False).head() # agrupa o dataframe movie_rating pela coluna titulo
#encontra a média das notas de cada filme e ordena a lista dos valores
```

titulo	rating
Marlene Dietrich: Shadow and Light (1996)	5.0
Prefontaine (1997)	5.0
Santa with Muscles (1996)	5.0
Star Kid (1997)	5.0
Someone Else's America (1995)	5.0

Name: rating, dtype: float64

Como pode ser visto pela Figura 58, filmes pouco conhecidos apresentam uma média de notas muito alta. Isso pode indicar, por exemplo, que esses filmes receberam poucas notas (foram pouco vistos). Assim, o valor médio das notas será mais alto que o esperado. Para verificar essa hipótese, é realizada uma análise sobre a quantidade de notas que cada um dos filmes recebeu. Novamente, é utilizado o método “groupby” para agrupar os filmes, mas ao invés de retornar o valor médio das notas recebidas, vamos contar o número de notas que cada filme recebeu. A Figura 59 mostra esse procedimento.

Figura 59 – Aplicação do método “groupby” para contar o numero de reviews de cada filme.

```
#conta a quantidade de notas (rating) que cada filme recebeu
movie_rating.groupby('titulo')['rating'].count().sort_values(ascending=False).head() #agrupa o dataframe movie_rating pela coluna titulo
#soma a quantidade de notas que cada filme recebeu e ordena a lista
```

titulo	rating
Star Wars (1977)	583
Contact (1997)	509
Fargo (1996)	508
Return of the Jedi (1983)	507
Liar Liar (1997)	485

Name: rating, dtype: int64

Agora, os procedimentos presentes nas Figuras 58 e 59 podem ser combinados para a construção de um dataframe que permita a análise desses dados de maneira mais rápida e, assim, verificar a hipótese criada. A Figura 60 mostra esse

dataframe criado. Pela Figura 60 é possível perceber que existem filmes com poucas notas, assim esses filmes devem ser excluídos da análise, pois não possuem dados suficientes para serem analisados.

Figura 60 – Criação do dataframe para validar a hipótese.

```
[14] #adiciona a coluna que contem a quantidade de notas que cada um dos filmes recebeu
ratings['num de ratings']=pd.DataFrame(movie_rating.groupby('titulo')['rating'].count())
ratings.head()
```

	rating	num de ratings
titulo		
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

Depois dessa análise, podem ser encontradas as correlações entre cada um dos filmes. As correlações serão obtidas a partir das notas recebidas por cada filme. Para isso, deve ser criado um dataframe que relaciona cada usuário e a nota que ele atribuiu a cada filme. O método que auxilia na criação desse dataframe é o “pivot”. Por meio desse método a coluna dos usuários é definida como sendo o pivot, assim conseguimos obter as notas que cada um dos usuários atribuiu a cada um dos filmes. A Figura 61 apresenta o resultado desse procedimento.

Figura 61 – Utilização do método pivot.

```
[18] #construção da matriz que relaciona cada usuário à nota (rating) que ele atribuiu a cada filme
movie_rating_matrix = movie_rating.pivot_table(index='user_id', columns='titulo', values='rating')
movie_rating_matrix.head()
```

titulo	'Til There Was You (1997)	1-900 (1994)	Dalmatians (1996)	101 Angry Men (1957)	12 Angry Men (1957)	187 in the Valley (1996)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	8 1/2 (1963)	8 Heads in a Duffel Bag (1997)	8 Seconds (1994)	A Chef in Love (1996)	Above the Rim (1994)	Absolute Power (1997)	Abyss, The (1989)	Ace Ventura: Pet Detective (1994)	Ace Ventura: When Nature Calls (1995)	Across the Sea of Time (1995)	Addams Family Values (1993)
user_id	1	NaN	NaN	2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	3.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	2.0	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	2.0

5 rows x 1664 columns

Utilizando o dataframe da Figura 61, pode ser aplicado o método de correlação entre cada uma das colunas (filmes). A Figura 62 mostra a matriz de correlação obtida. As correlações apresentadas nessa matriz é que serão utilizadas para indicar a similaridade entre cada um dos filmes. Valores de correlações próximos de 1 indicam filmes similares.

Figura 62 – Matriz de correlação entre os filmes.

```
[21] #refina a matriz de correlação
movie_corr_matrix=movie_rating_matrix.corr(method='pearson', min_periods=50)# utiliza o método de pearson para gerar a correlação
#apenas considera as correlações entre filmes que tiveram mais de 50 avaliações
movie_corr_matrix.head()
```

titulo	'Til There Was You (1997)	1-900 (1994)	Dalmatians (1996)	101 Angry Men (1957)	12 Angry Men (1957)	187 in the Valley (1996)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	8 1/2 (1963)	8 Heads in a Duffel Bag (1997)	8 Seconds (1994)	A Chef in Love (1996)	Above the Rim (1994)	Absolute Power (1997)	Abyss, The (1989)	Ace Ventura: Pet Detective (1994)	Ace Ventura: When Nature Calls (1995)	Across the Sea of Time (1995)	Addams Family Values (1993)
'Til There Was You (1997)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1-900 (1994)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dalmatians (1996)	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
101 Angry Men (1957)	NaN	NaN	NaN	1.0	NaN	NaN	NaN	0.176848	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12 Angry Men (1957)	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
187 in the Valley (1996)	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows x 1664 columns

Agora que foram encontradas as similaridades entre cada um dos filmes, pode ser realizada a recomendação. Adotando-se o usuário presente na Figura 63 como exemplo, vamos utilizar o procedimento apresentado na Figura 64 para encontrar os filmes a serem recomendados a esse usuário.

Figura 63 – Usuário utilizado como teste.

```
testUser=movie_rating_matrix.iloc[688].dropna()
testUser.head().sort_values(ascending=False)
```

titulo	
Contact (1997)	5.0
Conspiracy Theory (1997)	5.0
Air Force One (1997)	5.0
Courage Under Fire (1996)	4.0
Breakdown (1997)	1.0
Name: 689, dtype: float64	

O procedimento utilizado consiste em procurar na matriz de correlação os filmes que apresentam as maiores correlações com os filmes que o usuário mais gostou (atribuiu nota máxima) e recomendá-los para esse usuário. Para isso, é utilizado um loop que percorre toda a matriz de correlação em busca dos filmes mais correlacionados com o cada um dos filmes da lista do usuário.

Figura 64 – Procedimento para recomendação de filmes.

```
similarMoviesCandidates=pd.Series()
for i in range(0,len(testUser.index)):
    print("Adicionando filme similar a " +testUser.index[i]+ "...")
    #recuperando o filme que apresenta correlação
    similar=movie_corr_matrix[testUser.index[i]].dropna()
    similar=similar.map(lambda x: x* testUser[i]) #escala para 5 os filmes com maior similaridade
    similarMoviesCandidates=similarMoviesCandidates.append(similar)
```

De posse desses filmes é possível indicar aqueles que o usuário não assistiu e que são mais relacionados com os filmes que ele mais gostou. A Figura 65 apresenta os filmes recomendados para o usuário. Como pode ser visto, os gêneros dos filmes recomendados são, realmente, similares aos que o usuário mais gostou. Assim, pode ser visto que essa abordagem resultou em sistema de recomendação que pode recomendar filmes relevantes ao usuário.

Figura 65 – Resultado de filmes recomendados.

