**LINGUAGEM DE PROGRAMAÇÃO**

AULA 2

Prof. Wellington Rodrigo Monteiro

**CONVERSA INICIAL**

Olá! Anteriormente, comentamos sobre o uso das bibliotecas e, ainda, a importância de lermos a documentação das bibliotecas. Aqui, vale reforçar: o objetivo é o de mostrar *o***caminho**para você, mas é impossível mostrar tudo que existe pela velocidade de mudança. Apostamos que, nas próximas 24h, pelo menos algumas **centenas**de bibliotecas em Python mudarão alguma coisa. Elas poderão ter novas funcionalidades, mudanças em funcionalidades já existentes, correções na documentação, novos bugs, bugs resolvidos, entre outros. Por outro lado, a**base** não muda: a forma de ler, procurar e interpretar a documentação que aprendemos aqui lhe ajudarão independentemente das possíveis mudanças no futuro.

Bom, e por que relembramos isso agora? Uma das áreas da ciência em maior crescimento e velocidade é a da Inteligência Artificial. E, naturalmente, existem bibliotecas em Python para Inteligência Artificial (IA) – de fato, o Python já é entendido atualmente como a linguagem padrão para o desenvolvimento de algoritmos de IA. Logo, que tal criarmos uma IA juntos?

**TEMA 1 – DEFININDO INTELIGÊNCIA ARTIFICIAL E APRENDIZAGEM DE MÁQUINA**

Quando nos aprofundamos um pouco mais no mundo de IA é comum nos confundirmos com vários termos como: *inteligência artificial, machine learning, aprendizagem de máquina, deep learning, ciência de dados, advanced analytics, redes neurais,*entre outros. Agora, qual é a diferença entre eles? O que é o quê? Vamos pensar nesses exemplos utilizando analogias.

1. Imagine uma **praça de alimentação** de um shopping: existem vários locais para comer e atendendo diversos gostos, como pizzas, massas, comidas típicas e vegetarianas, sorvetes, sanduíches e outros.
2. Agora, imagine que dentro dessa praça de alimentação existe um **restaurante** muito famoso. Pelo fato de ser um restaurante, você pode imaginar que diversos pratos são servidos, e para atender a todos os gostos.
3. Ainda que esse restaurante sirva vários pratos ele é mais conhecido pelos seus **pratos italianos**.
4. E, na cidade, “pratos italianos” é praticamente um sinônimo para **massas**.
5. Quando falamos em massas, uma **macarronada**bem-feita já é digna de muitos elogios e satisfaz o desejo de quase todos os clientes.
6. Por outro lado, quase sempre a **lasanha**vem à mente também. O problema, contudo, é que ela exige mais trabalho para fazer (e limpar) em comparação com a macarronada.
7. Só que existem vários tipos de lasanhas. Lasanhas mais complexas podem **ter mais camadas** de massa e recheio, e existem recheios mais complexos do que outros.



Crédito: Artisticco/ Shutterstock.

Dito isso, vamos explicar as analogias. A praça de alimentação é análoga ao universo de **Advanced Analytics**, uma área que engloba não somente IA, mas também a visualização de dados. Já ouviu falar em *Business Intelligence*, ou BI? Dentro das empresas, BI é praticamente sinônimo de relatórios e painéis visuais (*dashboards*) contendo vários gráficos mostrando o desempenho de indicadores importantes para a área de negócio. Esses indicadores podem ser, por exemplo, o histórico de vendas; a quantidade de problemas nas fábricas; o volume de produção registrado até o momento; entre outros. Note que, nesses exemplos, quase sempre falamos **do passado e do presente**. Esses dashboards, por outro lado, também podem ser combinados com outros algoritmos de IA e outras técnicas para prever o futuro. É como uma praça de alimentação de um shopping no qual podemos comprar as refeições de mais de um lugar e comer em um lugar só.

Já o restaurante é análogo ao mundo da **Ciência de Dados**. Veja que Ciência de Dados fica **dentro**do mundo de Advanced Analytics. Quando trabalhamos com Ciência de Dados não trabalhamos **somente**com IA, mas também usamos a Estatística e os nossos próprios conhecimentos em TI.

Os pratos italianos são análogos à **IA**: é um tópico interessante e conhecido no mundo todo. É também amplo e cheio de possibilidades. Para alguns, pode parecer simples. Para outros, pode parecer algo bem refinado e que exige um bom conhecimento para ser feito com qualidade.

Por outro lado, se você perguntar a uma pessoa qual é a primeira coisa que vem à cabeça dela é provável que tenha respostas bem diferentes. Para alguém novo, isso pode ser sinônimo de lasanha. Para uma pessoa com mais experiência de vida, por outro lado, pode ser sinônimo de um tiramisu artesanal. E, para você, pode ser um gelato. Observe que diferentes públicos podem possuir interpretações completamente diferentes sobre o que é um prato italiano. E, ainda, diferentes públicos possuem diferentes **expectativas**sobre o que seria um bom prato italiano. A mesma coisa é com IA: pessoas que não são da área podem possuir como expectativa um algoritmo superpoderoso e que aprende em um toque de mágica como resolver problemas. Imaginam uma IA próxima daquelas superinteligências que vemos nos filmes – como um assistente ou um robô que conversasse conosco e que resolvesse qualquer problema de qualquer área. Podem, até mesmo, imaginar algo próximo ao J.A.R.V.I.S. do Homem de Ferro (ou, por outro lado, a Skynet do Exterminador do Futuro). Ou, em outro extremo, podem entender que a IA hoje em dia é somente um termo novo para algo mais antigo: Estatística (e/ou Matemática).

Já as massas são análogas ao **machine learning**(ML ou **aprendizagem de máquina**). Apesar dos exemplos anteriores (lasanha, tiramisu e gelato), se entrevistássemos hoje um grande conjunto de pessoas e perguntássemos a eles por um exemplo de “prato italiano servido em restaurante”, certamente a grande maioria das respostas seria massas. E aí vai uma informação importante: **hoje, no contexto da maioria das empresas, ML e IA são sinônimos**.

