



Aprenda com quem faz

Fundamentos de Aprendizagem de Máquina_

Túlio Philipe Ferreira e Vieira

2022



SUMÁRIO

Capítulo 1. Introdução ao Aprendizado de Máquina.....	6
Tipos de Aprendizado.....	8
Overfitting e Underfitting	14
Dimensionalidade	17
Linearidade e Não Linearidade	19
Capítulo 2. Tarefas do Aprendizado de Máquina	25
Séries Temporais.....	25
Tendência	26
Sazonalidade.....	28
Estacionariedade	28
Autocorrelação	29
Análise de Sentimento	31
Tokenização.....	32
Remoção de Stopwords.....	33
Stemização.....	34
Lematização	35
Processamento do Texto.....	36
Bag of Words.....	36
Word2vec	38
Processamento de Imagens (Visão Computacional)	40
Sistemas de Recomendação.....	41
Chatbot.....	45
Capítulo 3. Algoritmos de Machine Learning.....	49
Relembrando alguns conceitos e estatística.....	49
Correlação entre variáveis	53

Exemplo de aplicação e análise de correlação	55
Análise de Regressão	59
Regressão Linear.....	59
K-means.....	66
KNN.....	68
Árvore de Decisão	72
SVM.....	75
Capítulo 4. Capítulo 4. Redes Neurais Artificiais	79
Redes Neurais Artificiais e Deep Learning.....	79
Anexo A. Introdução ao Python.....	84
Instalação e Configuração do Ambiente	87
Segunda Forma de Utilização dos Códigos	90
Estrutura e Sintaxe do Python.....	90
Estrutura e Tipos de Dados	93
Tipos de Dados em Python	93
Dados Numéricos.....	94
Strings.....	96
Listas.....	96
Conjuntos (sets)	98
Dicionários (dict)	99
Tuplas	100
Controle de Fluxo	101
Condicionais	102
Construção de Loops.....	103
Funções	105
Classes	105



Anexo B. Desenvolvendo aplicações com Keras.....	107
Diferenças entre Keras, TensorFlow e Pytorch	107
Entendendo o Keras	109
Referências	112



> Capítulo 1



Capítulo 1. Introdução ao Aprendizado de Máquina

Podemos até não perceber, mas o aprendizado de máquina está presente em nosso cotidiano. Quando realizamos compras on-line e recebemos indicações de produtos, quando interagimos com atendentes eletrônicos por meio de *chatbots*, quando somos “salvos” de e-mails promocionais que são direcionados para nossa caixa de *spam*, quando vamos tirar uma fotografia que é disparada ao sorrirmos e até quando assistimos filmes que não estávamos acostumados e acabamos por gostar da indicação. Todas essas ações são exemplos de aplicações dos algoritmos de aprendizado de máquina na nossa vida.

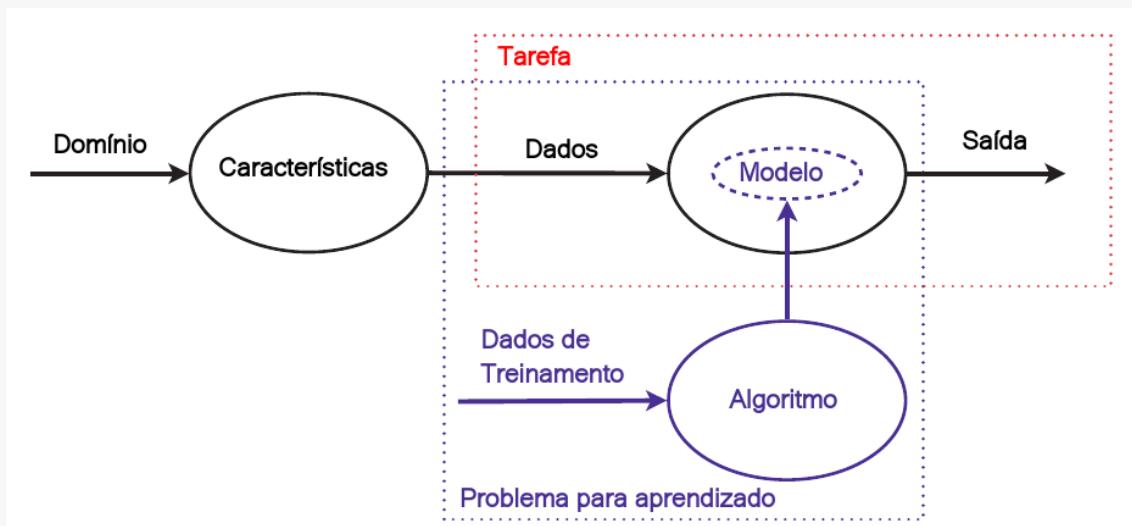
Mark Weiser, pesquisador e cientista da computação conhecido como o pai da computação ubíqua, em seu artigo *The computer for the 21st century* diz que “as tecnologias mais profundas e duradouras são aquelas que desaparecem. Elas dissipam-se nas coisas do dia a dia até tornarem-se indistinguíveis” (WEISER, 1999). Essa ponderação apresentada no contexto da computação ubíqua pode ser utilizada no contexto do aprendizado de máquina. Como mostrado nos exemplos acima, o aprendizado de máquina e a inteligência artificial já fazem parte do nosso cotidiano e não mais percebemos as profundas transformações que essas tecnologias trazem para as nossas vidas.

Quando assistimos a um filme ou série é apresentado ao final uma sugestão de possíveis títulos após a mensagem: “Quem assistiu a esse filme também gostou de....”, nesse caso estamos utilizando um conjunto de algoritmos de aprendizado de máquina. Segundo Flach (2012), o aprendizado de máquina pode ser visto como um estudo sistemático de algoritmos e sistemas que melhoram o conhecimento e *performance* por meio da experiência. Nesse exemplo de recomendação de filmes, o algoritmo melhora a *performance* por meio das respostas dos usuários que indicam se

as sugestões de filmes ou séries foram boas ou não, ou seja, a partir dos *feedbacks* dos usuários é possível melhorar as sugestões de filmes ou séries. Nesse cenário, a experiência do algoritmo é adquirida mediante os *feedbacks* dos usuários e a *performance* pode ser vista como a capacidade do algoritmo recomendar filmes e séries que os expectadores gostem de assistir.

Flach (2012) propõe uma visão geral sobre o funcionamento de um algoritmo de aprendizado de máquina. A Figura 1 apresenta esse modelo. Por meio dela é possível perceber que para o funcionamento correto de um algoritmo são necessários alguns componentes. Esses componentes são a tarefa a ser resolvida, o modelo e o algoritmo de aprendizagem.

Figura 1 – Componentes de um sistema de aprendizado de máquina.



Fonte: adaptado de Flach (2012).

Para o exemplo de recomendações de filmes, a **tarefa** pode ser vista como recomendar filmes que os usuários gostem de assistir, ou seja, filmes que refletem os gostos dos usuários. Como sabemos, cada usuário tem um perfil ou gosto por um determinado tipo de filme/série. Desse modo, é necessário que o sistema seja capaz de realizar uma recomendação personalizada para cada usuário ou grupos de usuários. Assim, para esse exemplo, é necessário construir um **modelo** que seja capaz de mapear e

encontrar filmes relevantes a serem oferecidos para cada perfil de usuário. Nesse contexto, os **dados** representam as características dos filmes que podem ser os atores presentes em cada um dos filmes, o gênero do filme, a duração e até a época em que ocorre a trama. Para conhecer o perfil do usuário, podem ser utilizados, por exemplo, quais foram os últimos filmes assistidos (do início ao fim), quais foram as notas atribuídas pelo usuário e até quais são os filmes presentes na lista de “assistir mais tarde”.

O algoritmo deve ser capaz de receber todas essas características dos filmes e dos perfis dos diferentes usuários e conseguir encontrar um mapeamento adequado entre o filme/série assistido e qual é o filme/série mais adequado a ser oferecido, ou seja, qual ou quais devem ser os filmes/séries que tem maior probabilidade de o usuário gostar de assistir. Desse modo, a tarefa de recomendar filmes é alcançada por meio de um modelo que é obtido mediante o treinamento de um algoritmo que “aprendeu” as características dos filmes que o usuário mais gosta de assistir. É por meio do treinamento que o algoritmo aprende a encontrar o perfil de cada um dos usuários e construir um modelo que seja capaz de recomendar o filme/série mais indicado para ele.

Tipos de Aprendizado

Os algoritmos de aprendizado de máquina podem ser divididos em três grandes grupos:

- Algoritmos supervisionados;
- Algoritmos não supervisionados;
- Algoritmos baseados em aprendizagem por reforço.

Essa divisão reflete, principalmente, como esses grupos de algoritmos adquirem a experiência para construir um determinado modelo a fim de resolver uma tarefa.

Os algoritmos que pertencem ao grupo dos supervisionados apresentam a característica de aprenderem por meio do mapeamento entre exemplos de dados de entrada e saída. Esses exemplos constituem os dados a serem utilizados no processo de treinamento do algoritmo. É nessa fase de treinamento que o algoritmo encontra o mapeamento entre as entradas e saídas. Para os algoritmos supervisionados, esse mapeamento é obtido por meio de dados históricos que indicam qual é a combinação de características que geram uma determinada saída.

O primeiro tipo de aprendizado é conhecido como **supervisionado**. Um exemplo de dados históricos utilizados em modelos **supervisionados** pode ser encontrado na Figura 2. Ela apresenta um conjunto de dados que contém informações sobre a altura, o peso e a classificação utilizada pela Organização Mundial de Saúde para calcular o peso ideal de cada pessoa (Departamento de Atenção Básica, 2006). Esses dados podem ser utilizados para endereçar a tarefa de classificar se um indivíduo está com sobrepeso ou não, utilizando a altura e o peso dessa pessoa.

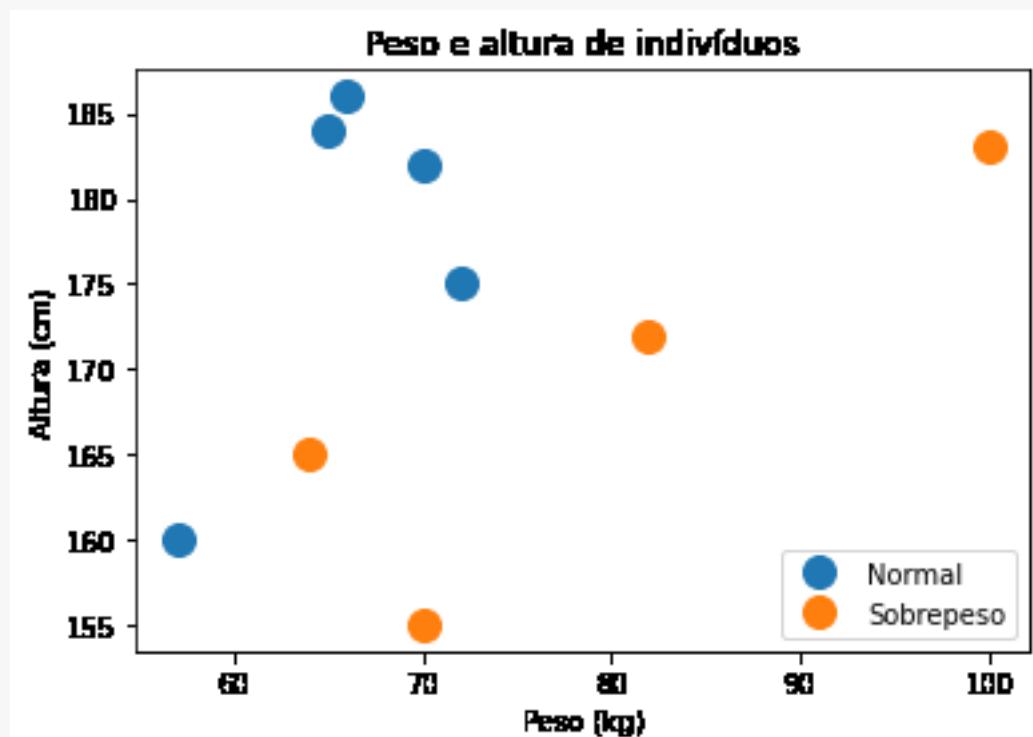
Figura 2 – Conjunto de dados de altura e peso de pessoas

	Peso (kg)	Altura (cm)	Sobrepeso
0	70	155	Sobrepeso
1	57	160	Normal
2	100	183	Sobrepeso
3	72	175	Normal
4	64	165	Sobrepeso
5	70	182	Normal
6	82	172	Sobrepeso
7	66	186	Normal
8	65	184	Normal

O conjunto de dados (*dataset*) presente na Figura 2 apresenta nas colunas as **características** (peso, altura e a indicação de sobrepeso ou não do indivíduo) de cada pessoa entrevistada. Essas características também são chamadas de *features*. São essas *features* que devem ser utilizadas para o aprendizado do algoritmo e construção do modelo de previsão. Cada linha desse conjunto de dados apresenta os valores de peso, altura e marcação de sobrepeso ou não para cada pessoa. Cada uma dessas linhas é chamada de **instâncias** do *dataset*.