```
[28] #filtrando os filmes que o usuário já viu
      filtra_movies=similarMoviesCandidates.drop(testUser.index)
      filtra_movies.head(10)
```

Broken Arrow (1996)	36.737953
River Wild, The (1994)	32.249836
Indiana Jones and the Last Crusade (1989)	31.854684
True Lies (1994)	31.789648
Terminator 2: Judgment Day (1991)	31.411326
Back to the Future (1985)	31.262819
Dragonheart (1996)	30.697433
Top Gun (1986)	30.175430
Long Kiss Goodnight, The (1996)	30.174053
Braveheart (1995)	29.777554
dtype: float64	

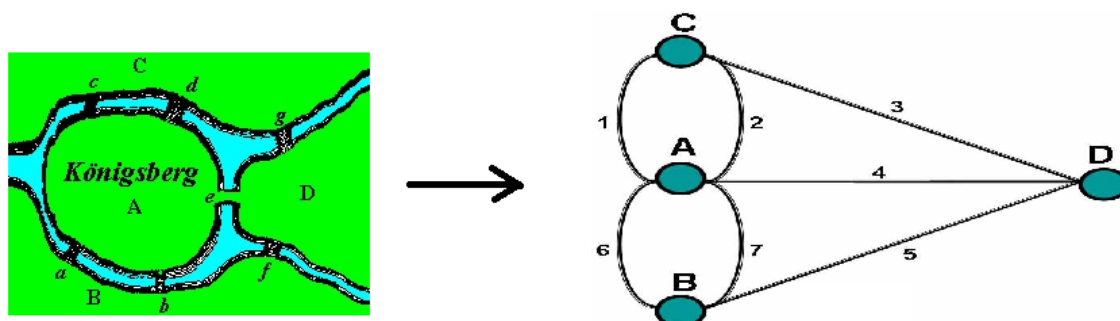
Capítulo 6. Teoria dos Grafos

Neste capítulo são apresentados os principais conceitos, ferramentas e aplicações que utilizam os grafos como forma de armazenamento e processamento de um conjunto de dados.

Introdução

Em 1736, Leonhard Euler propôs uma solução para o problema conhecido como “As Sete Pontes de Königsberg” (NEEDHAM e HODLER, 2019). Esse problema consiste em demonstrar se é possível visitar as quatro áreas da cidade (A, B, C e D) através da conexão das sete diferentes pontes (a, b, c, d, e, f, g), cruzando cada uma dessas pontes apenas uma única vez (a Figura 66 apresenta o problema). Ao resolver esse problema, Euler traçou as primeiras definições sobre a teoria dos grafos.

Figura 66 – As sete pontes de Königsberg.

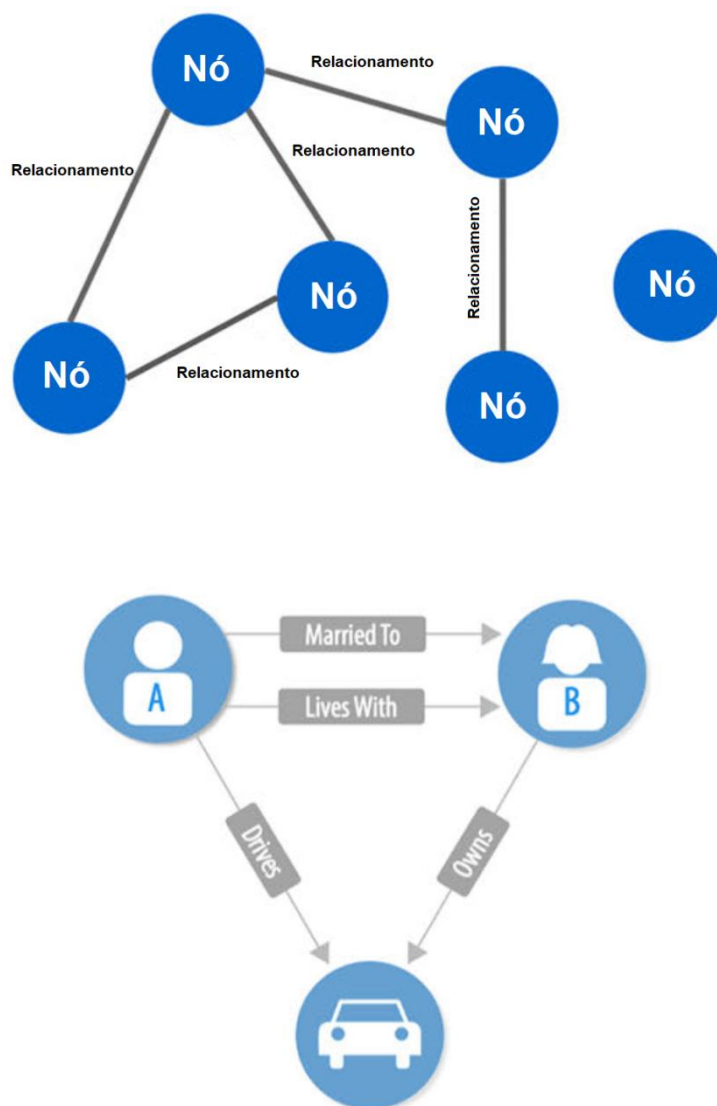


Fonte: adaptado de Needham e Holdler (2019).

Como é possível perceber, a teoria dos grafos tem origens na matemática, entretanto, ela pode ser utilizada para a modelagem dos relacionamentos existentes em um grande volume de dados. Pela Figura 66, cada uma das regiões da cidade de Königsberg (A, B, C e D) podem ser vistas como os nós (vértices) e cada uma das pontes que ligam essas regiões podem ser vistas como as arestas.

Quando os grafos são empregados para a representação de um grande volume de dados, é comum atribuir o nome de “nós” para cada um dos nomes existentes no conjunto de dados e de “relacionamentos”, como sendo os verbos presentes em cada uma das sentenças. A Figura 67 mostra a representação dos nós e relacionamentos para um conjunto de dados.

Figura 67 – Nós e relacionamentos.

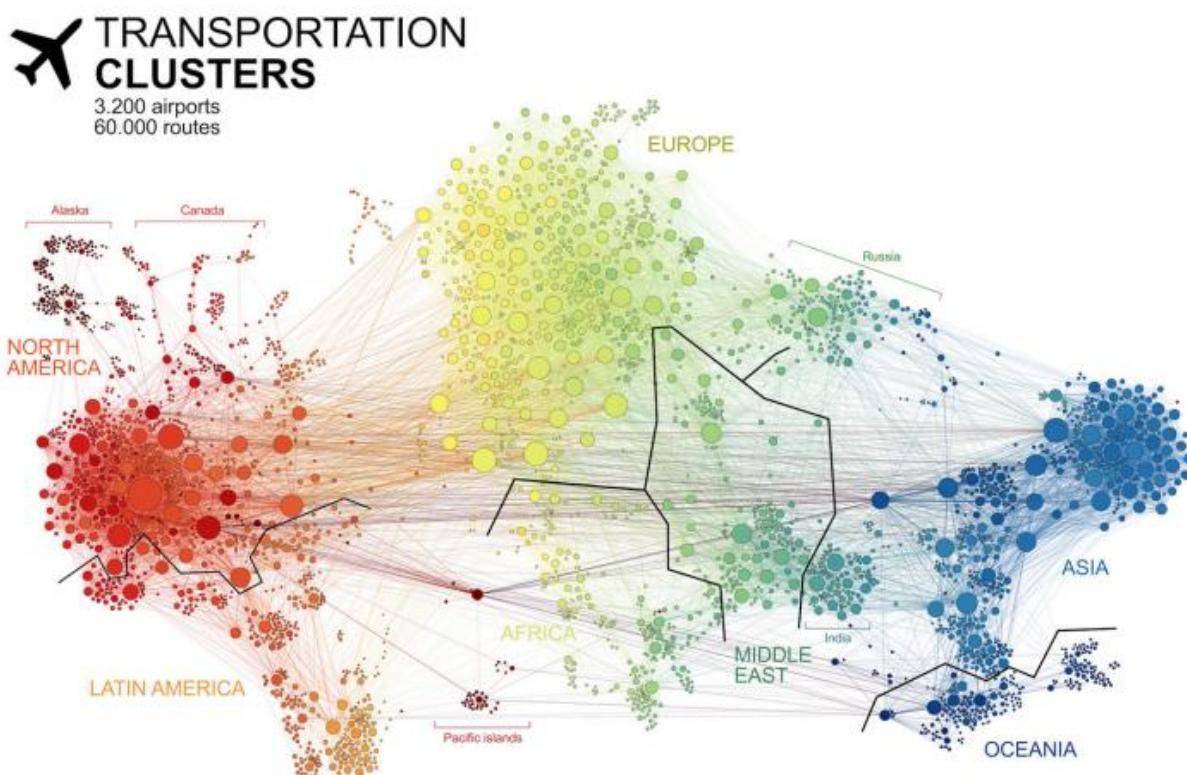


Fonte: adaptado de Needham e Holder (2019).

Ao analisar a Figura 67 é possível construir um conjunto de sentenças que descreve o banco de dados representado. Por exemplo, o indivíduo A vive e é casado com a pessoa B e também dirige o carro pertencente à pessoa B.

Uma vez que os dados são modelados a partir de grafos, é possível empregar diferentes algoritmos para processar toda a informação contida nos relacionamentos existentes entre cada um dos nós. A partir do processamento através de grafos, um conjunto complexo de estruturas pode ser analisado e informações valiosas podem ser obtidas. A Figura 68 apresenta uma ilustração dos aeroportos mundiais e as conexões existentes entre cada um deles.

Figura 68 – Conexão entre aeroportos ao redor do mundo.



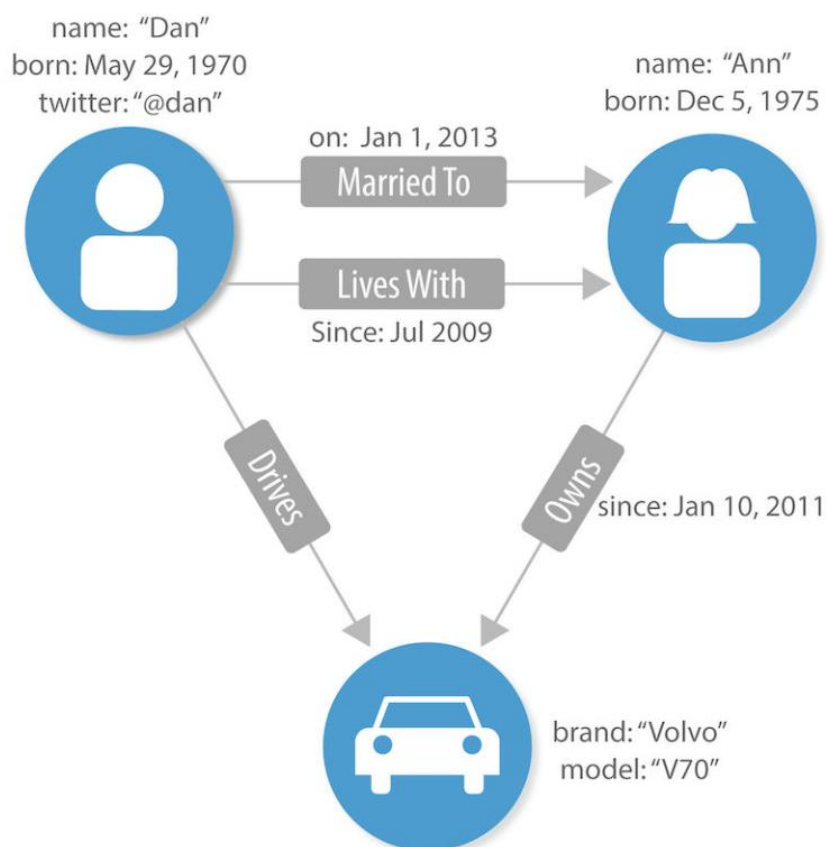
Fonte: Needham e Holdler (2019).

Conceitos e terminologias

A Figura 69 apresenta um exemplo de um grafo rotulado. Os grafos rotulados consistem em um grafo em que cada um dos **nós** (arestas) e **relacionamentos** possuem um rótulo. Por exemplo, na Figura 69, é possível identificar dois diferentes grupos de **nós** (Pessoa e Carro) e quatro tipos de **relacionamentos** (Drives, OWNS,

LIVES_WITH e MARRIED_TO). Também por essa figura é possível identificar alguns atributos para cada um dos nós e relacionamentos. Por exemplo, nome, data de nascimento, data de casamento, marca e modelo de automóvel são chamados de “**propriedades**”. As “**propriedades**” são armazenadas através do modelo **nome-valor**, assim, a cada nome (**atributo**) é atribuído um valor.

Figura 69 – Grafo rotulado.



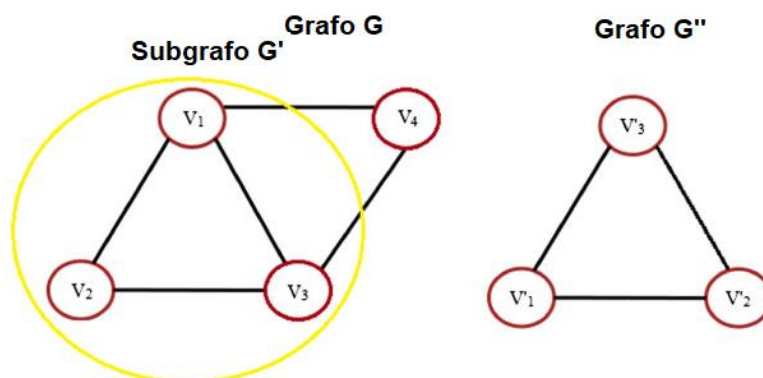
Fonte: adaptado de Needham e Holder (2019).

Os **caminhos** em um grafo correspondem ao conjunto de nós e relacionamentos entre esses nós. Por exemplo, na Figura 69 um caminho pode conter os nós Dan, Ann e o carro, além dos relacionamentos “Married to” e “Owns”.

Quando é necessária a utilização de apenas uma porção do grafo para a construção de uma característica particular, por exemplo, temos o conceito de

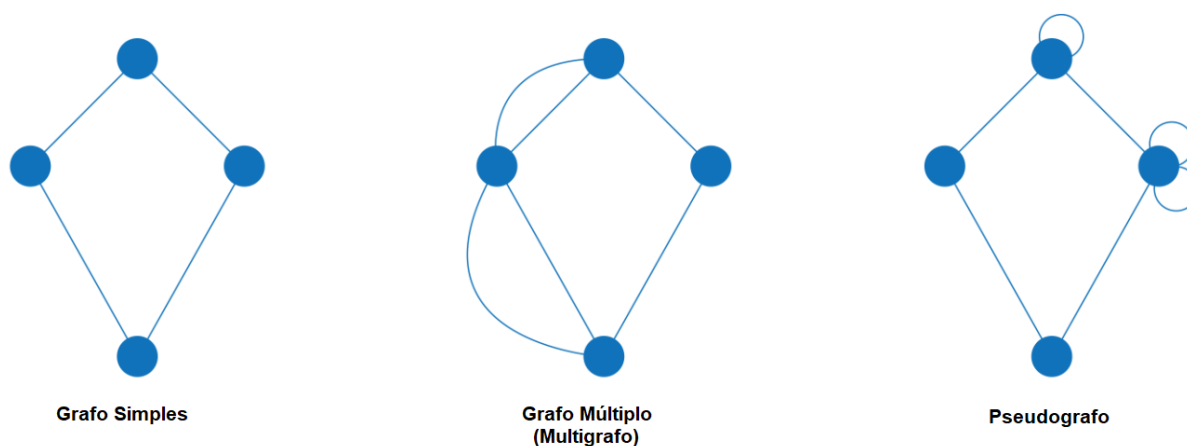