O que se entende por IA foi mudando ao longo do tempo: até os anos 1990 IA poderia ser sinônimo de força bruta (o Deep Blue, famoso na década de 1990 por vencer o campeão mundial de xadrez Garry Kasparov, usava força bruta para ganhar: isto é, ele avaliava milhões de combinações futuras de jogadas e escolhia a melhor) e, ainda, de **sistemas especialistas** contendo várias regras construídas à mão: um sistema responsável pela segurança dos equipamentos e maquinário de uma usina hidrelétrica entra nessa regra, por exemplo. Por outro lado, esses algoritmos podem ser complexos para serem construídos e geridos – principalmente com o exponencial aumento de regras e necessidade de ajustes nessas regras ao longo do tempo. Já o que veio nos anos 2000 é o ML: algoritmos capazes de reconhecer padrões e aprender, de forma generalizada, a partir de uma base de dados histórica. É esse o tipo de algoritmo utilizado atualmente e entendido como “IA”. Já as superinteligências dos filmes (ainda) não são realidade, mas podem ser em um futuro próximo. Note que Ciência de Dados inclui IA, mas não é somente isto: também inclui Estatística, conhecimento sobre a área na qual se aplicaria o problema (como Vendas, Logística, RH, Jurídico e outros), e de computação.

**1.1 *MACHINE LEARNING***

Continuando: da mesma forma que a macarronada é um tipo de massa, existem técnicas e bibliotecas mais simples em ML, como o **scikit-learn**. Essa é uma das bibliotecas (lembra da nossa última conversa sobre bibliotecas?) mais conhecidas em Python e inclui técnicas simples como uma árvore de decisão ou um algoritmo de agrupamento de dados.

Por outro lado, a lasanha é outro tipo de massa mais complexa. Da mesma forma, também existem algoritmos mais complexos como as **redes neurais** – algoritmos que tiveram suas origens em trabalhos que tentaram traduzir a estrutura dos neurônios de um cérebro em algoritmos e circuitos. Esse tipo de algoritmo possui várias funções de ativação (como neurônios) agrupados em **camadas**. Existem, ainda, diversas conexões entre essas camadas. Sabemos que esse tipo de algoritmo é tratado como se fosse o melhor de todos, mas aqui vale um ponto de atenção: entenda as redes neurais como um “canhão para matar uma formiga”: às vezes, o problema a ser resolvido é tão simples que não precisaria de uma rede neural. Por outro lado, se mesmo assim você quiser usar uma rede neural, você pode acabar tendo um resultado pior do que um algoritmo mais simples – em outras palavras, o canhão também pode “errar” a formiga.

Existem várias arquiteturas de redes neurais como o multi-layer perceptron (MLP), recurrent neural network (RNN), feed-forward (FF), entre outros. Na prática, existem dezenas de arquiteturas: servem tanto para trabalhar com imagens, textos, sons, prever valores, grupos, entre outros: da mesma forma que também existem diferentes tipos de lasanhas. E, ainda, existem lasanhas com várias e várias camadas – da mesma forma, existem redes neurais com várias e várias camadas: essas redes se chamam *deep learning* – ou aprendizagem profunda. *A escolha da melhor arquitetura, biblioteca ou tipo de algoritmo depende muito de cada caso. Existem cientistas de dados que possuem algoritmos e técnicas de “estimação”, enquanto outros cientistas gostam de testar diferentes técnicas para cada problema. A escolha da melhor técnica que se aplica a cada caso é um exemplo de trabalho de Ciência de Dados.*

**TEMA 2 – TRABALHANDO COM PROBLEMAS DE CIÊNCIA DE DADOS**

Um projeto de Ciência de Dados sempre utiliza alguns passos – o nome dos passos e a ordem podem mudar ligeiramente dependendo de cada metodologia. Por outro lado, a execução possui alguns passos típicos. São eles:

1. a definição do problema;
2. a definição dos dados;
3. a preparação dos dados;
4. o desenvolvimento dos modelos;
5. a avaliação dos modelos; e
6. a disponibilização dos modelos.

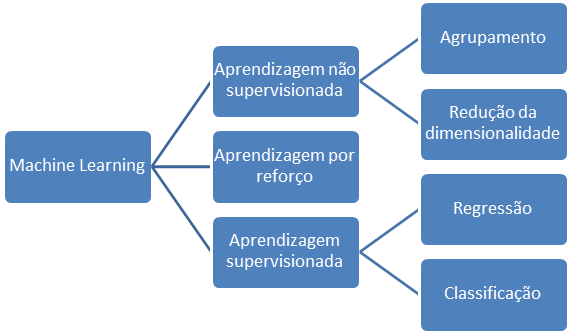
Vamos com calma por cada um desses itens. Primeiro, falaremos aqui sob um olhar de **projeto**, e depois sobre **execução**.

**2.1 DEFINIÇÃO DO PROBLEMA**

Esse passo trata, em sua essência, de responder: “o que você quer fazer com ML?”. Ou, ainda, em que ML pode te ajudar a resolver algum problema. Pensemos do ponto de vista de TI, e não sejamos vagos: “quero diminuir os gastos da empresa” é algo vago. Existem tantas formas de se diminuir os gastos e várias delas não envolvem ML – o consumo consciente de água por parte das pessoas já ajudaria isso, e nada teria a ver com ML.

Agora, um exemplo que poderia envolver ML é: “quero saber quais são os clientes que podem ser inadimplentes no futuro próximo”. Nesse caso, temos um exemplo específico do **escopo** do trabalho (os clientes), qual é a **dor** **da área de negócio** a ser endereçada (as inadimplências) e **o que se deseja** fazer (prever e apontar quem poderiam ser os futuros inadimplentes).

Ao definir o problema, poderemos saber qual é a melhor tipo de técnica a ser utilizada. ML é dividido em alguns subgrupos, conforme ilustrado pela figura a seguir.



A **aprendizagem supervisionada**seaplica quando sabemos **o que**queremos prever e em base **do quê**. Ela é dividida em dois principais tipos.

* **classificação**: quando queremos prever um “grupo”. Pode ser algo como cartão de crédito aprovado/cartão de crédito reprovado; sim/não; foto de cachorro/foto de gato; candidato pertencente ao grupo A/B/C.
* **regressão**: quando queremos prever um valor em uma “faixa numérica”. Pode ser algo como temperatura, cotação do Bitcoin, número de novos acessos no site, valor de venda da casa, nota do estudante.

Quando trabalhamos com problemas de aprendizagem supervisionada partimos da premissa de que precisamos de **uma base de dados**. Se queremos prever a temperatura de uma cidade para os próximos meses precisaremos de uma boa base histórica dela contendo, se possível, alguns anos de dados contendo informações que possuam relação com o que queremos prever (para pegarmos o comportamento dependendo da estação do ano; a relação entre temperatura, vento e chuva; entre outros); e, naturalmente, a coluna que queremos prever (no caso, a temperatura).