A Figura 3 apresenta um gráfico com os valores de peso e altura das diferentes pessoas entrevistadas e a classificação sobre o sobrepeso ou não desses indivíduos. Pelas Figuras 2 e 3 é possível perceber que existem duas diferentes “classes de pessoas”: pessoas com sobrepeso e normais.

Figura 3 – Dispersão dos dados de peso e altura de indivíduos



Para que um algoritmo seja classificado como supervisionado, é necessário que sejam apresentadas instâncias de **entrada** e **saída** para o algoritmo. Nesse exemplo, as **entradas** correspondem à **Altura** e ao **Peso** de

cada um desses indivíduos. Já a **saída** corresponde à classificação de sobre peso ou não de cada uma dessas instâncias. As saídas também são conhecidas como *label* ou *target* em um conjunto de dados. Desse modo, os algoritmos supervisionados podem realizar o mapeamento entre as entradas e as saídas para aprender as características de cada uma das classes alvo (sobre peso ou não).

Como mostrado, os algoritmos supervisionados necessitam de um conjunto de instâncias para cada uma das classes utilizadas durante o treinamento. Já a classe de algoritmos não supervisionados não necessita dos dados de saída para realizar previsões.

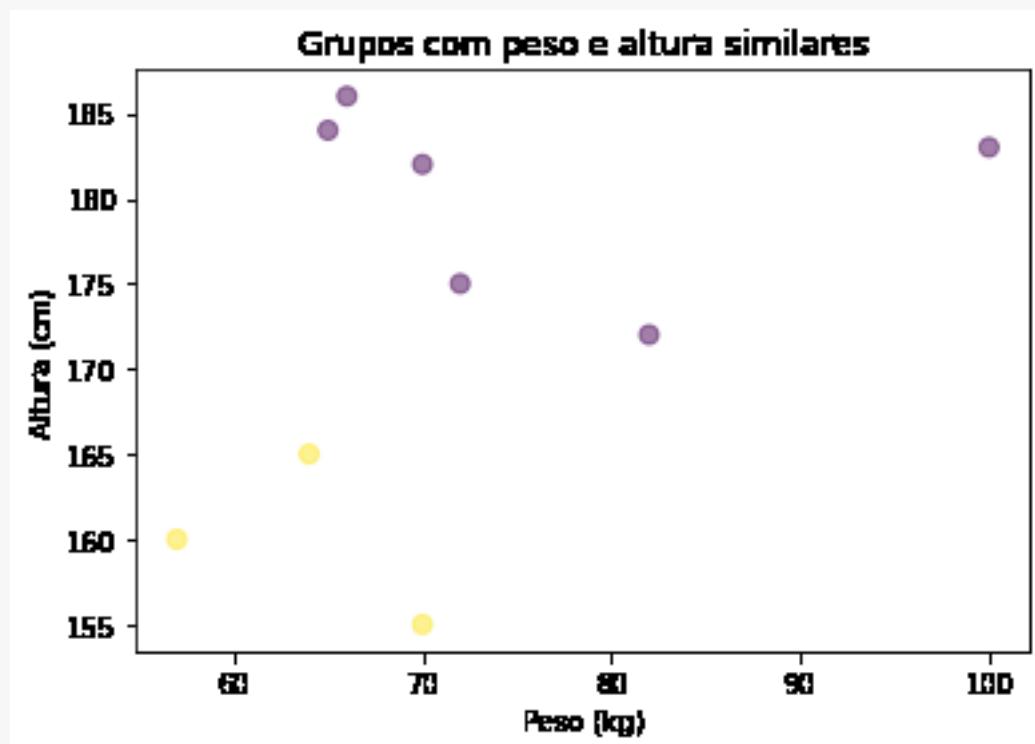
O segundo tipo de aprendizado corresponde ao aprendizado **não supervisionado**. Por exemplo, ao invés de criar um modelo que seja capaz de classificar se uma pessoa está com sobre peso ou não, podemos utilizar apenas os dados de entradas presentes na Figura 1.2 para encontrar quais são os indivíduos que possuem Peso e Altura similares. A Figura 4 apresenta esse conjunto de dados.

Figura 4 – Exemplo de dataset utilizado em modelos não supervisionado

	Peso (kg)	Altura (cm)
0	70	155
1	57	160
2	100	183
3	72	175
4	64	165
5	70	182
6	82	172
7	66	186
8	65	184

Como pode ser visto na Figura 4, para cada uma das instâncias, não existe uma saída (*label*) associada. Desse modo, o algoritmo deve encontrar grupos de indivíduos que possuem Peso e Altura similares. Esse procedimento de agrupar os dados em grupos com características similares é conhecido como *clusterização (clustering)* ou agrupamento. Essa técnica de agrupamento corresponde ao exemplo mais conhecido de algoritmos não supervisionados. A Figura 5 apresenta a divisão desses indivíduos (instâncias) em dois diferentes grupos. Cada um desses grupos apresenta pessoas com alturas e pesos similares. Podemos entender, por exemplo, que o grupo em amarelo representa pessoas com baixo peso e altura enquanto o grupo em roxo representa pessoas com maiores pesos e alturas.

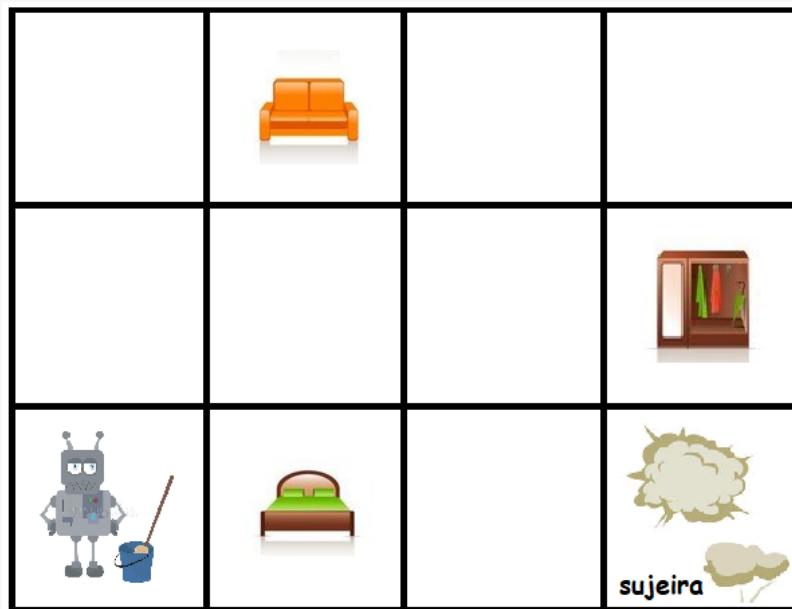
Figura 5 – Exemplo de agrupamento para o dataset presente na Figura 4



O terceiro tipo de aprendizado é o aprendizado **por reforço**. Ele corresponde a um tipo especial de aprendizado, pois é baseado no **feedback do ambiente**. No aprendizado por reforço não são utilizados *labels*, mas como esse aprendizado ocorre por meio de *feedbacks* do ambiente, eles também não podem ser classificados como não supervisionados. No

aprendizado por reforço, os algoritmos interagem com o ambiente a fim de aprenderem as características do problema a ser resolvido. A Figura 6 apresenta um exemplo de problema a ser resolvido utilizando o aprendizado por reforço.

Figura 6 – Exemplo de aprendizado por reforço



Nesse problema, temos um robô aspirador que deve chegar até o lixo a ser recolhido. Para chegar até o lixo a ser recolhido, esse robô deve encontrar um caminho possível sem tocar em nenhum dos obstáculos. No cenário proposto na Figura 6 temos o robô, os móveis de uma residência e a sujeira a ser limpa.

O dever desse robô é chegar até a sujeira sem tocar na mobilha existente. Para isso, o robô deve escolher os diferentes caminhos possíveis e identificar como solução aquele caminho que gera a recompensa (coleta da sujeira) com o menor número de batidas nas mobilhas. A cada decisão tomada pelo robô (passos em uma direção) que não gere encontros com os obstáculos dará uma recompensa a esse robô e a cada caminho que ocasione o encontro com uma mobilha é subtraída uma recompensa. O valor total de recompensas é calculado quando o robô aspirador conseguir

alcançar a recompensa final que é o recolhimento da sujeira. Desse modo, por meio de recompensas é possível que o robô encontre o caminho “ideal” mediante os *feedbacks* obtidos pela interação com o meio (recompensas ou não).

Overfitting e Underfitting

Os modelos de aprendizado de máquina aprendem por meio da experiência a fim de melhorarem o desempenho e resolverem uma tarefa específica. Esse aprendizado ocorre no momento em que o algoritmo é treinado. Durante o processo de **treinamento** o algoritmo identifica os padrões existentes no conjunto de dados. Ao identificar esses padrões é possível que o modelo consiga generalizar o que ele “aprendeu” e aplicar esse aprendizado em outras instâncias que não foram apresentadas ao algoritmo durante o processo de treinamento. O processo de avaliar a capacidade do algoritmo em generalizar o aprendizado é conhecido como **teste**.

Quando um algoritmo gera um modelo capaz de generalizar o aprendizado, dizemos que o modelo é **balanceado**, ou seja, o modelo é capaz de resolver a tarefa proposta mesmo que instâncias não vistas durante o processo de treinamento sejam apresentadas.

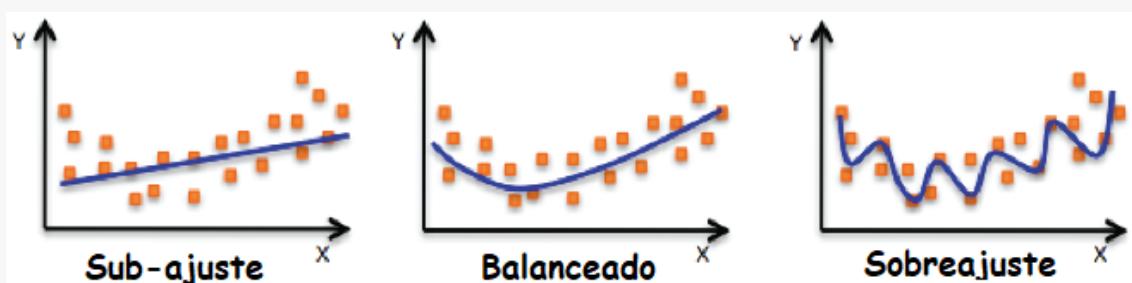
Um outro caso pode ocorrer quando o modelo apresenta um desempenho muito abaixo do esperado, ou seja, o modelo gerado pelo algoritmo de aprendizado não consegue capturar o relacionamento existente entre os dados de entrada e saída. Desse modo, o mapeamento entre as saídas e entradas ocorre de maneira precária e o modelo não é capaz de resolver satisfatoriamente a tarefa proposta. Essa situação é conhecida como **sub-ajuste** ou *underfitting*.