“**subgrafo**”. O **subgrafo** constitui-se na representação de uma porção menor de um grafo. A Figura 70 apresenta a representação de um grafo e um possível subgrafo.

Figura 70 – Grafo e subgrafo.



Existem, basicamente, três diferentes tipos de grafos: Grafos **Simples**, **Grafos Múltiplos** (Multigrafo) e **Pseudografo**. Nos **Grafos Simples** são permitidos apenas um relacionamento para cada um dos pares de nós. Já nos **Grafos Múltiplos**, podem existir múltiplos relacionamentos entre cada um dos pares de nós e no **Pseudografo**, cada nó também pode possuir “loop” de relacionamentos entre ele mesmo. A Figura 71 ilustra esses tipos de grafos.

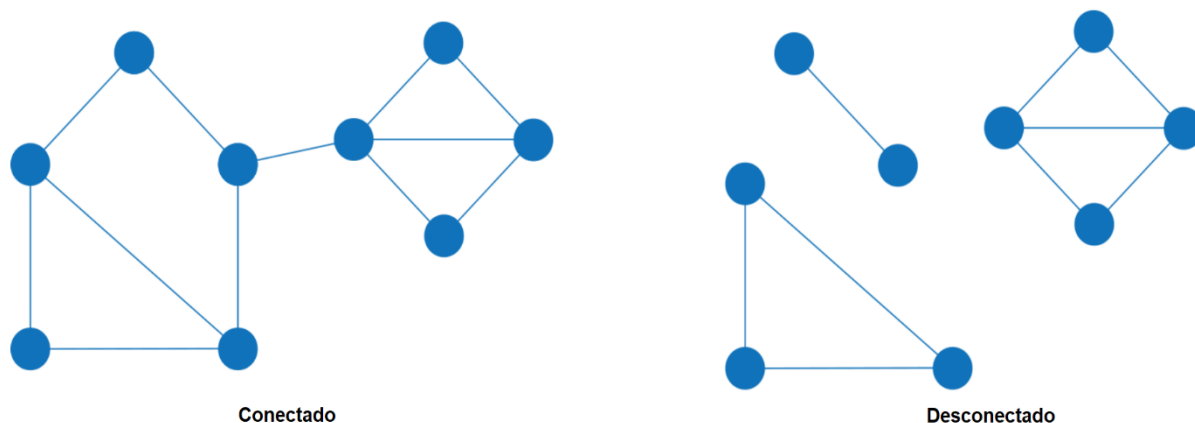
Figura 71 – Tipos de grafos.



Fonte: adaptado de Needham e Holdler (2019).

Cada um dos tipos de grafos pode ser **conectado** ou **desconectado**. Nos grafos **conectados** existe um caminho possível para chegar a cada um dos nós. Já em grafos **desconectados**, não é possível visitar um nó ou conjunto de nós partindo. Desse modo, existem **ilhas** que não possuem conexão direta com outros nós. A Figura 72 apresenta a representação desses grafos.

Figura 72 – Grafos conectados e desconectados.



Fonte: adaptado de Needham e Holdler (2019).

Grafos também podem possuir peso ou não. Esses pesos podem ser atribuídos aos nós ou aos relacionamentos. Os pesos podem ser valores que representam custo, tempo, distância, prioridade etc. Desse modo, cada relacionamento ou nó possui valores que representam o comportamento dentro da rede. A Figura 73 apresenta esses tipos de grafos.

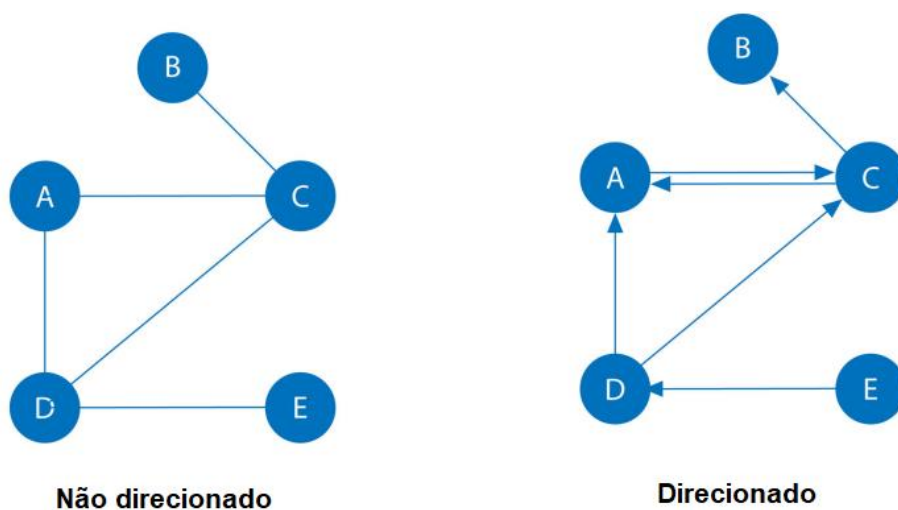
Figura 73 – Grafos com pesos e sem pesos.



Fonte: adaptado de Needham e Holdler (2019).

Grafos também podem ser classificados entre direcionados e **não direcionados**. Nos grafos **direcionados**, cada um dos relacionamentos possuem um sentido. Nos grafos **não direcionados**, cada um dos relacionamentos é considerado como sendo bidirecional, ou seja, os relacionamentos podem ocorrer em ambos os sentidos. A Figura 74 apresenta a ilustração para os grafos direcionados e não direcionados.

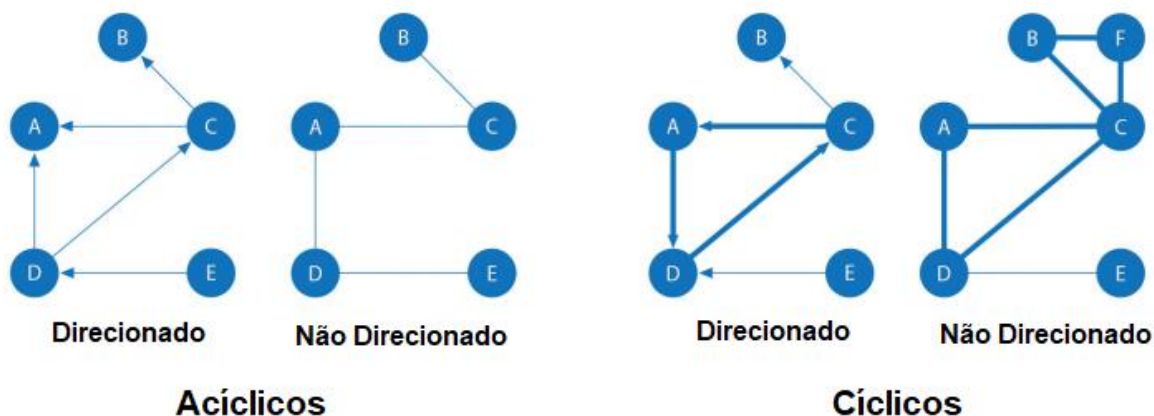
Figura 74 – Grafos direcionados e não direcionados.



Fonte: adaptado de Needham e Holdler (2019).

Outra definição importante corresponde aos grafos **cíclicos** e **acíclicos**. Nos grafos **cíclicos**, existe um ou mais caminhos que começam e terminam em um mesmo nó. Tanto grafos **não direcionados** quanto **direcionados** podem possuir ciclos. Já nos grafos acíclicos, não existe a possibilidade de iniciar e terminar um caminho em um mesmo nó. A Figura 75 mostra um exemplo desses grafos.

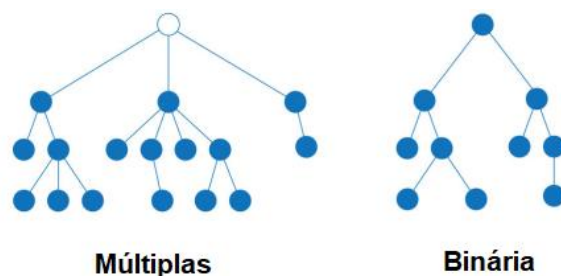
Figura 75 – Grafos cíclicos e acíclicos.



Fonte: adaptado de Needham e Holdler (2019).

Árvores também são comumente encontradas por meio da representação através de grafos. **Árvores** constituem um tipo de grafo acíclico em que dois nós consecutivos são conectados apenas através de um caminho. A Figura 76 ilustra dois tipos de árvores. **Árvores** com múltiplos filhos e **árvores** com nós possuindo apenas dois filhos (binária).

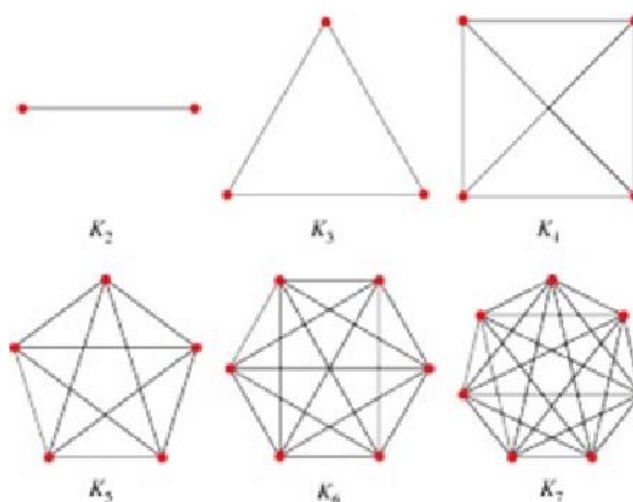
Figura 76 – Representação de árvores.



Fonte: adaptado de Needham e Holdler (2019).

Grafos são considerados **completos** quando cada um dos nós possui, pelo menos, um relacionamento com um dos outros nós, ou seja, cada um dos vértices (nós) é adjacente a todos os outros vértices. A Figura 77 ilustra alguns exemplos de grafos completos.

Figura 77 – Grafos completos.



Fonte: adaptado de Needham e Holder (2019).

A **densidade de um grafo** é a medida que indica a quantidade de relacionamentos existente em um grafo. O número máximo de relacionamentos possíveis em um grafo é conhecido como a **máxima densidade** de um grafo. Ela é dada pela Equação 5.

Equação 5

$$MaxD = \frac{N(N-1)}{2}$$

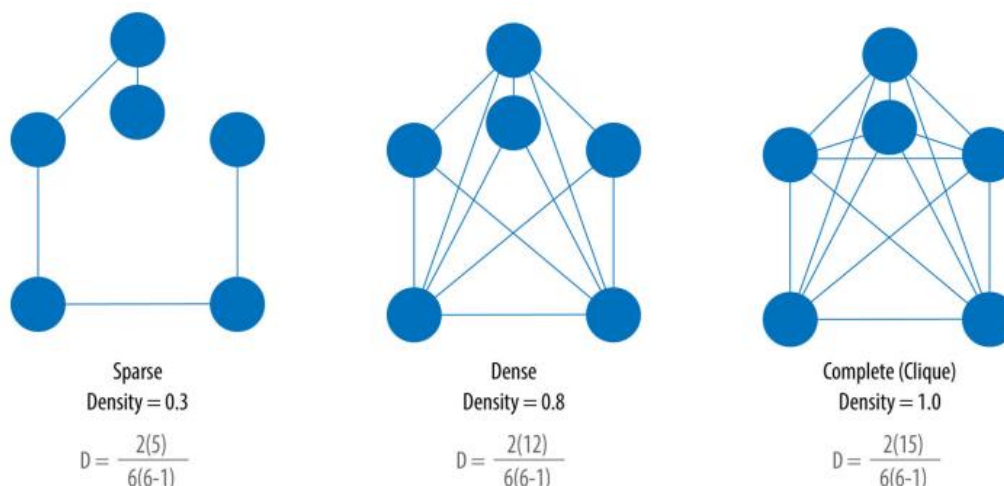
Em que N é o número de nós existente no grafo. A **densidade real** pode ser obtida através da Equação 6:

Equação 6

$$D = \frac{2(R)}{N(N-1)}$$

Em que R representa o número de relacionamentos existente no grafo. A Figura 78 mostra alguns exemplos de cálculo para densidade em cada um dos grafos.

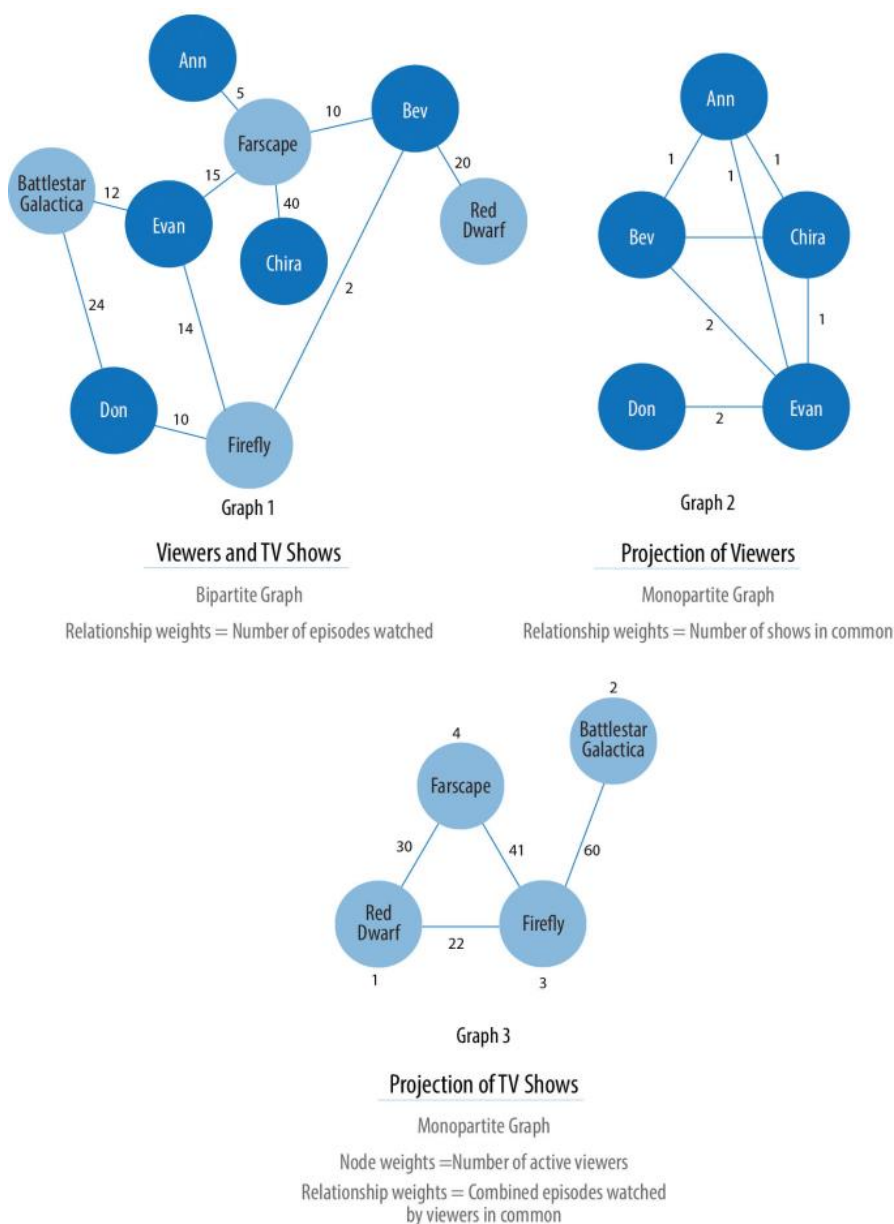
Figura 78 – Cálculo da densidade para alguns grafos.