Caso queiramos prever se um futuro cliente terá ou não o seu pedido de cartão de crédito aprovado a partir do seu histórico, precisaremos ter também uma base de dados histórica com outros clientes, contendo dados que são relevantes à concessão (ou não) do cartão de crédito e incluindo também a coluna que queremos prever (no caso, se o cartão de crédito foi ou não aprovado). Ou, ainda, se queremos prever se uma nova imagem possui um gato ou cachorro dentro dela, precisaremos obrigatoriamente ter em mãos uma base de dados contendo várias outras fotos com cachorros ou gatos.

Já a **aprendizagem não supervisionada**trata majoritariamente da **exploração**de dados: às vezes, não sabemos exatamente **o que**queremos prever, mas precisamos manipular a base de dados, como a seguir.

* **redução da dimensionalidade**: quando precisamos selecionar quais dados realmente importam/são relevantes. Peguemos uma base de dados da meteorologia com 100 colunas diferentes: será que todaselas são importantes? Quando temos um número muito alto de dados, podemos cair naquilo que se chama de *maldição da dimensionalidade*: como várias técnicas de aprendizagem supervisionada em ML tratam de reconhecer padrões, fica bem difícil de detectar padrões com tantas colunas a serem analisadas. Logo, às vezes, é preferível selecionarmos somente aquelas que são realmente úteis para o nosso problema.
* **agrupamento** (também mencionado, às vezes, como *clusterização*): peguemos as mesmas 100 colunas que comentamos anteriormente – será que não conseguiríamos de alguma forma agrupá-las de acordo com a sua similaridade em novas colunas de um jeito que não percamos muita informação ao fazermos isso?

Por fim, a **aprendizagem por reforço** inclui técnicas que são orientadas a resolver problemas a partir de **recompensas**. Incluem-se aqui algoritmos que buscam otimizartrabalhos complexos como a distribuição de rotas logísticas ou de escalas de trabalho tentando descobrir quais pequenos ajustes levam a um melhor resultado ou, ainda, certos algoritmos usados em jogos de computador e de console.

É possível dizer que, no contexto do mercado de trabalho, grande maioria dos trabalhos de Ciência de Dados se concentram em problemas de aprendizagem supervisionada e não supervisionada.

**2.2 DEFINIÇÃO DOS DADOS**

Um bom projeto de ciência de dados costuma ter também o apoio de quem utilizará essa solução e que entende do problema. Nesse caso, quem entende do problema são as pessoas que lidam com as inadimplências hoje. É imprescindível que elas caminhem junto para resolver o problema. Pensemos aqui: quais fatores impactam? A visão do cientista de dados e do analista da área de negócio que o apoiará são igualmente importantes:

* o **analista da área de negócio**dará várias informações que fazem parte já da **experiência**dele: ele provavelmente pode citar itens como o período do ano (já que podem ter épocas do ano em que a inadimplência é maior); o score de crédito; a data de fundação do comércio; entre outros. Essas informações são valiosas para você por dois motivos.
  + ganho de tempo: você “aproveita” a experiência dessa pessoa ao descobrir com antecedência quais informações **podem ser**importantes. Às vezes, a intuição da pessoa pode estar errada em alguns pontos, mas estará certa em outros: logo, sempre haverá algo bom a se aproveitar.
  + confiança: é o analista (e/ou a equipe na qual essa pessoa faz parte) quem fará uso do algoritmo. Como quase nada na vida é 100% previsível, é improvável que o algoritmo acerte 100% das vezes. Por outro lado, se essa pessoa nunca participou no processo de construção do algoritmo e não sabe quais fatores foram levados em conta, no primeiro momento em que o algoritmo der um resultado diferente do esperado haverá uma desconfiança. E, como você deve imaginar, quando alguém não confia em algo é bem difícil de mudarmos a sua opinião.
* o **cientista de dados** ajudará a trazer um conhecimento fora do contexto do dia a dia do analista: é parte do trabalho do cientista de dados pesquisar em bases científicas quais foram os principais achados sobre um determinado problema: no caso da inadimplência, existem vários estudos apontando quais fatores culminam na inadimplência – pode ser que alguns desses fatores não sejam do conhecimento do analista e poderão ser úteis para o seu trabalho. Logo, um trabalho de ciência de dados vai além de criar somente um algoritmo – é também o de ajudar os humanos em seu trabalho. Além disso, ele poderá trazer a sua experiência em projetos anteriores para fazer perguntas que possam ajudar no desenvolvimento do algoritmo: será que a tendênciado score nos últimos 12 meses é também relevante? Um score baixo pode ser uma coisa, mas e se o score estiver melhorando? Ou, ainda, e se o score é alto, mas está caindo vertiginosamente? Será que isso não é um sinal preocupante? Será que a inflação ao longo do tempo também não poderia ser relevante? Ou a cidade? Cidades menores e em regiões rurais podem ter uma dependência maior com a época da colheita de principais safras, ou cidades mais turísticas podem receber movimentar mais dinheiro nos finais de ano ou feriados estendidos. Veja que é um pensamento “fora da caixa” que pode ser bem-vindo durante o desenvolvimento do projeto.

Com o apoio dessas pessoas, é possível definir **a base de dados**a ser utilizada. Isso ocorre principalmente porque algoritmos de ML não “descobrem” sozinhos novos dados: é importante que um humano informe ao algoritmo que conjunto de dados deve ser utilizado. Esse conjunto pode vir de um banco de dados da própria empresa ou, ainda, de fontes de dados externas como sites governamentais, empresas de consultoria especializadas ou sites de acesso público. A preocupação aqui é a de garantir que a base de dados faça sentido no futuro: ou seja, que todas as informações a serem usadas estejam disponíveis também ao testarmos novos dados futuros.

Peguemos novamente o caso da inadimplência: não adianta treinar um algoritmo para prever casos futuros de inadimplência usando como apoio uma base de dados governamental que foi desativada no mês passado. Novos casos jamais poderão utilizar essa mesma base e, assim, você tem um algoritmo que não funcionará corretamente no futuro.

Algoritmos de ML são capazes de aprender sobre um conjunto de dados predefinido e, a partir dele, cria-se uma **generalização**para que possa funcionar bem com casos futuros. Imaginemos o próprio caso dos nossos estudos: você não quer só criar algoritmos com os conjuntos de dados trabalhados com o que nós aprendemos, não é? Na realidade, você quer aprender diferentes tipos e aplicações de algoritmos, saber o que funciona e o que não funciona para que você possa **aprender com os exemplos** e saber aplicar em **novos cenários**no futuro.