Outra possibilidade é a de que o modelo, durante o processo de treinamento, apresenta um desempenho satisfatório, mas quando novas instâncias ainda não vistas pelo modelo são apresentadas na etapa de teste

do modelo, o desempenho do algoritmo não é satisfatório. Nesse caso, o modelo “memorizou” as características das instâncias utilizadas para o treinamento, mas **perdeu a capacidade de generalização**, ou seja, o desempenho do modelo só é satisfatório durante a etapa de treinamento. Esse caso é conhecido como **sobreajuste** ou *overfitting*. Desse modo, esse modelo também não é capaz de resolver a tarefa de maneira satisfatória.

A Figura 7 mostra uma comparação entre modelos que apresentam o sub-ajuste (underfitting), o balanceamento adequado e o sobreajuste (overfitting). Considerando que cada um dos pontos na figura representa a instância de um problema e que as curvas em azul representam os modelos gerados, por meio da Figura 7 é possível perceber que no sub-ajuste a complexidade do modelo gerado é menor do que a apresentada pelo problema, já no sobreajuste, o modelo apresenta uma complexidade maior do que a apresentada pelo problema. Em um modelo de aprendizado de máquina bem ajustado (balanceado) existe um balanceamento entre a complexidade do modelo e a complexidade do problema a ser resolvido.

Figura 7 – Comparação entre Underfitting, overfitting e modelos平衡ados.



Fonte: adaptado de Amazon, (2021).

Nos casos em que ocorre o sobreajuste ou o sub-ajuste, é necessário que algumas medidas sejam tomadas, já que o modelo não apresenta o desempenho necessário para a solução de uma tarefa.

Para reduzir o underfitting, algumas técnicas podem ser utilizadas:

- Aumentar a complexidade do modelo;

- Aumentar o número de características utilizadas pelo modelo;
- Remover o ruído dos dados. O ruído pode ser visto como valores que provocam alterações no comportamento esperado dos dados;
- Aumentar a base de treinamento para o algoritmo.

Quando ocorre o overfitting, algumas estratégias podem ser adotadas, como:

- Aumentar a quantidade de dados para o treinamento;
- Reduzir a complexidade do modelo;
- Parar o treinamento antes de alcançar o resultado final (early stopping);
- Aplicar estratégias de penalização como as regularizações Ridge e Lasso.

Para identificar o overfitting ou underfitting de um modelo, é necessário que os dados sejam divididos entre treinamento e teste. Desse modo, os dados de treinamento devem conter instâncias **distintas** das presentes no grupo de dados de teste. Uma sugestão de divisão consiste em escolher entre 70% e 80% das instâncias para treinamento e 30% a 20% das instâncias para testar o desempenho do modelo.

A Figura 8 apresenta um exemplo de divisão dos dados presentes na Figura 2 para treinamento e teste do modelo.

Figura 8 – Divisão de dados para treinamento e teste.

	Peso (kg)	Altura (cm)	Sobrepeso	
0	70	155	Sobrepeso	<input type="checkbox"/> Treino
1	57	160	Normal	<input checked="" type="checkbox"/> Teste
2	100	183	Sobrepeso	<input checked="" type="checkbox"/> Teste
3	72	175	Normal	<input type="checkbox"/> Treino
4	64	165	Sobrepeso	<input type="checkbox"/> Treino
5	70	182	Normal	<input type="checkbox"/> Treino
6	82	172	Sobrepeso	<input type="checkbox"/> Treino
7	66	186	Normal	<input type="checkbox"/> Treino
8	65	184	Normal	<input checked="" type="checkbox"/> Teste

Segundo o princípio conhecido como Occam's razor (Navalha de Ockham), os modelos de aprendizado de máquina devem ser mantidos os mais simples possíveis, ou seja, algoritmos que apresentam uma menor complexidade e uma boa generalização devem ser escolhidos em detrimento a modelos mais complexos. Seguindo esse princípio, começar com modelos mais simples e ir aumentando a complexidade a fim de alcançar uma boa generalização torna-se uma estratégia bastante eficiente para resolver problemas com o aprendizado de máquina.

Dimensionalidade

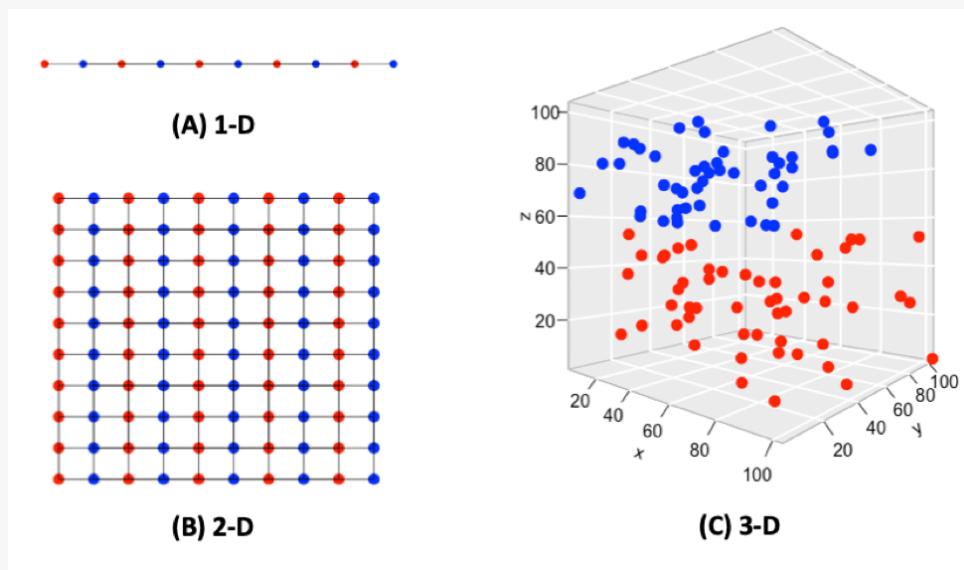
A dimensionalidade para o aprendizado de máquina corresponde às dimensões presentes em um problema. Por exemplo, o problema presente na Figura 4 apresenta duas dimensões (altura e peso dos indivíduos). O gráfico presente na Figura 5 mostra que a distribuição de todas as instâncias pode ser visualizada por meio de um plano cartesiano, com o eixo X apresentando a característica altura e o eixo Y apresentando a característica peso. Desse modo, a dimensionalidade pode ser entendida como o espaço dimensional que representa as características de um problema.

Em problemas reais, é comum existir um grande número de características que compõem um conjunto de dados. Por exemplo, imagine a quantidade de características necessárias para realizar a previsão da temperatura. Quanto mais variáveis são utilizadas para realizar a construção de um modelo, maior será a dimensionalidade.

Quanto maior a quantidade de dimensões presentes em um problema, mais complexo será para o modelo conseguir executar a tarefa de maneira satisfatória. Com o aumento da dimensionalidade a densidade dos dados presentes no *dataset* diminui exponencialmente. Com uma menor densidade é necessário que sejam apresentadas uma maior quantidade de instâncias para que o modelo consiga aprender as características do problema.

Para entender o efeito desse aumento da dimensionalidade, considere um *dataset* que contenha 3 características e 1000 instâncias. Digamos que cada uma dessas 3 características possui valores que variam entre 0-10. Desse modo, uma visualização possível desse problema em 3 dimensões pode ser vista como um cubo de lados 10x10x10. A densidade ou quantidade de pontos por unidade desse cubo é igual a $1000/1000 = 1$, ou seja, 1 instância para cada parte do cubo. Se ao invés de 3 características possuíssemos 5 e mantivermos a mesma quantidade de instâncias, essa densidade cairia para $1000/100000 = 0,01$. Esse fenômeno é conhecido como a Maldição da Dimensionalidade. Quando isso ocorre, para que o modelo consiga ter um bom desempenho, é necessário que quantidade de instâncias seja aumentada, o que, muitas vezes, não é possível. Assim, garantir que apenas as características importantes para o aprendizado sejam escolhidas é fundamental para conseguir que o modelo resolva de maneira satisfatória a tarefa proposta. A Figura 9 mostra a representação gráfica dessa redução da densidade para o caso apresentado.

Figura 9 – Exemplo da redução da densidade e a maldição da dimensionalidade.

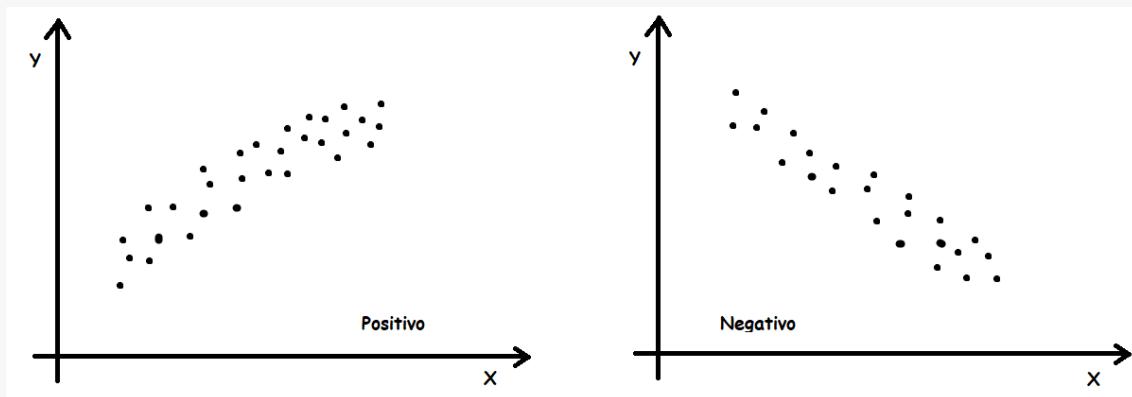


Fonte: Tiange (2019).

Linearidade e Não Linearidade

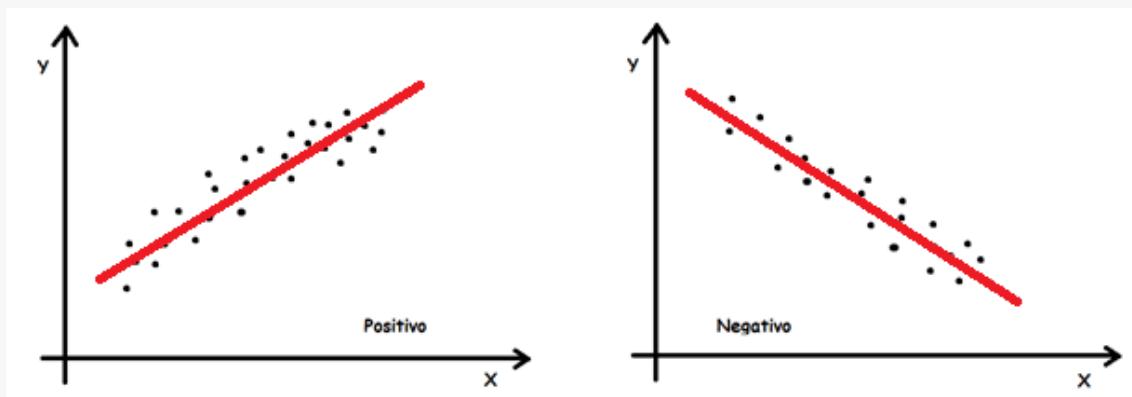
Os conceitos de linearidade e não linearidade, podem ser utilizados para definir o comportamento de um modelo ou de um problema. Um problema é dito linear quando o relacionamento entre os dados de entrada e saída é linear, ou seja, quando um dado de entrada aumenta, a saída também aumenta na mesma proporção. Esse tipo de relação linear é chamado de relação linear positiva. Quando uma variável de entrada aumenta e o valor da saída diminui na mesma proporção do aumento da entrada, temos um exemplo de relação linear negativa. A Figura 10 apresenta um exemplo de problemas lineares positivos e negativos.