Fonte: Needham e Holder (2019).

De acordo com os tipos de nós e relacionamentos, os grafos podem ser classificados em **monopartido**, **bipartido** ou **K-partido**. Um grafo é dito **monopartido** quando possui nós e relacionamentos de apenas um único tipo. Um grafo **bipartido** ou bigrafo, corresponde a um grafo que pode ser dividido em dois conjuntos distintos, tal que existam relacionamentos apenas entre elementos de grupos distintos. Grafos **K-partido** refere-se ao número de diferentes nós existente no grafo (K). A Figura 79, apresenta exemplos de grafos bipartidos e monopartidos.

Figura 79 – Exemplo de grafos bipartido e monopartido.



Fonte: Needham e Holdler (2019).

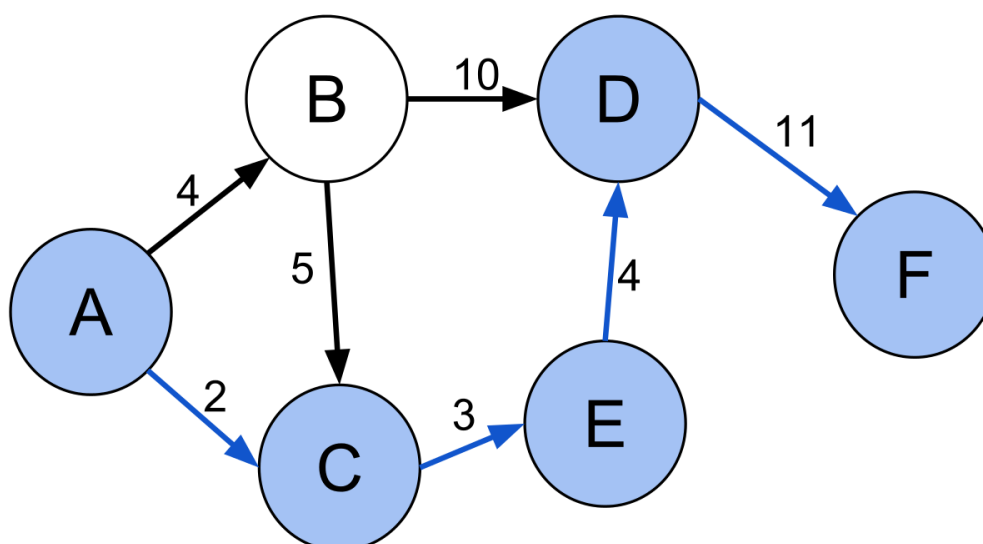
Algoritmos para grafos

Os algoritmos são os responsáveis por obter informações sobre os dados armazenados através de grafos. Existem diferentes algoritmos que, quando empregados da forma correta, possibilitam obter “conhecimento” por meio da análise

dos grafos. Existem famílias de algoritmos que são utilizados para encontrar caminhos, encontrar a centralidade em um grafo e realizar a detecção de comunidades.

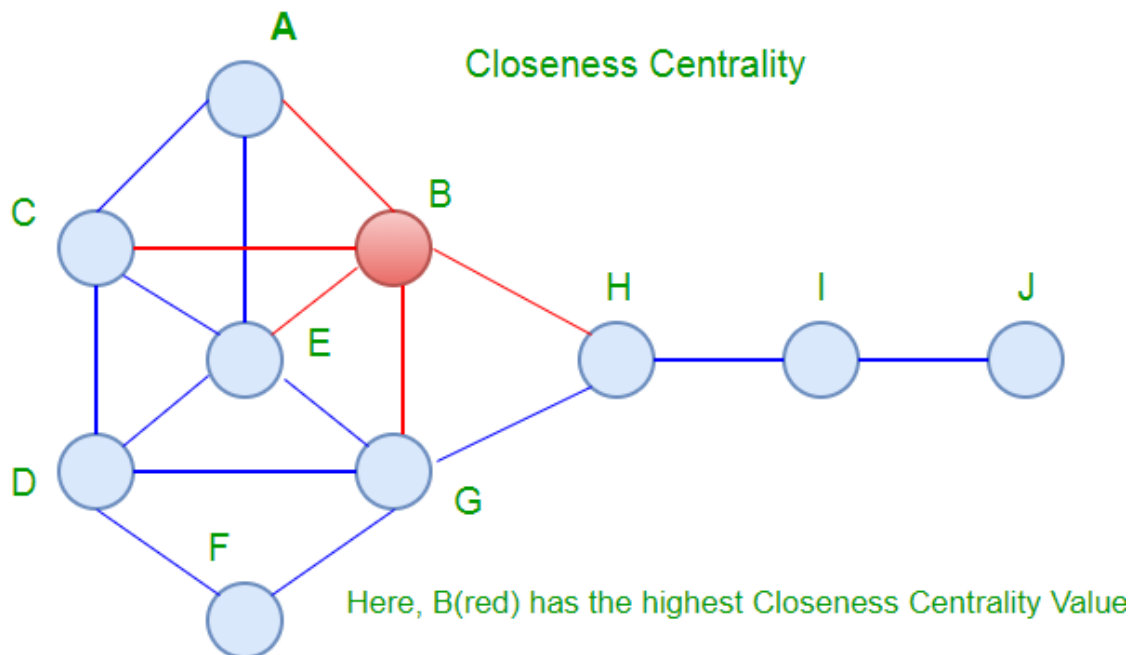
Os algoritmos responsáveis por **encontrar caminhos** são utilizados para recomendar caminhos a partir da análise dos nós e relacionamentos existentes no grafo. Os algoritmos mais conhecidos são os empregados para encontrar o menor caminho existente em um grafo. Problemas como do “caixeiro viajante” são abordados por esse conjunto de algoritmos. A Figura 80 apresenta um problema e o resultado obtido após a aplicação do algoritmo.

Figura 80 – Exemplo de resultado após a aplicação do algoritmo do menor caminho entre os nós A-F.



Os algoritmos que são responsáveis por encontrar quais são os nós mais importantes em uma rede, são conhecidos como **algoritmos de centralidade**. Existem vários algoritmos e estratégias que podem ser utilizadas para encontrar a centralidade de um grafo. A Figura 81 apresenta um desses algoritmos, que o nó B apresenta o maior valor de centralidade.

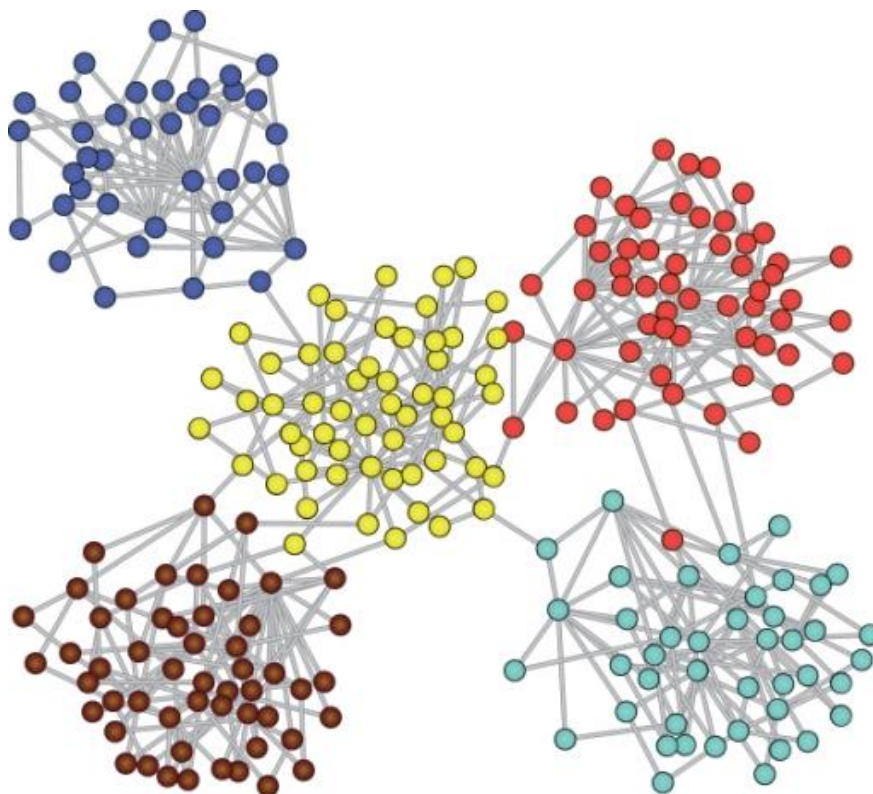
Figura 81 – Exemplo de resultado após a aplicação do algoritmo de centralidade de proximidade.



Fonte: adaptado de OKAMOTO et. al (2008).

Deteccção de comunidade também representam um conjunto de algoritmos bastante utilizado para a análise através de grafos. Como o nome sugere, esses algoritmos são empregados para encontrar grupos de nós com características similares e tendências. A Figura 82 ilustra a utilização desse algoritmo em um grafo.

Figura 82 – Exemplo de resultado após a aplicação do algoritmo de detecção de comunidade.



Fonte: adaptado de FORTUNATO (2010).

Neo4j e Cypher

Neo4j é um sistema de construção e gerenciamento de banco de dados grafos, através do qual é possível armazenar e processar dados por meio desses grafos. O Neo4j é um sistema em código aberto com uma grande comunidade ativa, desse modo é o sistema mais utilizado no mundo para a construção e análise de banco de dados grafo.

Cypher é a linguagem utilizada pelo Neo4j para realizar as consultas sobre os grafos armazenados. A linguagem Cypher é similar à utilizada pelos bancos de dados do tipo SQL. Cada um dos elementos que constituem um grafo (nós,

relacionamentos, atributos, propriedades etc.) possuem uma representação específica para a linguagem Cypher.

A Figura 83 apresenta a representação de um nó para a linguagem Cypher. Um nó, utilizando a linguagem Cypher, é representado através de parênteses (). Para adicionar um nome ao nó, basta colocar um nome dentro desses parênteses (ex.: (carro)). Por meio da Figura 83, é possível perceber que também podem ser adicionados atributos em cada um dos nós criados.

O que difere a linguagem Cypher de várias outras utilizadas para realizar consultas em uma base de dados, é a capacidade de definir e identificar padrões (*Patterns*). Por exemplo, seja a sentença a ser pesquisada: “Uma Pessoa Vive na Cidade” ou “A Cidade é Parte de Um País”, pode ser pesquisada da seguinte forma:

(:Pessoa) - [:Vive_em] -> (:Cidade) - [:Parte_de] -> (:País)

Essa sentença possui dois relacionamentos e três nós diferentes. Desse modo, é possível criar consultas complexas sobre os grafos criados.

Figura 83 – Exemplos de construção de um nó utilizando a Cypher.