É como se você estivesse aprendendo a dirigir um carro na autoescola pela primeira vez: você não quer aprender a dirigir somente o Volkswagen Up 2016 1.0 branco, placa ABC-1234 da “Autoescola Centro” e somente no bairro “Ribeira”, mas você quer aprender diferentes situações de tempo, ruas, trânsito e outros carros para que você saiba o que fazer com outros carros, outros horários e outras ruas, não é? Por outro lado, para que você saiba o que fazer, você precisa ser exposto a diferentes cenários enquanto está aprendendo. Como você vai saber como estacionar o seu próprio carro se nunca teve contato com isso na vida? Como irá reagir se nunca você nem viu uma aula ou vídeo ensinando isso?

**2.3 PREPARAÇÃO DOS DADOS**

Com os algoritmos de ML é a mesma coisa: precisamos expor a eles as situações nas quais desejamos que ele aprenda para que ele saiba como reagir no futuro. Se vamos criar um algoritmo que vá prever a temperatura nas próximas horas é importante que forneçamos uma base para que o algoritmo aprenda. Agora, que base de dados é essa? Vamos imaginar uma planilha de Excel. Vamos pensar na sua própria cidade: qual é a granularidade? Queremos prever por hora? Por dia? Por semana? Por mês? Bom, comentamos anteriormente que seria por hora – logo, teremos **no mínimo** duas informações, a data com a hora em uma coluna e a temperatura em outra. Vamos usar os dados do Instituto Nacional de Meteorologia (INMET) para a cidade de Curitiba como base:

|  |  |
| --- | --- |
| **DataHora** | **Temperatura** |
| 2021-01-01 00:00 | 20.5 |
| 2021-01-01 01:00 | 19.9 |
| 2021-01-01 02:00 | 19.7 |
| 2021-01-01 03:00 | 19.6 |
| 2021-01-01 04:00 | 19.9 |
| 2021-01-01 05:00 | 18 |
| 2021-01-01 06:00 | 18 |
| 2021-01-01 07:00 | 17.7 |
| 2021-01-01 08:00 | 18.3 |
| 2021-01-01 09:00 | 18.9 |
| 2021-01-01 10:00 | 20.3 |
| 2021-01-01 11:00 | 21 |
| 2021-01-01 12:00 | 21 |
| 2021-01-01 13:00 | 20 |
| 2021-01-01 14:00 | 22.8 |

Veja que essa pequena amostra somente possui dados de algumas horas do dia 1º de janeiro de 2021. Vamos imaginar que queiramos prever a temperatura das 15h do dia 1º de janeiro de 2021 (em que, a partir de agora, sempre deixaremos no padrão *ano-mês-dia hora:minuto*, também convencionado nas documentações das bibliotecas como *yyyy-mm-dd hh:mm*– logo, essa data será escrita como 2021-01-01 15:00). Bom, nós, humanos, esperamos que a temperatura fique por volta dos 22 graus. Sabemos disso porque nas últimas horas a temperatura se manteve nessa faixa e porque é uma tarde de verão. Quer dizer, pode ser que abaixe um pouco (talvez uns 19 graus) ou aumente (talvez 25 graus), talvez? Veja que, para chegar a essa decisão, temos todo um conjunto de conhecimentos prévios: sabemos que a temperatura geralmente não muda muito hora a hora; sabemos que a temperatura à tarde geralmenteé maior do que a temperatura de madrugada; sabemos que uma mudança de 2 ou 3 graus é aceitável, mas 20 ou 30 graus de uma hora para outra não. Veja, também, que esse conhecimento depende de onde vivemos: se você cresceu em uma cidade na qual a temperatura não muda muito de uma hora para outra, você provavelmente chegará à decisão diferente de outra pessoa que está acostumada a grandes variações na temperatura (como é o próprio caso de Curitiba).

Agora, como um algoritmo vai aprender todos esses conceitos? Um algoritmo não sabe o que é “hora”, “sol” ou “Curitiba”. É como se tudo fosse grego para ele – algo incompreensível, algo assim:

|  |  |
| --- | --- |
| **ΗμερομηνίαΩρα** | **Θερμοκρασία** |
| ??? | 20.5 |
| ??? | 19.9 |
| ??? | 19.7 |
| ??? | 19.6 |
| ??? | 19.9 |
| ??? | 18 |
| ??? | 18 |
| ??? | 17.7 |
| ??? | 18.3 |
| ??? | 18.9 |
| ??? | 20.3 |
| ??? | 21 |
| ??? | 21 |
| ??? | 20 |
| ??? | 22.8 |

Mas, e aí? Como resolvemos isso? Uma alternativa é incluirmos as informações que são importantes para que o algoritmo tome uma decisão. O que influencia muitoa temperatura em Curitiba? Será que a quantidade de liquidificadores ligados ao mesmo tempo em São Paulo influencia? Ou seria a umidade, ou a incidência do sol (isto é, se o céu está limpo ou nublado, por exemplo)? O fato de estar chovendo ou não, talvez? Ou o vento? Ou a quantidade de potes de margarina vendidos no maior supermercado da cidade?

Veja que existem informações mais úteis do que outras, e algumas delas realmente mais atrapalham do que ajudam. Não sei se percebeu, mas às 14h a temperatura aumentou quase 3 graus. Se você fosse apostar a temperatura às 15h olhando somente o histórico da temperatura, o que você apostaria? Agora observe como fica a tabela quando incluímos a informação da radiação solar:

|  |  |  |
| --- | --- | --- |
| **DataHora** | **Temperatura** | **RadiacaoSolar** |
| 2021-01-01 00:00 | 20.5 |  |
| 2021-01-01 01:00 | 19.9 |  |
| 2021-01-01 02:00 | 19.7 |  |
| 2021-01-01 03:00 | 19.6 |  |
| 2021-01-01 04:00 | 19.9 |  |
| 2021-01-01 05:00 | 18 |  |
| 2021-01-01 06:00 | 18 |  |
| 2021-01-01 07:00 | 17.7 |  |
| 2021-01-01 08:00 | 18.3 | 33.3 |
| 2021-01-01 09:00 | 18.9 | 268.2 |
| 2021-01-01 10:00 | 20.3 | 583.9 |
| 2021-01-01 11:00 | 21 | 848.6 |
| 2021-01-01 12:00 | 21 | 572.5 |
| 2021-01-01 13:00 | 20 | 621.6 |
| 2021-01-01 14:00 | 22.8 | 2513.2 |