Figura 10 – Exemplo de relacionamento linear positivo e negativo



Para os modelos serem considerados lineares eles devem utilizar equações lineares para modelar o relacionamento entre os valores de entrada e saída. A Figura 11 apresenta um modelo de regressão linear para encontrar os relacionamentos entre a variável de entrada x e de saída y.

Figura 11 – Exemplo de modelo linear.



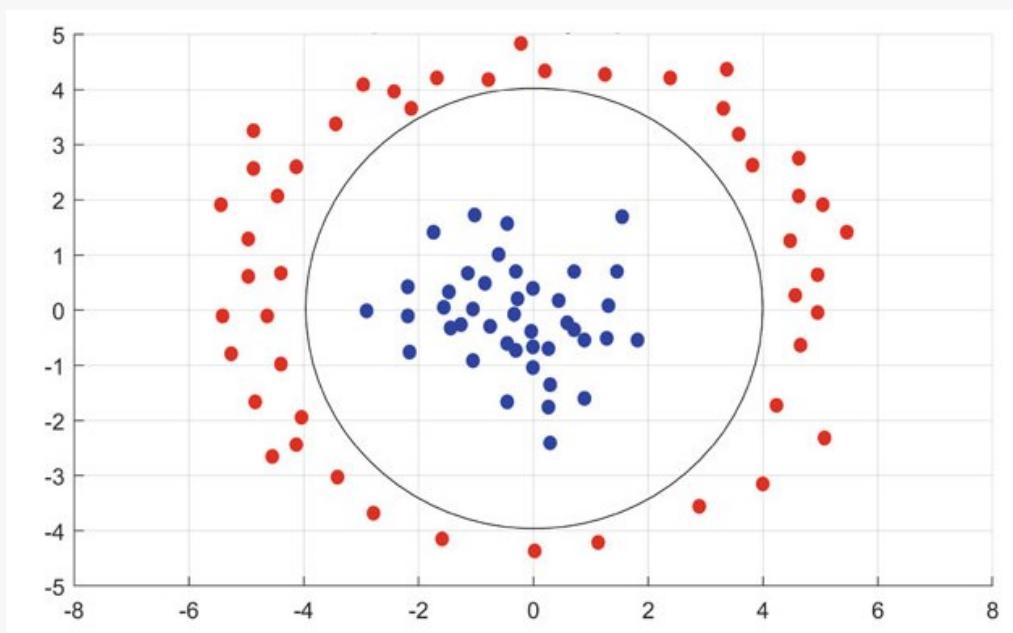
Em tarefas de classificação, também podem existir modelos de separação linear entre as diferentes classes. A Figura 12 mostra um exemplo disso. Analisando essa imagem, é possível perceber que por meio de uma reta é possível separar as duas classes de dados (vermelhas e verdes).

Figura 12 – Exemplo de modelo linear em classificação.



Um exemplo de problema não linear é apresentado na Figura 13. Pelos dados presentes nela, não é possível gerar uma reta de separação que garanta a correta classificação de todas as instâncias presentes. Essa imagem apresenta um exemplo de classificação que não linear.

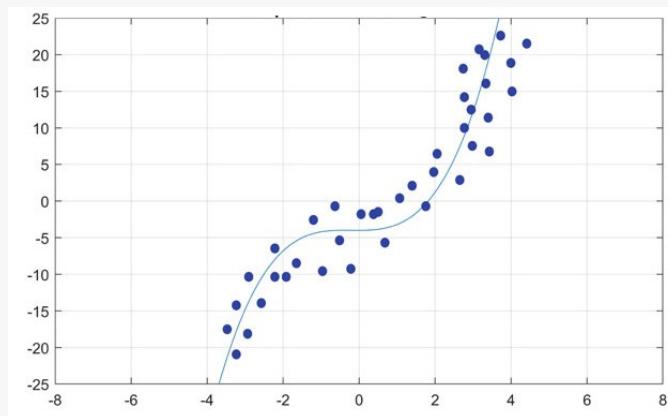
Figura 13 – Exemplo de problema não linear em classificação.



Fonte: adaptado de Joshi (2020).

Em problemas de regressão também é possível que o relacionamento seja não linear e o modelo para resolver essa tarefa também deve ser não linear. A Figura 14 apresenta um exemplo desse caso.

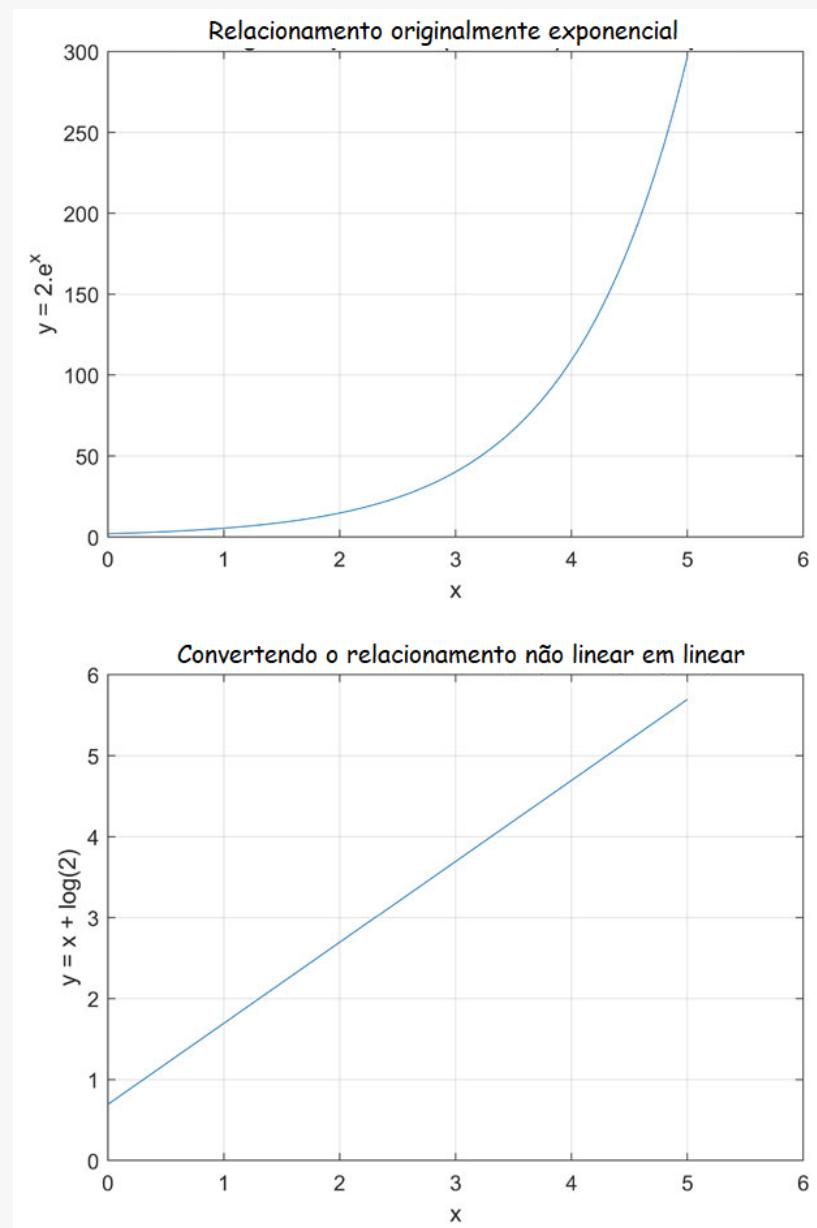
Figura 14 – Exemplo de modelo não linear em regressão.



Fonte: adaptado de Joshi (2020).

Problemas não lineares podem ser linearizados por meio de algumas transformações matemáticas, como presente na Figura 15. Nesse caso, é aplicada uma transformação logarítmica sobre os dados de entrada a fim de encontrar um modelo linear que expresse o relacionamento entre os valores de entrada e saída.

Normalmente, quando problemas não lineares não são linearizados e tentamos adequar um modelo linear ao problema, acabamos por gerar um *underfitting* do modelo. No caso contrário, podemos gerar um *overfitting* para o nosso modelo, ou seja, quando utilizamos um modelo não linear para resolver um problema linear. Portanto, garantir a generalização dos modelos é fundamental para a solução de qualquer tarefa.

Figura 15 – Exemplo de transformação linear



> Capítulo 2



Capítulo 2. Tarefas do Aprendizado de Máquina

Neste capítulo, serão apresentados e discutidos alguns conceitos de diferentes áreas do conhecimento que utilizam o Aprendizado de Máquina para encontrar soluções de problemas reais. Ao final, o aluno terá conhecimento de aplicações de aprendizado de máquina nas seguintes áreas:

- Séries Temporais;
- Análise de Sentimento;
- Processamento de Imagens;
- Sistemas de Recomendação;
- Chatbots.

Séries Temporais

Séries temporais estão presentes no cotidiano de todas as pessoas. Quando um vendedor anota a quantidade diária de produtos vendidos e, ao final do mês, analisa esses dados, ele está empregando os conceitos de análise de séries temporais. Elas podem ser definidas como uma sequência de dados ou observações coletadas, sucessivamente, em intervalos periódicos (PAL; PRAKASH, 2017). Normalmente, o período de coleta dos dados permanece constante durante toda a sequência de observações.

Exemplos de séries temporais são encontrados na variação do volume de vendas de produtos, no mercado de ações, na previsão do tempo, no consumo diário de energia, na variação da demanda diária de abastecimento, no acompanhamento do Produto Interno Bruto (PIB) e em várias outras áreas. A sua utilização correta pode auxiliar no melhor entendimento de um processo e conduzir a melhorias na qualidade de produtos. Os objetivos da análise de séries temporais podem ser categorizados em:

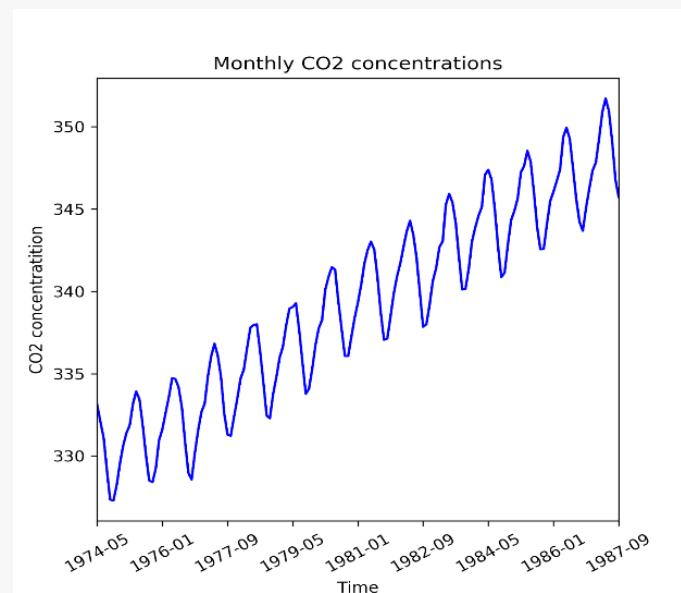
- Entender e interpretar os elementos que conduzem o comportamento da variável medida durante o período observado (compreender a dinâmica do sistema);
- Prever o estado futuro do sistema através dos dados observados.

As séries temporais são compostas por uma série de elementos que modelam o comportamento da variável observada. As estruturas internas que compõem uma série temporal podem ser compostas de tendência, sazonalidade, autocorrelação, movimento cíclicos, dentre outras. Para compreender como essas estruturas influenciam no comportamento das séries temporais, vamos analisar cada uma delas através da análise gráfica.