```
()
(matrix)
(:Movie)
(matrix:Movie)
(matrix:Movie {title: "The Matrix"})
(matrix:Movie {title: "The Matrix", released: 1997})
```

Fonte: adaptado de NEO4J (2020).

Os relacionamentos também podem ser definidos de diferentes formas. A Figura 84 apresenta algumas das formas presentes na linguagem Cypher. Por meio dessa figura, verifica-se que a forma mais simples de definir um relacionamento direcional é através da seta (->). Essa seta representa o caminho existente entre dois nós ou o próprio nó. Também é possível definir um relacionamento unidirecional através de “traços” (--). Os colchetes ([..]) são empregados para dar um maior nível de detalhamento para o relacionamento, como labels, propriedades e atributos.

Figura 84 – Exemplos de construção de relacionamentos utilizando a Cypher.

```
-->
-[role]->
-[:ACTED_IN]->
-[role:ACTED_IN]->
-[role:ACTED_IN {roles: ["Neo"]}]->
```

Fonte: adaptado de NEO4J (2020).

Quando os nós e relacionamentos são combinados, é possível criar os relacionamentos existentes entre cada um dos nós ou grupos de nós. Por exemplo, para indicar que o ator Keanu Reeves atuou como Neo no filme Matrix, pode ser criado o padrão indicado na Figura 85.

Figura 85 – Exemplo de construção de padrões utilizando a Cypher.