A radiação solar aumentou muito de 13h para 14h: aparentemente o céu ficou com menos nuvens nessa hora. E agora? Você mudaria o valor da sua “aposta” na temperatura às 15h? Veja o que realmente aconteceu:

|  |  |  |
| --- | --- | --- |
| **DataHora** | **Temperatura** | **RadiacaoSolar** |
| 2021-01-01 00:00 | 20.5 |  |
| 2021-01-01 01:00 | 19.9 |  |
| 2021-01-01 02:00 | 19.7 |  |
| 2021-01-01 03:00 | 19.6 |  |
| 2021-01-01 04:00 | 19.9 |  |
| 2021-01-01 05:00 | 18 |  |
| 2021-01-01 06:00 | 18 |  |
| 2021-01-01 07:00 | 17.7 |  |
| 2021-01-01 08:00 | 18.3 | 33.3 |
| 2021-01-01 09:00 | 18.9 | 268.2 |
| 2021-01-01 10:00 | 20.3 | 583.9 |
| 2021-01-01 11:00 | 21 | 848.6 |
| 2021-01-01 12:00 | 21 | 572.5 |
| 2021-01-01 13:00 | 20 | 621.6 |
| 2021-01-01 14:00 | 22.8 | 2513.2 |
| 2021-01-01 15:00 | 23.2 | 2337.6 |

A radiação solar diminuiu um pouco, mas permaneceu em um patamar alto. A temperatura aumentou um pouco também, pulando de 22.8 para 23.2 graus. É nesse sentido que um cientista de dados poderia trabalhar: entendendo e mapeando quais variáveis poderiam ajudar na tomada de decisão. Nesse caso, poderíamos incluir a umidade, vento e pressão atmosférica (mas não os liquidificadores ou os potes de margarina, que nada possuem relação com o nosso caso).

**2.4 DESENVOLVIMENTO DOS MODELOS**

O passo de gerar uma base de dados confiável (o que também inclui limpara base de dados de possíveis erros – se alguém errou um ponto a temperatura de algum dia pode ser apontada como “213” em vez de “21.3” graus) e o passo de desenvolver os modelos pode demandar até 80% de um projeto. É um trabalho que envolve muitos testes e, naturalmente, muito “vai e volta”.

Para os casos de **aprendizagem supervisionada**, retornemos novamente ao caso das inadimplências: após termos uma base de dados em mãos poderemos realizar o treinamento de um modelo preditivo. Para isso, é importante dividirmos a nossa base de dados original em pelo menos duas partes.

* A **base de treino** é uma amostra obtida a partir da base original (entre 60% e 80% da base na maioria das vezes) e é essa base que utilizamos para *treinar*um algoritmo – isto é, é ela que utilizamos para que um algoritmo reconheça e “aprenda” os padrões que estão dentro daquela base e, se possível, algumas exceções.
* A **base de testes** é o restante da base que não foi utilizada pela base de treinamento. Ela é usada para avaliarum algoritmo. Esse passo é importante porque representa como ele funcionaria na vida real e nos dá uma forma de compararum valor previsto com um valor que de fato aconteceu já que nós (humanos) sabemos o resultado que o algoritmo deveria prever, mas ele (o algoritmo) não. Afinal de contas, é fácil o algoritmo ter uma taxa de acerto alta olhando somente para a base de treino, mas não é nada representativo sobre como o algoritmo iria desempenhar no dia a dia: já a métrica de acerto sobre a base de testes possui uma representatividade bem melhor nesse sentido.

Ainda nesse passo, escolheremos o algoritmo a ser utilizado. Existem várias técnicas.

* aprendizagem supervisionada:
  + modelos lineares;
  + algoritmos estatísticos;
  + redes neurais;
  + árvores de decisão;
  + máquinas de vetor de suporte (*support vector machines*, ou SVMS);
  + florestas (múltiplas) de árvores de decisão; e
  + comitês de algoritmos.
* aprendizagem não supervisionada:
  + análise fatorial;
  + análise de componentes principais (*principal component analysis*, ou PCA);
  + K-médias (*k-means*); e
  + eliminação recursiva de atributos.

**2.5 AVALIAÇÃO DOS MODELOS**

Após treinar um modelo, é importante avaliarmos se ele está funcionando como esperado (ou não). Falar de “acurácia” é um termo complicado porque existem dezenas de métricas que possuem como objetivo explicar se um modelo está funcionando como esperado – logo, a forma de se calcular essa acuracidade muda radicalmente dependendo do caso.

Para saber se os resultados estão funcionando de acordo com o esperado, as métricas mais comuns são as que se segue.

* aprendizagem supervisionada
  + regressão:
    - erro médio absoluto (*mean absolute error*, ou MAE);
    - erro quadrático médio (*mean squared error*, ou MSE);
    - erro médio absoluto percentual (*mean absolute percentage error*, ou MAPE); e
    - raiz quadrada do erro quadrático médio (*root* *mean squared error*, ou RMSE).
  + classificação:
    - matriz de confusão;
    - precisão e revocação (*precision*e *recall*); e
    - curva ROC (*receiver operating characteristic*) e a área sob essa curva (*area under curve*, ou AUC).
* aprendizagem não supervisionada
  + testes estatísticos, como os testes de hipótese.

**2.6 DISPONIBILIZAÇÃO DOS MODELOS**

Após os resultados estarem aceitáveis (e a definição do “aceitável” depende muito de cada caso: às vezes, 75% de acerto para as inadimplências pode ser algo muito bom, mas os mesmos 75% para um algoritmo da área médica pode ser completamente inaceitável), o modelo pode ser disponibilizado para o uso diário.

Essa disponibilização (*deployment*) do modelo trata de deixá-lo disponível para um uso futuro. Isso inclui a sua integração com uma arquitetura técnica em uso por uma empresa e a forma que os dados serão alimentados ao algoritmo e seus resultados propriamente utilizados. É nesse ponto que se determina quem visualizará os valores e de qual forma.