Tendência

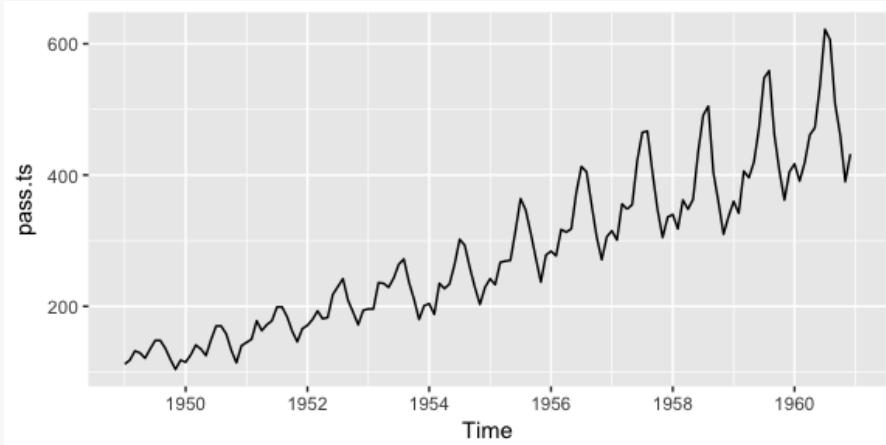
Quando uma série temporal exibe algum comportamento de crescimento ou decaimento ao longo do tempo, é dito que ela apresenta uma tendência. Uma maneira simples de identificar essa tendência é por meio da análise gráfica. A Figura 16 apresenta um gráfico com tendência de crescimento. Através de uma análise mais aprofundada, é possível identificar que essa tendência de crescimento é linear. A Figura 17 mostra um gráfico com tendência de crescimento exponencial.

Figura 16 – Série Temporal com tendência de crescimento linear.



Fonte: PAL e PRAKASH (2017).

Figura 17 – Série Temporal com tendência de crescimento linear.



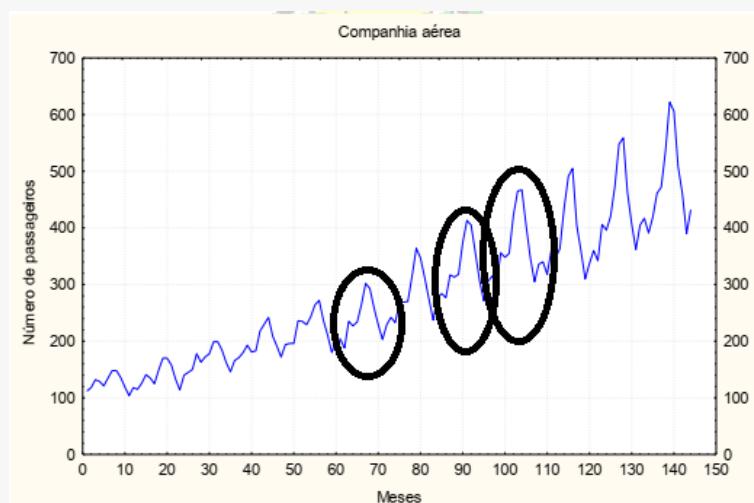
Fonte: BISGAARD; KULAHCI, 2011.

Por meio da análise de tendência, é possível identificar o comportamento em longo prazo da série temporal. Por exemplo, analisando o gráfico da Figura 16 é possível prever que a concentração de CO₂ no ano de 1988 será maior que a identificada no ano de 1987. Esse tipo de análise é interessante para realizar uma rápida, mas não precisa, previsão dos dados.

Sazonalidade

Sazonalidade é a manifestação repetida e periódica de variações nas séries temporais. As sazonalidades são padrões que podem ser identificados graficamente. Esses padrões indicam um comportamento regular da série temporal. Por exemplo, no gráfico presente na Figura 18, é possível identificar que o número de passageiros da companhia aérea varia, periodicamente, a cada ano. Por meio dessa análise, é possível identificar picos e vales a cada ano, essas oscilações podem ocorrer devido a feriados, férias etc., que modificam os períodos de maior procura por passagens aéreas. Essas flutuações indicam a sazonalidade desses dados.

Figura 18 – Série Temporal com sazonalidade.

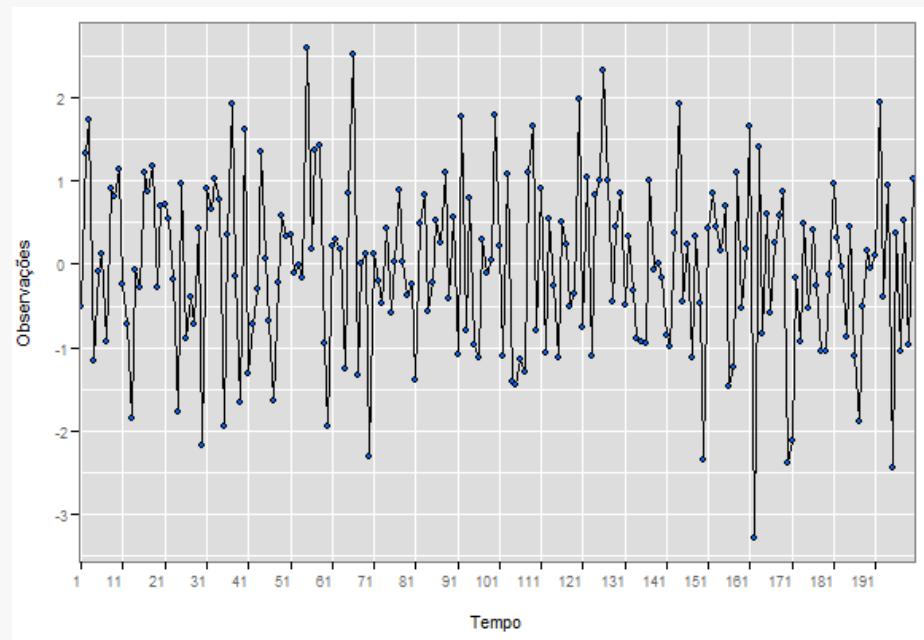


Fonte: adaptado de EHLERS, (2007).

Estacionariedade

Uma série temporal é dita estacionária quando ela se desenvolve no tempo aleatoriamente ao redor de uma média constante, refletindo alguma forma de equilíbrio estável. Na prática, quase todas as séries temporais apresentam algum tipo de não estacionariedade. A Figura 19 mostra um exemplo de série estacionária. É possível identificar que os dados dessa série temporal são aleatórios e apresentam uma variação em torno do valor zero. Esse tipo de comportamento, normalmente, não é encontrado, pois as séries possuem tendências e sazonalidades.

Figura 19 – Série Temporal estacionária.



Fonte: ACTION, 2019.

Para a análise de séries temporais por meio de modelos paramétricos, como *Autoregressive Integrated Moving Average* (ARIMA), é necessário que exista uma transformação das séries não estacionárias em estacionárias. Esse tipo de transformação é, normalmente, realizado por meio do procedimento de diferenciação das séries temporais. Quando utilizamos os algoritmos de Deep Learning, não é necessário esse tipo de modificação, pois a dinâmica do sistema é capturada pelo treinamento do algoritmo.

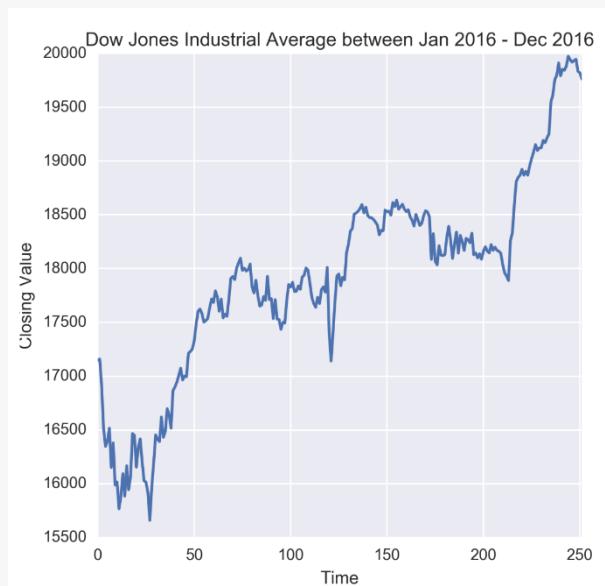
Autocorrelação

Autocorrelação reflete o grau de dependência linear entre os índices (valores) de uma série temporal. Uma autocorrelação positiva indica que os valores presentes e futuros movem em uma mesma direção. Caso esse valor seja negativo, indica que os valores movem em direções opostas. Se a autocorrelação for próxima a zero, indica que os valores não possuem dependência linear. A análise de autocorrelação é importante para análise

de séries temporais, pois é ela que indica a quantidade de passos anteriores que devem ser utilizados para prever o futuro de uma medida.

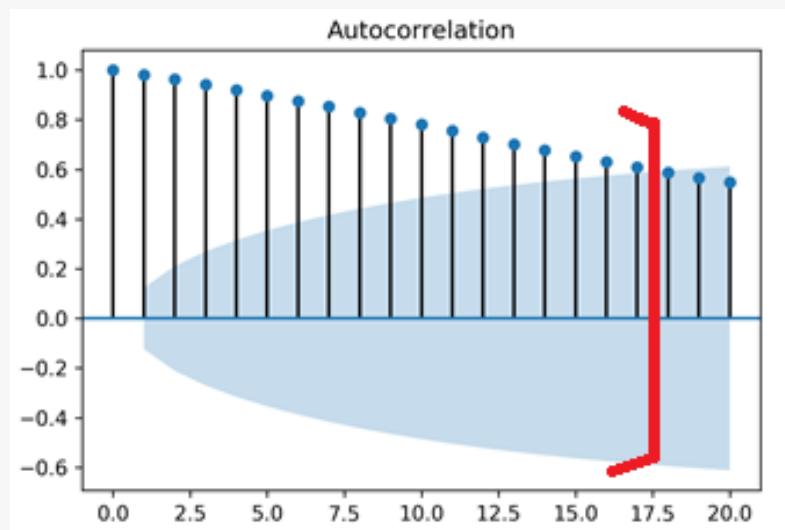
A Figura 20 indica a variação do índice Dow Jones de janeiro de 2016 a dezembro de 2016. A Figura 21 mostra o resultado de autocorrelação após a aplicação da *Autocorrelation Function* (ACF). Analisando o gráfico da Figura 21, verifica-se que após 18 medidas (passos), as medidas entram na região em que não é possível, estatisticamente, garantir que exista dependência entre os dados. Assim, podemos dizer que a medida (dado) atual é influenciado por 18 dados anteriores. Logo, podemos utilizar como entrada em nossa rede neural o dado atual e 18 passos anteriores para realizar a previsão do preço do índice Dow Jones Industrial.

Figura 20 – Série Temporal Índice Dow Jones.



Fonte: PAL; PRAKASH, 2017.

Figura 21 – Função de Autocorrelação para o Índice Dow Jones.



Fonte: adaptado de PAL e PRAKASH (2017).

Análise de Sentimento

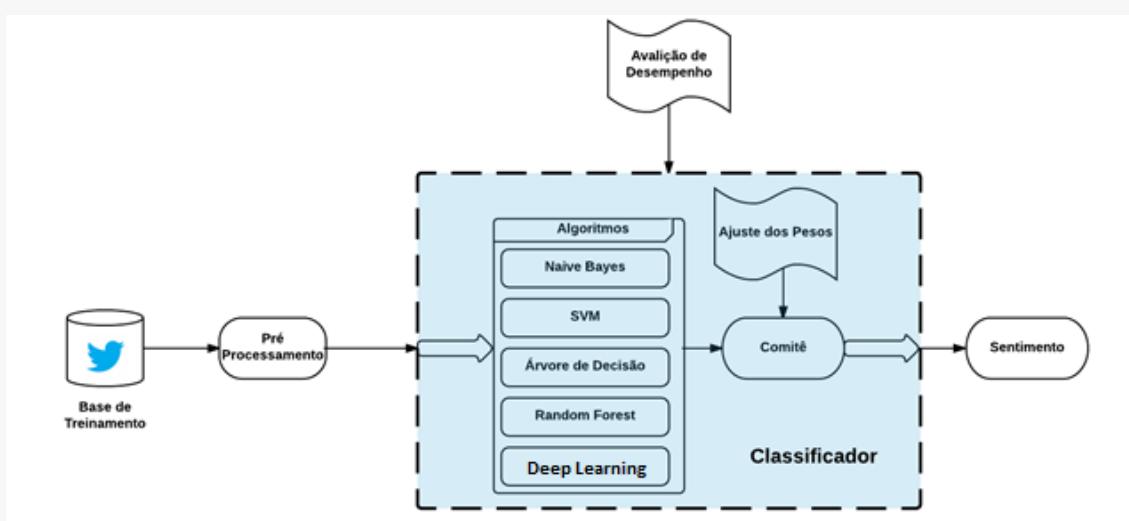
Análise de Sentimento consiste na mineração de textos a fim de identificar e extrair informações subjetivas que possam auxiliar no entendimento e classificação da opinião do usuário que redigiu um texto. Esse tipo de análise é realizado de maneira automática a fim de possibilitar classificar a opinião do usuário. Análise de sentimento é utilizada por empresas a fim de compreender a opinião que seus clientes possuem em relação à marca, um produto ou serviço oferecido. Após essa análise, é possível que a empresa adeque os produtos e serviços oferecidos ao gosto dos clientes. Assim, pode existir um aumento das vendas e, consequentemente, melhoria dos lucros da empresa.