```
(keanu:Person:Actor {name: "Keanu Reeves"})
-[role:ACTED_IN {roles: ["Neo"]} ]->
(matrix:Movie {title: "The Matrix"})
```

Fonte: adaptado de NEO4J (2020).

Para evitar repetições constantes de cada um dos padrões estabelecidos, através da linguagem Cypher, é possível criar variáveis para definir os padrões de consulta. A Figura 86 mostra como pode ser definida a consulta presente na Figura 85 através de uma variável.

Figura 86 – Exemplo de definição de padrões utilizando a Cypher.

```
acted_in = (:Person)-[:ACTED_IN]->(:Movie)
```

Fonte: adaptado de NEO4J (2020).

Criando um banco de dados grafo

Para criar um banco de dados grafo é necessário, inicialmente, adicionar os nós. Na linguagem Cypher, a adição de um nó (*Movie*) ao grafo ocorre como mostrado na Figura 87. Após adicionar esse nó, a Cypher retorna uma tabela contendo o número de mudanças ocasionadas pela inserção dessa nova entidade ao banco de dados. Além disso, nessa mesma figura, é possível verificar como o nó é adicionado ao grafo.

Figura 87 – Exemplo de adição de um nó utilizando a Cypher.

```
CREATE (:Movie { title:"The Matrix",released:1997 })
```

```
+-----+
| No data returned. |
+-----+
Nodes created: 1
Properties set: 2
Labels added: 1
```

Movie
title = 'The Matrix' released = 1997

Fonte: adaptado de NEO4J (2020).

Para retornar o nó (*Movie*) logo após a adição ao grafo, basta informar a palavra-chave “RETURN” após a definição desse novo nó. A Figura 87 mostra esse procedimento e o resultado obtido após a adição do nó “p”, que corresponde ao ator Keanu Reeves.

Figura 88 – Retorno do nó adicionado.

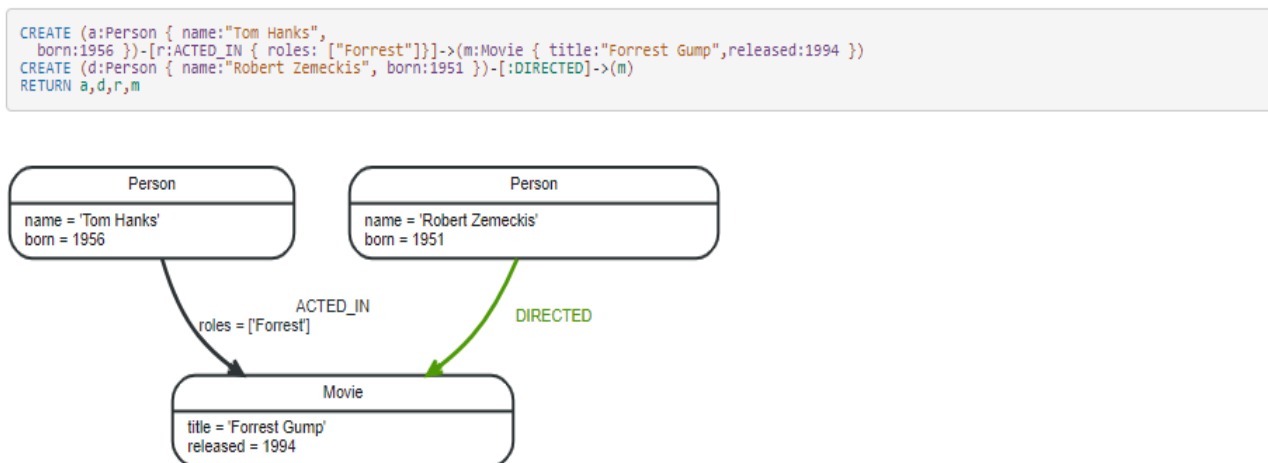
```
CREATE (p:Person { name:"Keanu Reeves", born:1964 })
RETURN p
```