**TEMA 3 – TRABALHANDO COM DADOS**

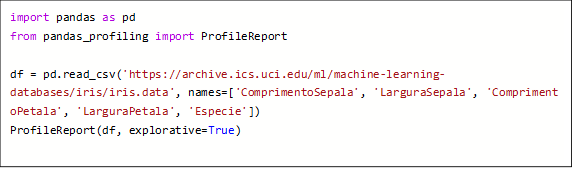
Agora que conversamos sobre o passo a passo de um projeto típico de Ciência de Dados, poderemos falar sobre o trabalho em si. Existem trabalhos nessa área que são executados em várias linguagens de programação, como Scala, R, Java e Python. Do ponto de vista corporativo, o Python recentemente se tornou a mais utilizada dentre essas.

As bases de dados (*datasets*) trabalhadas em ML possuem alguns tipos – os principais são os que se seguem.

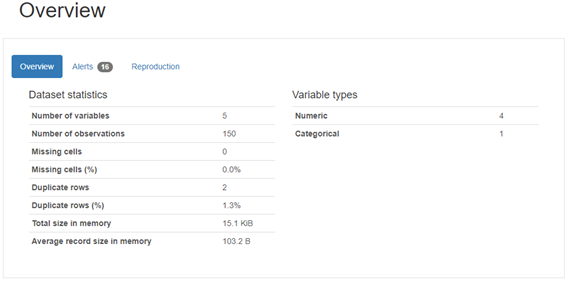
1. Dados **tabulares**: são aqueles que se parecem com planilhas de Excel, e são os mais comuns. São caracterizados por **atributos** (ou *features*): características (colunas) que fazem parte da base; e **instâncias**, isto é, indivíduos (linhas) que fazem parte da base de dados.
   1. Para problemas de aprendizagem supervisionada podemos ter um **label** ou **target** – um atributo que é aquele que queremos prever.
   2. Os dados podem ser numéricos (como valores referentes à temperatura, idade, preços e afins); categóricos (como aprovado/reprovado; as categorias da Carteira Nacional de Habilitação; tipos sanguíneos; entre outros); dados temporais (que possuem relação com data e hora, por exemplo); e textos (como esta descrição que você está lendo agora).
2. Bases de **imagens**: são fotos ou outras imagens as quais queremos detectar o seu contexto (por exemplo: ao tirarmos uma foto de dentro de um comércio estamos interessados em saber se ele é uma loja de autopeças ou um supermercado “ao olhar o todo”, mas não que estejamos interessados em reconhecer cada item das gôndolas) ou seus objetos (ou seja, quais produtosestão dentro das gôndolas sendo mostradas na foto).
3. Bases de **texto**: são conjuntos de texto nos quais queremos entender o seu contexto; encontrar entidades (pessoas ou lugares); detectar similaridades entre documentos; traduções; entre outros.

Independentemente da base sendo trabalhada é importante relembrar de um conceito chamado *garbage in, garbage out*(entra lixo, sai lixo): algoritmos de ML aprendem em cima de uma base em que os humanos fornecem. Logo, é importante que nós (humanos) repassemos uma base confiável e previamente curada por nós. Ainda que a ideia possa soar interessante, por enquanto, os algoritmos ainda não conseguem adivinhar o que nós pensamos e, ainda, procurar novos dados da internet sem supervisão alguma.

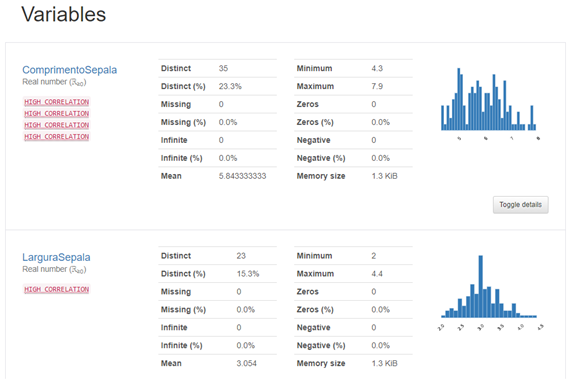
Nesse passo, nós podemos realizar uma **análise de dados exploratória** para ver como está o relacionamento entre os dados. Uma das bases de dados mais simples é a *Iris*, a qual possui dados sobre o comprimento e a largura das sépalas e pétalas de três espécies de flor do gênero Iris. O código a seguir combina duas bibliotecas para realizar a análise exploratória: o pandasspan >, responsável por gerar o DataFrame a partir dos dados de um arquivo CSV, e o span >pandas-profilingspan >, o qual nos gera um poderoso relatório sobre um DataFrame.



O relatório resultante nos fornece várias informações, como: o número de atributos (*variables*) e instâncias (*observations*), bem como os dados nulos e o espaço em memória gasto:



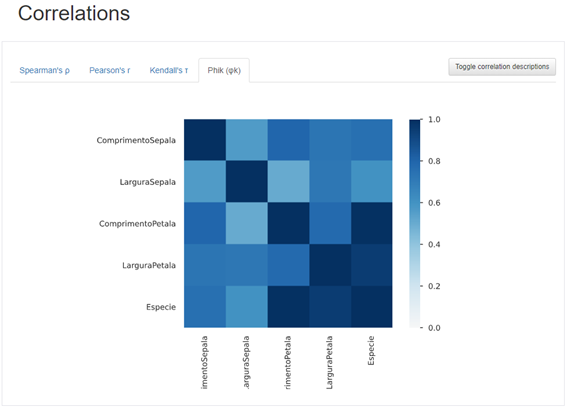
A distribuição dos dados entre as variáveis:



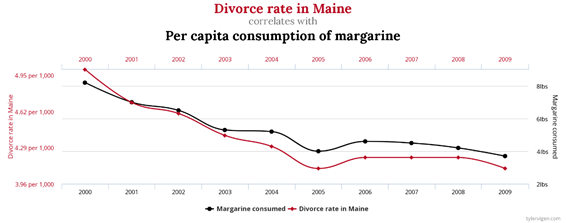
Dados duplicados:



As correlações com diferentes técnicas:



Entre outros. Isso é útil para detectarmos possíveis relacionamentos importantes entre as variáveis e que podem ser úteis para validar a base antes de avançarmos com os modelos. Sobre as correlações, vale também comentar sobre as “correlações espúrias”: não é porque temos uma alta correlação entre duas variáveis que isso significa alguma coisa útil. Um exemplo clássico é a alta correlação entre a porcentagem de divórcios no estado do Maine, nos EUA, com o consumo per capita de margarina:



Fonte: Vigen, [S.d.].

Observe que ambos possuem uma correlação altíssima: quando um aumenta, o outro também aumenta; quando um abaixa, o outro também abaixa. Por outro lado, um não influencia no outro de forma alguma. Tome o mesmo cuidado ao trabalhar com bases de dados, pois certamente não queremos detectar correlações espúrias nela!