Análise de sentimento é um campo do Processamento da Linguagem Natural que visa identificar e extrair a opinião de usuários. As opiniões são expressões subjetivas de sentimentos das pessoas. Para análise dessa subjetividade, é necessário transformar esses textos em uma linguagem que os computadores possam identificar a semântica (sentido) da opinião. A Figura 22 apresenta o pipeline para a realização da análise de

sentimento. Algumas técnicas de transformação de texto para análise de sentimento são:

- Tokenização;
- Remoção de StopWords;
- Stemização;
- Lematização.

Figura 22 – Exemplo de pipeline para a análise de sentimento.



Fonte: adaptado de AGUIAR et al. (2018).

Tokenização

O processo de tokenização consiste em separar o texto em “tokens”. Tokens podem ser vistos como as palavras ou sentenças que compõem o texto. Nesse sentido, existe a tokenização de sentenças e de palavras. Na tokenização de sentenças o texto é dividido em sentenças. Já na tokenização de palavras, as sentenças são divididas em palavras. A Figura 23 apresenta um exemplo desses dois processos.

Figura 23 – Exemplo dos processos de tokenização.

```
[57] #gerando os tokens de sentenças
sample_text=""" O menino gosta de jogar futebol aos finais de semana.
Ele gosta de jogar com seus amigos Marcos e João, mas não gosta de brincar
com a irmã Marcela"""
tokenizacao_sentencas=nltk.sent_tokenize
sample_sentence = tokenizacao_sentencas(text=sample_text)
pprint(sample_sentence)

[58] [' O menino gosta de jogar futebol aos finais de semana.',
      'Ele gosta de jogar com seus amigos Marcos e João, mas não gosta de brincar\n'
      'com a irmã Marcela']

[60] #tokenização de palavras
sample_sentence='O menino gosta de jogar futebol aos finais de semana.'
tokenizacao_palavras=nltk.word_tokenize
sample_words = tokenizacao_palavras(text=sample_sentence)
pprint(sample_words)

[61] ['O',
      'menino',
      'gosta',
      'de',
      'jogar',
      'futebol',
      'aos',
      'finais',
      'de',
      'semana',
      '.']
```

Remoção de Stopwords

StopWords são palavras que não possuem, ou possuem pouca, validade semântica no texto. A remoção dessas palavras auxilia na redução da quantidade de dados a serem tratados e facilita o reconhecimento de padrões pelos algoritmos empregados. A Figura 24 apresenta uma lista com algumas stopwords em português.

Figura 24 – Exemplo de stopwords existentes em Python (nltk).

```
[8] stopWordPortugues = nltk.corpus.stopwords.words('portuguese')
print(np.transpose(stopWordPortugues))

→ ['de' 'a' 'o' 'que' 'e' 'do' 'da' 'em' 'um' 'para' 'com' 'não' 'uma' 'os'
 'no' 'se' 'na' 'por' 'mais' 'as' 'dos' 'como' 'mas' 'ao' 'ele' 'das' 'à'
 'seu' 'sua' 'ou' 'quando' 'muito' 'nos' 'já' 'eu' 'também' 'só' 'pelo'
 'pela' 'até' 'isso' 'ela' 'entre' 'depois' 'sem' 'mesmo' 'aos' 'seus'
 'quem' 'nas' 'me' 'esse' 'eles' 'você' 'essa' 'num' 'nem' 'suas' 'meu'
 'ás' 'minha' 'numa' 'pelos' 'elas' 'qual' 'nós' 'lhe' 'deles' 'essas'
 'esses' 'pelas' 'este' 'dele' 'tu' 'te' 'vocês' 'vos' 'lhes' 'meus'
 'minhas' 'teu' 'tua' 'teus' 'tuas' 'nosso' 'nossa' 'nossos' 'nossas'
 'dela' 'delas' 'esta' 'estes' 'estas' 'aquele' 'aquela' 'aqueles'
 'aqueelas' 'isto' 'aquilo' 'estou' 'está' 'estamos' 'estão' 'estive'
 'estive' 'estivemos' 'estiveram' 'estava' 'estávamos' 'estavam'
 'estivera' 'estivéramos' 'esteja' 'estejamos' 'estejam' 'estivesse'
 'estivéssemos' 'estivessem' 'estiver' 'estivermos' 'estiverem' 'hei' 'há'
 'havemos' 'hão' 'houve' 'houvemos' 'houveram' 'houvera' 'houvéramos'
 'haja' 'hajamos' 'hajam' 'houvesse' 'houvéssemos' 'houvessem' 'houver'
 'houvermos' 'houverem' 'houverei' 'houverá' 'houveremos' 'houverão'
 'houveria' 'houveríamos' 'houveriam' 'sou' 'somos' 'são' 'era' 'éramos'
 'eram' 'fui' 'foi' 'fomos' 'foram' 'fora' 'fôramos' 'seja' 'sejamos'
 'sejam' 'fosse' 'fôssemos' 'fossem' 'for' 'formos' 'forem' 'serei' 'será'
 'seremos' 'serão' 'seria' 'seríamos' 'seriam' 'tenho' 'tem' 'temos' 'tém'
 'tinha' 'tínhamos' 'tinham' 'tive' 'teve' 'tivemos' 'tiveram' 'tivera'
 'tivéramos' 'tenha' 'tenhamos' 'tenham' 'tivesse' 'tivéssemos' 'tivessem'
 'tiver' 'tivermos' 'tiverem' 'terei' 'terá' 'teremos' 'terão' 'teria'
 'teríamos' 'teriam']
```

Stemização

O processo de stemização (stemming) consiste em reduzir a palavra a menor forma que mantêm o sentido, ou seja, ao seu tronco (stem). O tronco da palavra não é, necessariamente, o mesmo que o morfema da palavra e não precisar corresponder a outra palavra que existe no dicionário. A Figura 25 apresenta o processo de stemização pra algumas palavras em português utilizando a biblioteca NLTK.

Figura 25 – Exemplo de processo de Stemização em Python (nltk)

```
[49] ss = SnowballStemmer("portuguese")
print(ss.stem('casado'))
print(ss.stem('casarão'))
print(ss.stem('casa'))

→ cas
cas
cas
```

Lematização

O processo de lematização é similar ao realizado pela stemização. Entretanto, ele consiste em encontrar uma palavra que represente o sentido de um conjunto de palavras (Lema). A palavra encontrada deve existir no dicionário. Por exemplo, o conjunto de palavras {casarão, casinha, casebre} possui o lema casa, pois ela representa o sentido representado por esse conjunto. A Figura 26 mostra um exemplo de tokenização, seguido da análise do discurso (elementos gramaticais) e a lematização de uma sentença utilizando a biblioteca scapy.

Figura 26 – Exemplo de processo de lematização em Python (scapy)

```
#processo de lematização em uma sentença
import spacy
from spacy.lang.pt.examples import sentences

nlp = spacy.load('pt')      # carrega o idioma português
doc = nlp(sentences[0])    #carrega a primeira sentença existente no corpo de exemplo
print(doc)                 #imprime a sentença completa
print("Token\tPOS\tLemma")
for token in doc:           #encontra os token, POS (parts of speech).
    print(token.text, token.pos_, token.lemma_)
```

Apple está querendo comprar uma startup do Reino Unido por 100 milhões de dólares

Token	POS	Lemma
Apple	PROPN	Apple
está	AUX	estar
querendo	VERB	querer
comprar	VERB	comprar
uma	DET	umar
startup	NOUN	startup
do	DET	do
Reino	PROPN	Reino
Unido	PROPN	Unido
por	ADP	por
100	NUM	100
milhões	SYM	milhão
de	ADP	de
dólares	NOUN	dólar

Como pode ser visto, para a realização do processo de lematização é necessário realizar, anteriormente, o processo de tokenização. Após aplicar a lematização, pode ser visto, por exemplo, que a palavra “está” possui como lema a palavra “estar” e a palavra “querendo” possui como lema a palavra “querer”. Esses processos de pré-processamento são essenciais para que a análise de sentimento ocorra de maneira adequada.

Processamento do Texto

Após o pré-processamento do texto, é necessário que sejam aplicadas algumas técnicas para identificar o “sentimento” (opinião) do usuário que está expresso no texto. Essas técnicas são necessárias, pois os algoritmos que, normalmente, utilizamos para realizar a classificação da opinião dos usuários não foram desenvolvidos para tratarem textos. Assim, o texto deve ser representado de tal forma a ser corretamente interpretado pelos algoritmos. As principais técnicas utilizadas são:

- Bag of words;
- Word2vec.

Bag of Words

Na representação Bag of Words (BoW), os textos (frase ou um documento) são representados como um conjunto (bolsa) de palavras. Esse conjunto desconsidera a gramática e a ordem em que as palavras aparecem no texto, conservando apenas a multiplicidade dessas palavras, ou seja, o número de vezes que aparecem. Basicamente, esse procedimento pode ser dividido em 2 passos.

1. Encontrar o vocabulário que representa o documento.
2. Utilizar o vocabulário para converter as palavras das sentenças em um vetor de frequência de ocorrência das palavras no texto.

Para o exemplo, considere as seguintes sentenças que devem formar o nosso texto.

- O IGTI oferece especialização em Deep Learning.
- Deep Learning é utilizado em diversas aplicações.
- As aplicações de Deep Learning são estudadas nesta especialização.

O primeiro passo é encontrar o vocabulário. Para isso, é necessário o processo de tokenização e extração das stopwords. A Figura 27 mostra esses processos.

Figura 27 – Primeira etapa de construção do BoW

```
[86] sentenca="O IGTI oferece especializacao em Deep Learning. Deep Learning e utilizado em diversas
[87] #coloca toda a sentenca em lowercase
      sentenca=sentenca.lower()
[88] print(sentenca)
[89] #tokenização de sentenças
      tokenizacao_sentencas=nltk.sent_tokenize
      sample_sentence = tokenizacao_sentencas(text=sentenca)
      pprint(sample_sentence)
[131] #tokeniza palavras
      def tokenizaPalavras(sentenca):
          sample_words = tokenizacao_palavras(text=sentenca)
          return sample_words
      #removendo stopwords e criando o BoW
      def removeStopWords(list_of_words):
          my_stop_words=['o','em','as','de','sao','nesta','.','e','a','na','do'] # cria a lista de stopwords
          list_cleaned=set(list_of_words)-set(my_stop_words)
          return list_cleaned
      my_Bow=removeStopWords(list_words)
[128] print(my_Bow)
[128] {'oferece', 'estudadas', 'diversas', 'utilizado', 'especializacao', 'learning', 'aplicacoes', 'deep', 'igti'}
```

Após essa etapa, é necessário criar o vetor que representa a sequência de palavras e a frequência com que elas aparecem na sentença. Considere a seguinte sentença como exemplo: “O IGTI oferece especialização em Deep Learning”. A Figura 28 mostra a representação dessa sentença utilizando a BoW criada.