```
+-----+
| p |
+-----+
| Node[1]{name:"Keanu Reeves",born:1964} |
+-----+
1 row
Nodes created: 1
Properties set: 2
Labels added: 1
```

Fonte: adaptado de NEO4J (2020).

Para criar diferentes nós, relacionamentos ou padrões, podem ser utilizados várias palavras-chave “CREATE” e, caso seja necessário retornar mais desses novos elementos, pode ser utilizada a palavra “RETURN”, seguida pelos valores desejados separados por vírgula (,). A Figura 89 apresenta um exemplo de adição e o grafo retornado após a adição desses elementos.

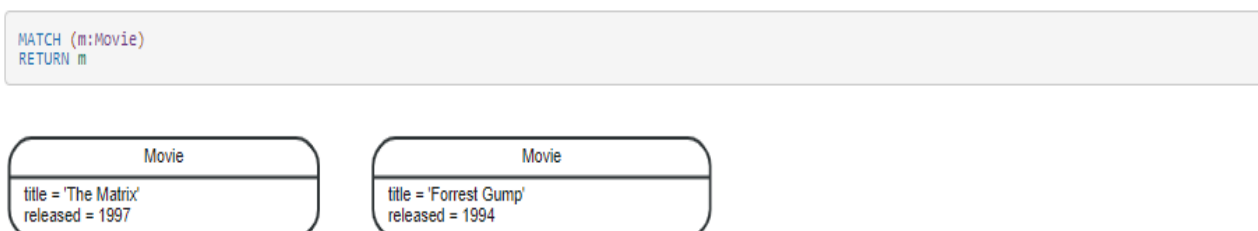
Figura 89 – Adição e retorno de vários elementos.



Fonte: adaptado de NEO4J (2020).

Para realizar consultas é utilizada a palavra-chave “MATCH”. Através dessa palavra podem ser realizadas consultas sobre determinados padrões presentes no grafo. Por exemplo, utilizando o comando “MATCH” para retornar todos os filmes existentes no grafo criado anteriormente, será encontrado o resultado presente na Figura 90.

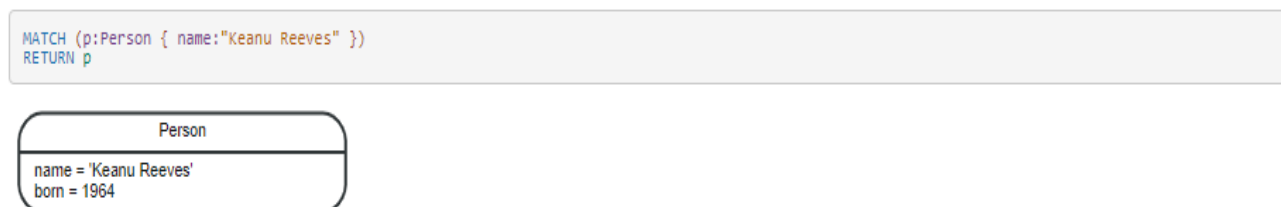
Figura 90 – Consulta dos filmes adicionados ao grafo.



Fonte: adaptado de NEO4J (2020).

Além disso, é possível pesquisar por nó em especial. Por exemplo, caso seja necessário retornar o nó Pessoa Keanu Reeves, pode-se ser utilizado o atributo desse nó. A Figura 91 mostra esse procedimento.

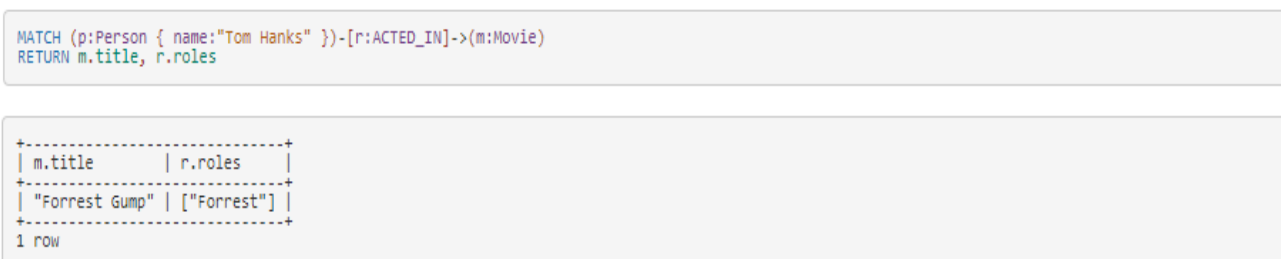
Figura 91 – Consulta por um nó específico.



Fonte: adaptado de NEO4J (2020).

Caso seja necessário executar uma consulta em todo o grafo para encontrar os nós que possuem um mesmo padrão de comportamento, também pode ser empregado o comando “MATCH”. Um exemplo para essa consulta através de um padrão é apresentado na Figura 92. Como retratado nessa figura, deseja-se obter os filmes (título e papéis) que a pessoa Tom Hanks atuou.

Figura 92 – Consulta através de um padrão.

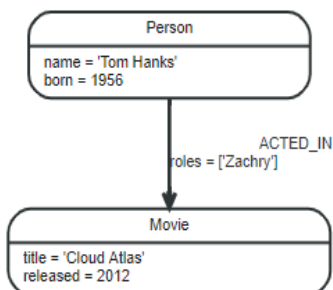


Fonte: adaptado de NEO4J (2020).

Para adicionar novas informação ao grafo já criado, é necessário primeiramente utilizar a palavra “MATCH”, para encontrar qual é o nó, que alvo e, posteriormente, utilizar a palavra “CREATE” a fim de adicionar a informação desejada. A Figura 93 mostra como é possível adicionar o filme “Cloud Atlas” e o papel “Zachry”, para o ator Tom Hanks.

Figura 93 – Adicionando e “linkando” com dados já existentes.

```
MATCH (p:Person { name:"Tom Hanks" })
CREATE (m:Movie { title:"Cloud Atlas",released:2012 })
CREATE (p)-[r:ACTED_IN { roles: ['Zachry']}]>(m)
RETURN p,r,m
```



Fonte: adaptado de NEO4J (2020).

Para completar padrões (patterns) já existentes é utilizada a palavra-chave “MERGE”. Ao utilizar o “MERGE”, internamente, a Cypher pesquisa se esse determinado atributo já está presente no grafo e, caso não esteja, ele é adicionado. A função do “MERGE” é evitar a duplicidade de elementos. Por exemplo, adicionar o ano de lançamento do filme Cloud Atlas ao grafo já existente pode ser feito através dos comandos presentes na Figura 94.

Figura 94 – Exemplo de utilização da função MERGE.

```
MERGE (m:Movie { title:"Cloud Atlas" })
ON CREATE SET m.released = 2012
RETURN m
```

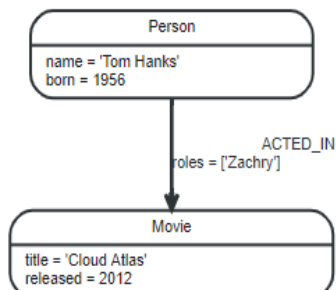
```
+-----+
| m      |
+-----+
| Node[5]{title:"Cloud Atlas",released:2012} |
+-----+
1 row
```

Fonte: adaptado de NEO4J (2020).

A função “MERGE” também pode ser utilizada para adicionar novos padrões de comportamento para as entidades presentes no grafo. A Figura 95 mostra como o relacionamento presente na Figura 93 pode ser criado utilizando a palavra-chave “MERGE”.

Figura 95 – Utilizando a função MERGE para adicionar novo padrão ao grafo.

```
MATCH (m:Movie { title:"Cloud Atlas" })
MATCH (p:Person { name:"Tom Hanks" })
MERGE (p)-[r:ACTED_IN]->(m)
ON CREATE SET r.roles = ['Zachry']
RETURN p,r,m
```

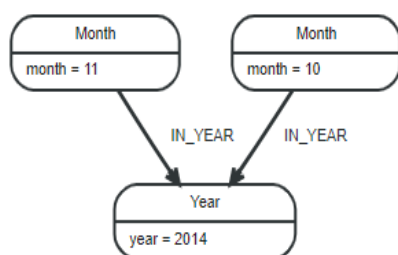


Fonte: adaptado de NEO4J (2020).

Para criar estruturas no formato de “árvore”, a função MERGE também pode ser utilizada. A Figura 96 apresenta um exemplo de construção para os nós “Ano” e “Mês”.

Figura 96 – Construção de “árvores” utilizando a função MERGE.

```
CREATE (y:Year { year:2014 })
MERGE (y)-[:IN_YEAR]->(m10:Month { month:10 })
MERGE (y)-[:IN_YEAR]->(m11:Month { month:11 })
RETURN y,m10,m11
```



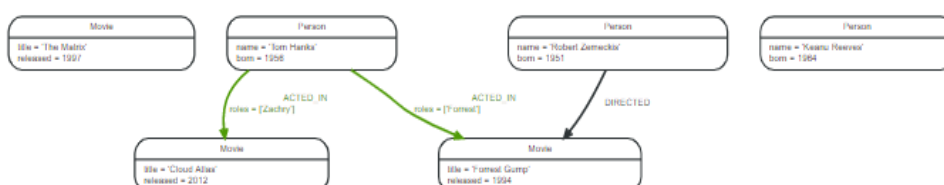
Fonte: adaptado de NEO4J (2020).

Realizando consultas sobre o banco de dados

Para a realização de todas as consultas, considere o grafo criado e apresentado na Figura 97. Nessa figura é possível ver a construção de um grafo com 6 nós (3 pessoas e 3 filmes) e outros 3 relacionamentos.

Figura 97 – Construção do grafo utilizado para consultas.

```
CREATE (matrix:Movie { title:"The Matrix",released:1997 })
CREATE (cloudAtlas:Movie { title:"Cloud Atlas",released:2012 })
CREATE (forrestGump:Movie { title:"Forrest Gump",released:1994 })
CREATE (keanu:Person { name:"Keanu Reeves", born:1964 })
CREATE (robert:Person { name:"Robert Zemeckis", born:1951 })
CREATE (tom:Person { name:"Tom Hanks", born:1956 })
CREATE (tom)-[:ACTED_IN { roles: ["Forrest"]}]>-(forrestGump)
CREATE (tom)-[:ACTED_IN { roles: ["Zachry"]}]>-(cloudAtlas)
CREATE (robert)-[:DIRECTED]>-(forrestGump)
```



Fonte: adaptado de NEO4J (2020).

A primeira palavra-chave utilizada para consulta é a “WHERE”. O funcionamento do “WHERE” para a linguagem Cypher é o mesmo que o encontrado nos bancos SQL. “WHERE” é utilizado como um filtro para as consultas. Desse modo, para consultar quais são as informações armazenadas no nó correspondente ao filme “The Matrix”, pode-se realizar a consulta presente na Figura 98. A função “WHERE” também pode ser utilizada junto com as expressões “OR”, “AND” e “NOT”.

Figura 98 – Aplicando filtro através da função “WHERE”.