Nesse momento, também podemos criar atributos adicionais ou manipular os que já temos. Os tamanhos máximo e mínimo de todas as dimensões da sépala e pétala são diferentes, e existem algoritmos (como as próprias redes neurais) que se beneficiam se todos estiverem na mesma escala. Logo, o código a seguir já poderia endereçar isso usando a MinMaxScaler:

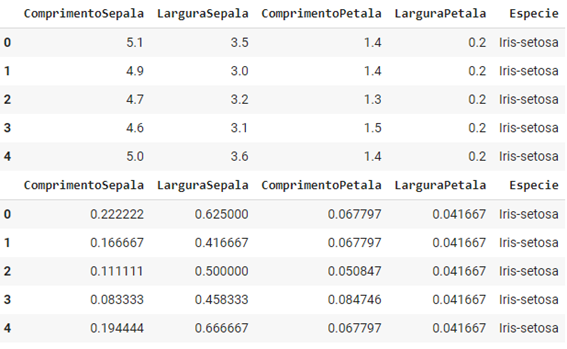
from sklearn.preprocessing import MinMaxScaler

display(df.head())

# colocando todas as colunas na mesma escala numérica
df_scaled = df.copy()
df_scaled[['ComprimentoSepala', 'LarguraSepala', 'ComprimentoPetala', 'LarguraPetala']] = MinMaxScaler().fit_transform(df_scaled[['ComprimentoSepala', 'LarguraSepala', 'ComprimentoPetala', 'LarguraPetala']])
display(df_scaled.head())



Agora, veja como eram as primeiras cinco linhas da tabela **antes**de colocarmos todos na mesma escala e como ficou depois:



Existem ainda outras técnicas, como:

* a transformação de categorias em múltiplas colunas numéricas;
* a extração de números como a semana, o dia da semana ou o mês a partir de uma coluna de data e hora; e
* o preenchimento de valores nulos com zeros ou outros valores.

**TEMA 4 – TRABALHANDO COM MODELOS**

Antes de qualquer coisa, vale um lembrete: **um modelo de Ciência de Dados não é necessariamente um modelo de ML!**Dependendo do caso, podemos ter modelos de áreas como Estatística e Econometria, por exemplo. Por outro lado, ML possui várias aplicações úteis no dia a dia.

As bibliotecas mais conhecidas em Python para a criação de modelos de ML são as que se seguem.

* Scikit-learn
  + O scikit-learn é potencialmente a biblioteca de ML mais versátil. Ela possui implementações de técnicas de aprendizagem supervisionada e não supervisionada.
  + Seus modelos vão desde a regressão linear até modelos de redes neurais, passando por SVMs, comitês de algoritmos, árvores de decisão e outros.
  + Possui técnicas de pré-processamento de dados e métricas de acuracidade dos algoritmos.
* Tensorflow, Keras e PyTorch
  + Os três são bibliotecas para o desenvolvimento de algoritmos de redes neurais (e *deep learning*). Pela complexidade desse tipo de algoritmo, recomenda-se que sejam usados computadores com mais poder de processamento.
  + O Tensorflow também possui um playground disponível para se testar diferentes configurações de redes com problemas simples e, assim, ver como o processo de treinamento de uma rede neural funciona na prática.
* LightGBM e XGBoost
  + Ambas as bibliotecas trabalham com *gradient boosting*, uma técnica de ML a qual é composta por várias árvores de decisão. Nessa técnica, as árvores vão sendo adicionadas no modelo para se obter um bom resultado de forma que uma possa reduzir o erro das anteriores.
* Statsmodels
  + O statsmodels é uma biblioteca com foco estatístico, e não exatamente em ML. De qualquer modo, ela permanece sendo relevante ao incluir funções para séries temporais (como o ARIMA) e outras técnicas de estatística descritiva.

Vejamos um exemplo do uso do scikit-learn. Testaremos dois modelos diferentes (RandomForest e SVC – um tipo de SVM específico para problemas de classificação) para a base de dados Iris, a qual comentamos anteriormente. Na prática, seria algo como: “aprenda os padrões entre os tamanhos da pétala e sépala de uma flor que determinam qual é a espécie dela. Validaremos em seguida com ‘novos’ exemplares de flor”.

Para melhor ilustrar como um algoritmo aprendeu utilizaremos também a biblioteca mlxtend junto com o matplotlib, que já vimos antes. Dito isso, observe o seguinte código. Note também os comentários (sempre iniciados com um “#”) para que saiba o que cada seção do código está fazendo. Nesse caso, estamos considerando somente o comprimento (e não a largura) da pétala e da sépala para tomar uma decisão.

Observe também que introduzimos duas novas funções que não vimos antes: a OrdinalEncoder, a qual transforma somente as categorias da coluna “Especie” que hoje estão como texto (“Iris-setosa”, “Iris-virginica” e “Iris-versicolor”) para números (0, 1 e 2). Essa transformação é necessária para utilizar a função plot\_decision\_regions do mlxtend. Além disso, usamos o train\_test\_split para dividir a nossa base de dados em duas partes: uma de treinamento e outra de teste. Veja que o parâmetro “test\_size” está com o valor 0.4. A explicação sobre esse parâmetro se encontra na própria documentação do scikit-learn e indica a porcentagem de distribuição aleatória para a base de treino e teste: no caso, 40% para a base de teste e os 60% restantes para a base de treinamento.

Ainda, veja que em alguns momentos existe também o parâmetro “random\_state”. Ele também é útil ao permitir a **reprodutibilidade** dos nossos testes. Vários algoritmos inicializam de forma aleatória e, para fins de teste e confiabilidade do algoritmo, em alguns casos, é importante garantir que conseguimos ter o mesmo resultado se rodarmos o mesmo código hoje ou amanhã, ou em computadores diferentes considerando a mesma base de dados. Dessa forma, ao indicarmos um valor para o “random\_state” (que chamamos de *seed*), garantiremos que o número aleatório utilizado para gerar a lógica do algoritmo será sempre o mesmo.

Veja, também, que no *train\_test\_split*não estamos só informando o dataframe *df*, mas sim dois parâmetros: df[['ComprimentoSepala', 'ComprimentoPetala']] (isto é, os atributos que queremos considerar no treinamento do modelo) e df['Especie'] (ou seja, o atributo que desejamos prever). O resultado, então, são quatro variáveis: os atributos utilizados para treinar e testar o modelo (X\_train e X\_test, respectivamente) – isto é, as variáveis que devem ser usadas pelo modelo aprender e reconhecer padrões para prever o nosso *target* (no caso, o atributo “Especie”); e o *target*em si, também correspondente à base de treinamento e teste e exatamente na mesma ordem do X\_train e X\_test, e que convenientemente chamamos de y\_train e y\_test.