Figura 28 – Segunda etapa de construção do BoW

```
[128] print(my_Bow)
[129] ['oferece', 'estudadas', 'diversas', 'utilizado', 'especializacao', 'learning', 'aplicacoes', 'deep', 'igti']

[134] #Cria o vetor que representa a sentenca na BoW
      def bagofwords(sentence, words):
          sentence_words = extract_words(sentence)
          # conta a frequencia de palavras que estao no vetor do BoW
          bag = np.zeros(len(words))
          for sw in sentence_words:
              for i,word in enumerate(words):
                  if word == sw:
                      bag[i] += 1
          return np.array(bag)
      sentenca_teste='o  igti oferece especializacao em deep learning'
      print(bagofwords(sentenca_teste,my_Bow))

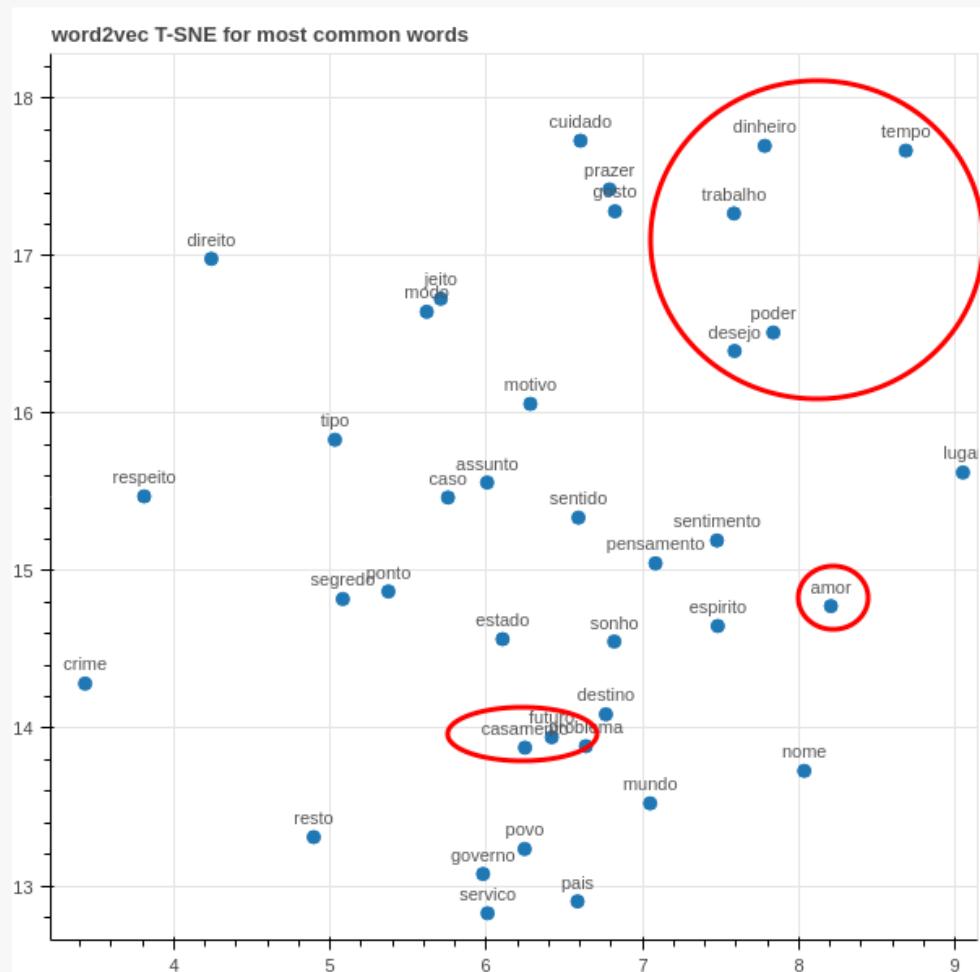
[135] [1. 0. 0. 0. 1. 1. 0. 1. 1.]
```

Word2vec

Word2vec é uma das mais populares técnicas para aprendizado de word embeddings. Word embeddings, assim como a Bag of Words, são representações de vocabulários de documentos. A maior vantagem dessa técnica, quando comparada com o Bag of Words, é que ela permite capturar a semântica, a similaridade e a relação entre as palavras no texto. Com a utilização do word2vec é possível representar as palavras como vetores.

É possível realizar operações com as palavras da mesma forma que realizamos com vetores quando as representamos em um espaço vetorial. A Figura 29 apresenta um exemplo desse espaço gerado por um conjunto de palavras em português.

Figura 29 – Representação de palavras com word2vec.



Fonte: FACURE, 2019.

Pela análise da Figura 29, podemos verificar que as palavras “desejo”, “poder” e “dinheiro” apresentam características semânticas parecidas. Também podemos ver que as palavras “crime” e “amor” possuem sentidos muito diferentes. Logo, por meio dessa técnica, é possível representar todo o texto como um conjunto de palavras em um espaço vetorial. De posse dessa transformação, podemos aplicar os algoritmos de aprendizado de máquina para realizar a classificação semântica dos textos. É por meio dessa classificação que é possível realizar a análise de sentimento dos textos. Assim, pode ser realizada uma avaliação sobre a opinião do usuário e identificar as preferências dos clientes.

Processamento de Imagens (Visão Computacional)

O processamento de imagens por meio da utilização de técnicas de aprendizado de máquina, em especial o Deep learning, proporciona novos horizontes para a geração e processamento automatizado de imagens. Atualmente, os computadores não são apenas capazes de, automaticamente, classificarem fotos. Eles podem identificar os elementos (pessoas, carros, objetos etc.) que estão presentes nas imagens, contar esses elementos e gerar uma descrição sobre essa imagem. Todas essas possibilidades foram propiciadas pelo desenvolvimento e avanço das técnicas de Machine Learning, em especial, pelo crescimento do Deep Learning.

Existe um grande número de bibliotecas e algoritmos que possibilitam a manipulação de imagens. Dentre elas, provavelmente, a mais conhecida seja a *Open Source Computer Vision Library* (OpenCV). Como o nome sugere, ela é uma biblioteca de código aberto que pode ser utilizada em várias linguagens como C++, Java e até MATLAB. OpenCV foi desenvolvido para poder realizar várias operações com imagens (carregar, mostrar, redimensionar etc.). Mas, após a união com o Deep Learning, novas oportunidades surgiram para o OpenCV. Com a união dessas tecnologias tornou-se possível, por exemplo, a detecção de objetos em imagens.

O OpenCV utiliza o HaaR Cascade para detecção de objetos nas imagens. O HaaR Cascade é um classificador que utiliza do Deep Learning para identificar diferentes tipos de objetos. O seu treinamento é conduzido por meio de imagens positivas de um objeto específico e imagens negativas contendo o objeto e um conjunto de outros elementos. Após esse treinamento, o HaaR consegue identificar esses objetos dentre um conjunto de objetos na imagem.

Com a utilização do OpenCV, realizar a identificação de rostos em fotografias, torna-se uma tarefa muito menos trabalhosa. A Figura 30

apresenta um exemplo de programa construído para a detecção de imagens em fotografias utilizando o OpenCV e o HaaR Cascade.

Figura 30 – Utilização da biblioteca OpenCV para detecção de rostos em fotos.

```
[135] import cv2 # biblioteca openCV

[147] # Criação do objeto CascadeClassifier
face_detection = cv2.CascadeClassifier("haarcascade_frontalface_default.xml") #arquivo xml que contém o treinamento do Haar

[148] # Carregando a imagem
img = cv2.imread("CAPA2.jpg")

[149] # Lendo a imagem em uma escala de cinza
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

[150] # procurando pelas coordenadas da imagem
faces = face_detection.detectMultiScale(gray_img, scaleFactor = 1.05, minNeighbors=5)

[160] from google.colab.patches import cv2_imshow # biblioteca necessária para o google colab mostrar a imagem
import matplotlib.pyplot as plt # mostrar a imagem no tamanho desejado
for x,y,w,h in faces: #identifica as coordenadas da foto e desenha o retângulo
    img = cv2.rectangle(img, (x,y), (x+w,y+h),(0,255,0),3)

plt.figure(figsize=(20,10))
cv2_imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Sistemas de Recomendação

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

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 baseando 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 31, nela o usuário A assistiu e gostou dos filmes Central do Brasil, Cidade de Deus e Alto da Comadecida. O usuário B assistiu e gostou dos filmes Central do Brasil e Alto da Comadecida. Assim, utilizando o filtro colaborativo, é possível dizer que os usuários A e B são similares. Como o usuário B não viu o filme Cidade de Deus que se encontra no catálogo de filmes que o usuário A viu e gostou, esse filme será recomendado para o usuário B.

Figura 31 – 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 usuários que itens no catálogo de produtos, a variação dos gostos dos usuários e a possibilidade de ele manipular, deliberadamente, o sistema de recomendação. Como a quantidade de usuário é, geralmente, maior que a de itens, encontrar a similaridade entre cada um é uma tarefa computacionalmente mais cara. Como as preferências dos usuários variam com o tempo, a similaridade entre grande parte deles 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 essas pessoas podem criar perfis falsos e manipularem 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 32 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 32, 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 32 – Filtro colaborativo baseado nos itens



Chatbot

Segundo dados da Gartner, mais de 85% das interações entre consumidores e serviços é realizada por meio de chatbots. Os chatbots são compostos por um sistema baseado em inteligência artificial que tenta imitar uma conversa entre seres humanos. Esses sistemas são capazes de interagir com as pessoas por meio da utilização dos conceitos de processamento da linguagem natural e do aprendizado profundo.

O primeiro chatbot que se tem notícia é o Eliza. Ele foi desenvolvido em 1966 por Joseph Weizenbaum nos laboratórios do Massachusetts Institute of Technology. O chatbot Eliza simula a conversa entre um terapeuta e seu paciente. Como não emprega nenhum dos algoritmos que conhecemos e possui, aproximadamente, 200 linhas de código, o Eliza não é capaz de manter conversas fluidas com os seres humanos. Você pode conversar com Eliza através deste [link](#).

Existem duas principais variantes de abordagens para construção dos chatbots: Baseada em Regras (Rule-Based) e Autoaprendizado (Self Learning). Na abordagem baseada em regras, os chatbots respondem as

questões baseando-se em regras previamente estabelecidas. Nesse tipo de estratégia, os chatbots conseguem lidar com questões simples, mas não tem a capacidade de tratar ou manter conversas mais complexas. O chatbot Eliza utiliza esse tipo de estratégia.

Já na abordagem de autoaprendizado são utilizadas as técnicas de aprendizado de máquina que possibilitam ao chatbot “aprender” à medida que vai conversando com o usuário. A estratégia de autoaprendizado pode ocorrer de duas formas: Baseada em Recuperação ou Generativa. Quando é utilizado método baseado na recuperação são empregadas algumas heurísticas para selecionar a resposta a uma pergunta dentre um conjunto de respostas predefinidas que estão armazenadas em uma biblioteca de respostas. São utilizados o contexto e a mensagem recebida para identificar a resposta mais adequada. No modelo generativo, as respostas são criadas no momento em que o chatbot recebe uma pergunta do usuário. Esse tipo de modelo é mais “inteligente” e simula de maneira mais satisfatória o comportamento humano. A Figura 33 apresenta um pequeno programa que implementa um chatbot. O chatbot vai “aprendendo” à medida que a interação com o usuário vai acontecendo. Esse aprendizado é lento, mas após alguns minutos de conversa ele passa a aprender e interagir de maneira mais satisfatória.

Figura 33 – Criação de um simples chatbots.