```
MATCH (m:Movie)
WHERE m.title = "The Matrix"
RETURN m
```

```
+-----+
| m      |
+-----+
| (:Movie {title: "The Matrix", released: 1997}) |
+-----+
1 row
```

Fonte: adaptado de NEO4J (2020).

Para a aplicação do filtro através da função “WHERE”, também podem ser utilizados valores numéricos, expressões regulares e comparações. A Figura 99 apresenta um exemplo de utilização de consultas utilizando esses elementos.

Figura 99 – Aplicando um filtro com diferentes expressões.

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name =~ "K.+" OR m.released > 2000 OR "Neo" IN r.roles
RETURN p,r,m
```

p	r	m
(:Person {name: "Tom Hanks", born: 1956})	[:ACTED_IN {roles: ["Zachry"]}]	(:Movie {title: "Cloud Atlas", released: 2012})

1 row

Fonte: adaptado de NEO4J (2020).

Expressões mais complexas também podem ser construídas através da utilização da função “WHERE”. Por exemplo, podem-se utilizar pesquisas através de padrões, como mostrado na Figura 100.

Figura 100 – Aplicando um filtro com definição de padrões.

```
MATCH (p:Person)-[:ACTED_IN]->(m)
WHERE NOT (p)-[:DIRECTED]->()
RETURN p,m
```

p	m
(:Person {name: "Tom Hanks", born: 1956})	(:Movie {title: "Cloud Atlas", released: 2012})
(:Person {name: "Tom Hanks", born: 1956})	(:Movie {title: "Forrest Gump", released: 1994})

2 rows

Fonte: adaptado de NEO4J (2020).

A função “RETURN”, mostrada anteriormente, pode ser utilizada para retornar, além dos nós, um grande conjunto de expressões. Expressões em Cypher podem ser nomes, números, strings, array, propriedades específicas, relacionamentos e até funções de avaliação como *length(array)*, *toInteger("12")* etc. Essas expressões, para retorno da consulta, são utilizadas como nome da coluna. Para facilitar o reconhecimento da coluna, normalmente é utilizado o conector “AS”

para alterar o nome da expressão. A Figura 101 apresenta alguns exemplos de expressões que podem ser utilizadas para a realização de consultas.

Figura 101 – Aplicando os retornos através de expressões.

```
MATCH (p:Person)
RETURN p, p.name AS name, toUpper(p.name), coalesce(p.nickname,"n/a") AS nickname,
{ name: p.name, label:head(labels(p))} AS person
```

p	name	toUpper(p.name)	nickname	person
(:Person {name: "Keanu Reeves", born: 1964})	"Keanu Reeves"	"KEANU REEVES"	"n/a"	{name: "Keanu Reeves", label: "Person"}
(:Person {name: "Robert Zemeckis", born: 1951})	"Robert Zemeckis"	"ROBERT ZEMECKIS"	"n/a"	{name: "Robert Zemeckis", label: "Person"}
(:Person {name: "Tom Hanks", born: 1956})	"Tom Hanks"	"TOM HANKS"	"n/a"	{name: "Tom Hanks", label: "Person"}

3 rows

Fonte: adaptado de NEO4J (2020).

Para retornar valores únicos em uma consulta pode ser utilizada a função “DISTINCT”. A Figura 102 apresenta como pode ser realizada a consulta para o retorno de valores distintos.

Figura 102 – Retornando valores distintos.

```
MATCH (n)
RETURN DISTINCT labels(n) AS Labels
```

Labels
["Movie"]
["Person"]

2 rows

Fonte: adaptado de NEO4J (2020).

Funções de agregação também podem ser utilizadas em conjunto com o “RETURN”. Funções como *count()*, *sum()*, *avg()*, *min()* e *max()*. Por exemplo, a Figura 103 apresenta o resultado da consulta que retorna o número de nós do tipo “Pessoa” existentes no grafo.

Figura 103 – Número de nós existentes.

```
MATCH (:Person)
RETURN count(*) AS people
```

```
+-----+
| people |
+-----+
| 3      |
+-----+
1 row
```

Fonte: adaptado de NEO4J (2020).

Outro exemplo de consulta consiste em retornar o número de vezes que um ator trabalhou em parceria com um diretor. A Figura 104 mostra o exemplo dessa consulta e o resultado obtido.

Figura 104 – Número de parcerias entre ator e diretor.

```
MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)<-[:DIRECTED]-(director:Person)
RETURN actor, director, count(*) AS collaborations
```

```
+-----+-----+-----+
| actor                                | director                                | collaborations |
+-----+-----+-----+
| (:Person {name: "Tom Hanks", born: 1956}) | (:Person {name: "Robert Zemeckis", born: 1951}) | 1             |
+-----+-----+-----+
```

1 row

Fonte: adaptado de NEO4J (2020).

Através da linguagem Cypher também é possível ordenar os resultados obtidos utilizando a função “ORDER BY”. Quando essa função é seguida da expressão “ASC”, os valores são ordenados de forma crescente. Para a ordenação em ordem decrescente, pode ser empregada a expressão “DESC”. A função “ORDER BY” também permite indicar o limite de resultados a serem exibidos. Para isso, basta adicionar a função “LIMIT” e informar o número máximo de resultados a serem exibidos. A Figura 105 mostra um exemplo desse tipo de consulta.

Figura 105 – Consultas com resultados ordenados.

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
RETURN a, count(*) AS appearances
ORDER BY appearances DESC LIMIT 10;
```

a	appearances
(:Person {name: "Tom Hanks", born: 1956})	2

1 row

Fonte: adaptado de NEO4J (2020).

A função *collect()* também pode ser utilizada na linguagem Cypher. Essa função retorna os resultados como uma lista. Essa função é útil para preencher valores retornados em uma consulta. A Figura 106 mostra um exemplo de retorno utilizando essa função.

Figura 106 – Exemplo de utilização da função collect ().

```
MATCH (m:Movie)<-[:ACTED_IN]-(a:Person)
RETURN m.title AS movie, collect(a.name) AS cast, count(*) AS actors
```

movie	cast	actors
"Forrest Gump"	["Tom Hanks"]	1
"Cloud Atlas"	["Tom Hanks"]	1

2 rows

Fonte: adaptado de NEO4J (2020).

Para combinar diferentes resultados em apenas uma consulta, pode ser utilizada a função "UNION". Por meio dessa função é possível criar apenas uma estrutura que retorne os atores e diretores presentes nos filmes. A Figura 107 apresenta a consulta e o resultado obtido.

Figura 107 – Exemplo de utilização da função UNION.

```
MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)
RETURN actor.name AS name, type(r) AS type, movie.title AS title
UNION
MATCH (director:Person)-[:DIRECTED]->(movie:Movie)
RETURN director.name AS name, type(r) AS type, movie.title AS title
```

name	type	title
"Tom Hanks"	"ACTED_IN"	"Cloud Atlas"
"Tom Hanks"	"ACTED_IN"	"Forrest Gump"
"Robert Zemeckis"	"DIRECTED"	"Forrest Gump"

3 rows

Fonte: adaptado de NEO4J (2020).

Na linguagem Cypher, para combinar diferentes partes de uma expressão, é necessário utilizar a palavra-chave “WITH”. Por meio da utilização do “WITH” é possível combinar partes individuais de uma consulta e gerar o resultado final. Na Figura 108 está presente a parte da consulta que retorna os filmes que tem atores presentes no grafo e combina com o filtro que seleciona apenas aqueles que atuaram em mais de um filme.

Figura 108 – Exemplo consulta utilizando o WITH.

```
MATCH (person:Person)-[:ACTED_IN]->(m:Movie)
WITH person, count(*) AS appearances, collect(m.title) AS movies
WHERE appearances > 1
RETURN person.name, appearances, movies
```

person.name	appearances	movies
"Tom Hanks"	2	["Cloud Atlas", "Forrest Gump"]

1 row

Fonte: adaptado de NEO4J (2020).

Os comandos presentes neste capítulo representam apenas os comandos básicos para a construção e consulta de grafos utilizando o Neo4j e a Cypher. Nas aulas gravadas são mostradas algumas aplicações que utilizam essas ferramentas para gerar conhecimento a partir da análise dos grafos.

Referências

BAHGA, A.; MADISETTI, V. *Big Data Science & Analytics: A Hands-On Approach*. 1. ed. VPT, 2016.

FORTUNATO, Santo. *Community detection in graphs*. *Physics reports*, v. 486, n. 3-5, p. 75-174, 2010.

NEEDHAM, M. HODLER, A. E. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. 1. ed. O'Reilly Media, 2019.

NEO4J. *Introduction to Cypher*. Disponível em: <<https://neo4j.com/docs/getting-started/current/cypher-intro/>>. Acesso em: 23 dez. 2020.

OKAMOTO, K.; CHEN, W.; LI, Xiang-Yang. Ranking of closeness centrality for large-scale social networks. In: *International Workshop on Frontiers in Algorithmics*. Berlin, Springer: Heidelberg. p. 186-195. 2008.

PEIXOTO, J.; FILHO, N.; CASTELLANI, T. *Variação Granulométrica das Praias Arenosas da Costa Leste da Ilha de Santa Catarina*. Brasil, Santa Catarina: Gravel. v.10. n. 1. p. 13-21. 2012.