# importação de libraries
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split

from mlxtend.plotting import plot_decision_regions

# transformando o texto da espécie em categorias numéricas
df['Especie'] = OrdinalEncoder().fit_transform(df[['Especie']]).astype(int)

# dividindo a base de dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(df[['ComprimentoSepala', 'ComprimentoPetala']], df['Especie'], test_size=0.4, random_state=0)

# treinando o modelo de RandomForest (RF)
modelo_rf = RandomForestClassifier(random_state=0)
modelo_rf.fit(X_train, y_train)

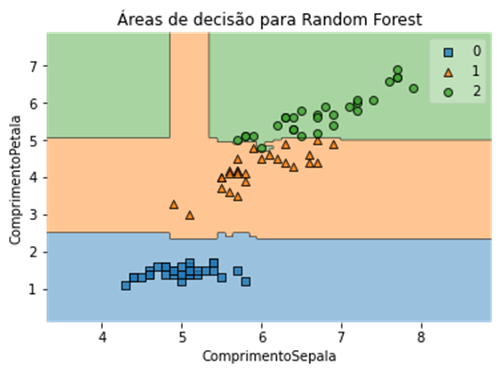
# mostrando como ele tomou decisões
fig = plot_decision_regions(X=X_train.values, y=y_train.values, clf=modelo_rf)
plt.xlabel('ComprimentoSepala')
plt.ylabel('ComprimentoPetala')
plt.title('Áreas de decisão para Random Forest')
plt.show()

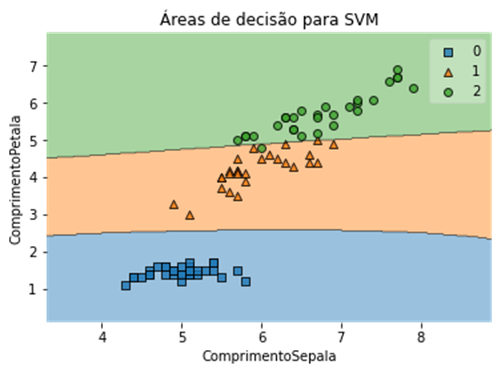
# treinando o modelo de GradientBoostingClassifier (RF)
modelo_svm = SVC(random_state=0)
modelo_svm.fit(X_train, y_train)

# mostrando como ele tomou decisões
fig = plot_decision_regions(X=X_train.values, y=y_train.values, clf=modelo_svm)
plt.xlabel('ComprimentoSepala')
plt.ylabel('ComprimentoPetala')
plt.title('Áreas de decisão para SVM')
plt.show()



Agora, veja como cada um dos modelos aprendeu:





**TEMA 5 – TRABALHANDO COM RESULTADOS**

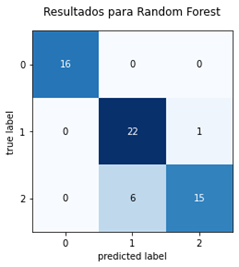
Veja que, no exemplo anterior, dividimos a base em uma de treinamento e de teste. Por outro lado, somente mostramos como o algoritmo aprendeu a partir da base de treinamento (no caso, X\_train e y\_train). Lembra quando comentamos que um dos passos necessários é a **avaliação**dos resultados? Pois bem – vejamos quais foram os resultados via matriz de confusão para as duas técnicas. Veja que agora estamos utilizando a base de testes para validar os resultados:

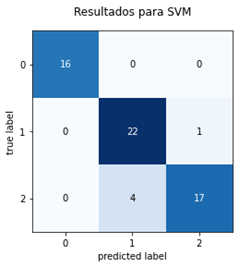
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix

plot_confusion_matrix(confusion_matrix(y_test, modelo_rf.predict(X_test)))
plt.title('Resultados para Random Forest')

plot_confusion_matrix(confusion_matrix(y_test, modelo_svm.predict(X_test)))
plt.title('Resultados para SVM')







Veja que cada linha indica a classe real e cada coluna indica qual foi o valor predito. Queremos o máximo de acertos para todas as classes. Logo, gostaríamos de ter o máximo de valores na diagonal principal da matriz: o máximo de valores para a primeira coluna da primeira linha; da segunda coluna da segunda linha e da terceira coluna da terceira linha. Veja que ambos acertaram todos os valores da classe 0 e erraram somente 1 caso da classe 1 (no caso, houve um 1 caso em que uma flor era da classe 1 e os algoritmos incorretamente classificaram como sendo da classe 2). O Random Forest classificou corretamente 15 flores como sendo da classe 2, mas 6 flores da mesma classe foram incorretamente classificadas como sendo da classe 1. O SVM foi um pouco melhor: acertou 17 flores e errou 4 da mesma classe.

**FINALIZANDO**

Aqui, nesta nossa conversa, tivemos a oportunidade de prover a você uma breve introdução ao mundo de Ciência de Dados em Python. O termo “Ciência” pressupõe o uso da Metodologia Científica para a análise e modelagem de dados e, com isso, também requer um rigor científico igualmente importante. Por essa razão, naturalmente, há uma fundamentação teórica forte por trás de um trabalho dessa área a qual não se restringe somente a ML: também é perfeitamente possível usarmos técnicas estatísticas para resolver esse problema.

Isso posto, mostramos o que se entende por ML, IA, Ciência de Dados, Advanced Analytics e demais termos utilizando uma analogia de uma praça de alimentação. Esse tipo de comparação é importante, uma vez que os termos podem ser confusos e serem trocados entre si ainda que representem coisas completamente diferentes. Também mostramos como funciona um projeto de Ciência de Dados – tanto do ponto de vista de gestão como do ponto de vista de implementação com Python.

Também ilustramos quais são as principais bibliotecas em Python para trabalharmos com ML com exemplos e mostrando algumas possibilidades da construção de algoritmos. Veja que estamos trabalhando com bibliotecas da mesma forma que vimos anteriormente e que aos poucos vamos também expandindo o nosso “repertório” de bibliotecas como o Tensorflow, scikit-learn, LightGBM, mlxtend e outros.

**REFERÊNCIAS**

VIGEN, T. **Spurious correlations**. Hachette UK, 2015.