```
[15] from chatterbot.trainers import ListTrainer #ListTrainer responsável por permitir que
                                             #uma lista de strings seja utilizada no processo de aprendizagem do Bot.
from chatterbot import ChatBot           #classe para criação do chatbot

[16] #criação das perguntas/respostas para o bot
bot = ChatBot('ChatBot IGTI')          #objeto Chatbot
conversa = ['Olá', 'Olá', 'Tudo bem?', 'Tudo ótimo', 'Você gosta de programar?', 'Sim, eu programo em Python']
#foram criadas 3 perguntas/respostas. O segundo item da lista corresponde a resposta do item anterior

trainer = ListTrainer(bot) #indica que o treinamento será através da lista de perguntas/respostas
trainer.train(conversa)    #realiza o treinamento através das perguntas e respostas

[17] [nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   /root/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
List Trainer: [#####] 100%
```

loop utilizado para capturar a interação com o usuário

```
while True:
    pergunta = input("Usuário: ")
    resposta = bot.get_response(pergunta)
    if float(resposta.confidence) > 0.5: # verifica se existe alguma resposta armazenada com probabilidade
                                              #maior que 50% de ser a resposta à pergunta
        print('IGTI Bot: ', resposta)
    else:
        print('IGTI Bot: Ainda não sei responder esta pergunta')

...
```

Usuário: Olá
 IGTI Bot: Tudo bem?
 Usuário: Tudo ótimo.
 IGTI Bot: Você gosta de programar?
 Usuário:



> Capítulo 3



Capítulo 3. Algoritmos de Machine Learning

O objetivo deste capítulo é apresentar aos alunos uma visão geral sobre alguns algoritmos de Machine Learning. Ao final, espera-se que o aluno possa reconhecer e utilizar alguns dos principais algoritmos supervisionados e não supervisionados para resolver problemas reais. Os conceitos abordados são:

- Kmeans;
- KNN;
- SVM;
- Árvore de Decisão.

Relembrando alguns conceitos e estatística

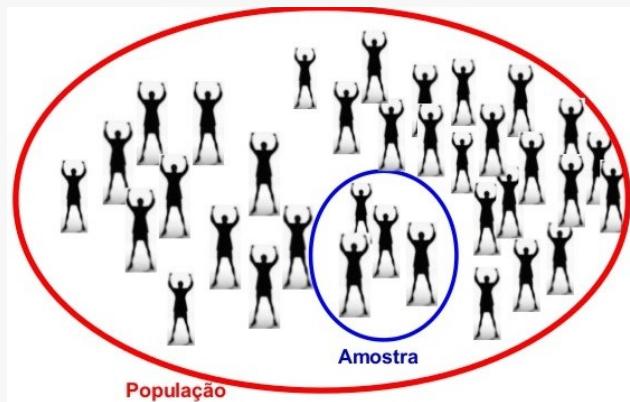
Antes de introduzir os temas principais deste capítulo, é necessário relembrar os conceitos fundamentais sobre estatística. Nesta primeira seção, será realizada uma rápida revisão destes. Não serão abordados os aspectos matemáticos de cada uma dessas definições, apenas as questões práticas serão abordadas.

População: conjunto de elementos que tem pelo menos uma característica em comum.

Amostra: subconjunto de elementos de uma população, que são representativos para estudar a característica de interesse da população.

A Figura 34 apresenta um exemplo que diferencia amostra de população.

Figura 34 – Diferença entre população e amostra.



Fonte: Ramos (2019).

Variável aleatória: é uma variável quantitativa, cujo resultado (valor) depende de fatores aleatórios. Um exemplo de uma variável aleatória é o resultado do lançamento de um dado que pode dar qualquer número entre 1 e 6.

Média aritmética: é a soma de vários valores dividido pelo total deles. Ou seja, o resultado dessa divisão equivale a um valor médio entre todos os valores.

Medidas de dispersão: estratégias para verificar se os valores apresentados em um conjunto de dados estão dispersos ou não e o quanto distantes um do outro eles podem estar.

Variância: é uma medida de dispersão em que, dado um conjunto de dados, ela mostra o quanto distante cada valor desse conjunto está do valor central (médio). Quanto menor é a variância, mais próximos os valores estão da média; mas quanto maior ela é, mais os valores estão distantes da média (ESCOLA, 2019). A Equação 3 apresenta a fórmula para o cálculo da variância amostral.

- Equação 3:

$$\text{Var}_{\text{amostral}} (\sigma^2 = \sigma_{ii}) = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2}{n-1}$$

Desvio padrão: também é uma medida de dispersão que é capaz de identificar o “erro” em um conjunto de dados, caso quiséssemos substituir um dos valores coletados pela média aritmética. O desvio padrão aparece junto à média aritmética, informando o quanto “confiável” é esse valor. A Equação 4 mostra o cálculo do desvio padrão através a variância.

- Equação 4:

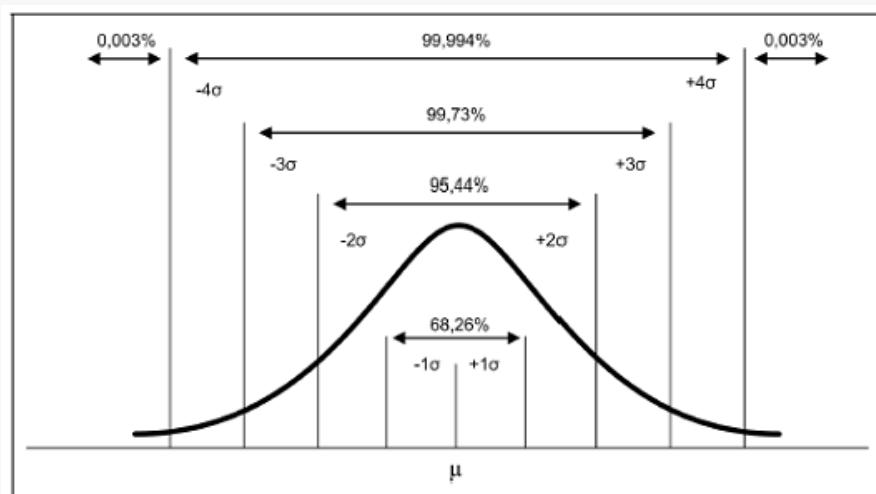
$$\text{Desvio padrão } (\sigma) = \sqrt{\text{Variancia}_{amostral}}$$

Moda: é uma medida de tendência central que corresponde ao dado (valor) mais frequente em conjunto. Por exemplo, seja o conjunto composto pelos números $A = \{1, 2, 2, 2, 3, 4, 5\}$. A moda desse conjunto é o valor 2.

Mediana: também é uma medida de tendência central que, se o conjunto de informações for numérico e estiver organizado em ordem crescente ou decrescente, a sua mediana será o número que ocupa a posição central da lista. Por exemplo, seja o conjunto $A = \{32, 33, 24, 31, 44, 65, 32, 21, 32\}$, ordenando esse conjunto temos $A = \{21, 24, 31, 32, 32, 33, 44, 65\}$. Assim, a mediana nesse conjunto será 32. Se o conjunto possuir um número par de itens, o valor da mediana corresponde à média aritmética dos 2 valores centrais.

Distribuição normal: é a distribuição dos dados mais utilizada em estatística. Ela também é conhecida como distribuição gaussiana. Para determinar o formato dessa distribuição, é necessário apenas informar a média (μ) e o desvio padrão (σ). A Figura 35 apresenta o formato dessa distribuição.

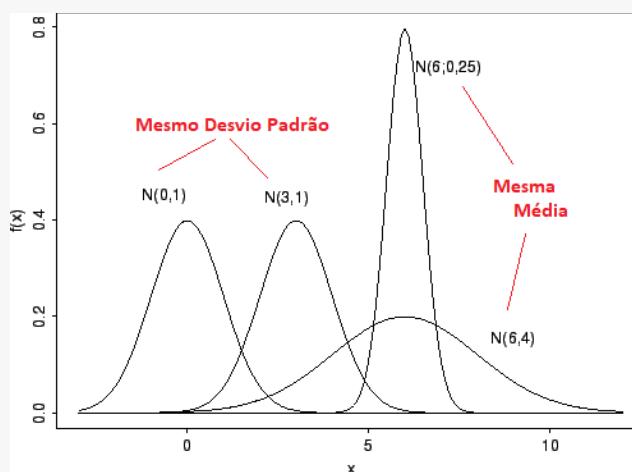
Figura 35 – Distribuição normal.



Fonte: ACTION (2017).

A distribuição normal é simétrica em torno da média, ou seja, a média, a mediana e a moda são todas coincidentes. A Figura 36 mostra alguns exemplos de distribuições normais com diferentes médias e desvio padrão. A área sob a curva normal (na verdade, abaixo de qualquer função de densidade de probabilidade) é 1. Então, para quaisquer dois valores específicos podemos determinar a proporção de área sob a curva entre esses dois valores.

Figura 36 – Diferentes distribuições normais.



Fonte: adaptado de ACTION (2017).

Correlação entre variáveis

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 5 e 6 apresentam diferentes fórmulas utilizadas para o cálculo do coeficiente de correlação.

- Equação 5:

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

- Equação 6:

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

Em que $-1 \leq r_{ij} \leq 1$, $i, j = 1, 2, \dots, n$. 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 3 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.

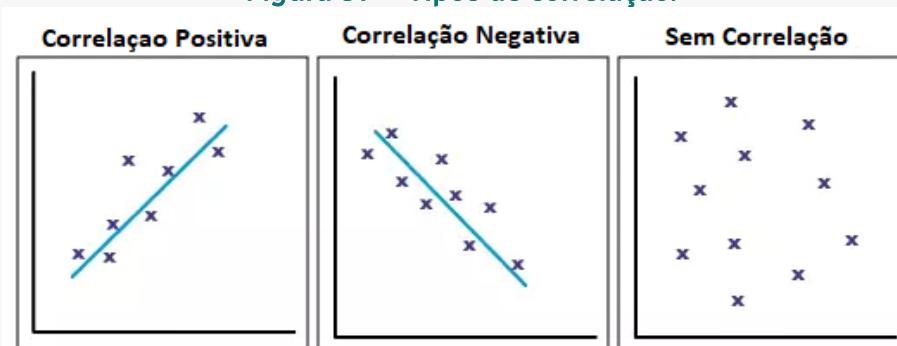
Correlação negativa: indica que o relacionamento linear entre duas variáveis ocorre em direções opostas, ou seja, caso uma variável seja aumentada a outra deve diminuir e vice-versa. Por exemplo, um aumento da

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.

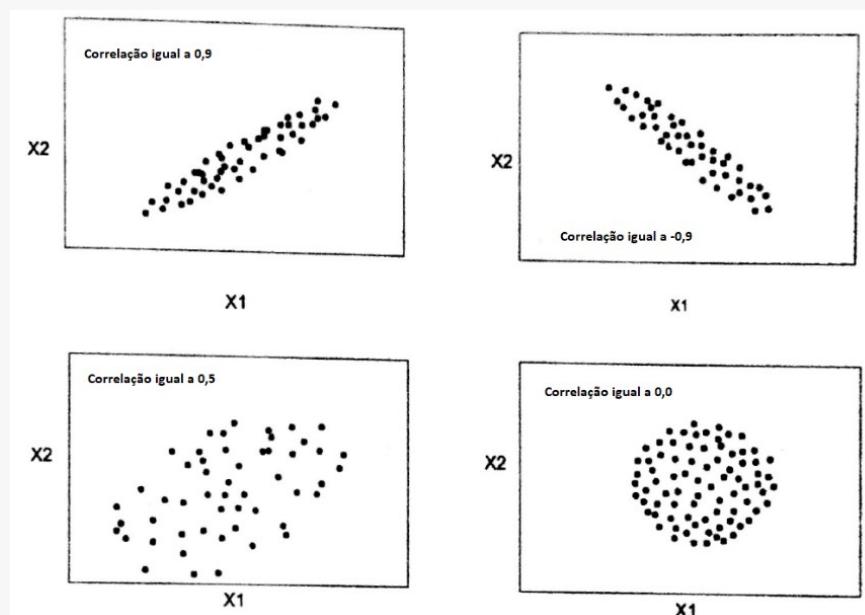
A Figura 37 mostra exemplos gráficos desses casos relatados.

Figura 37 – Tipos de correlação.



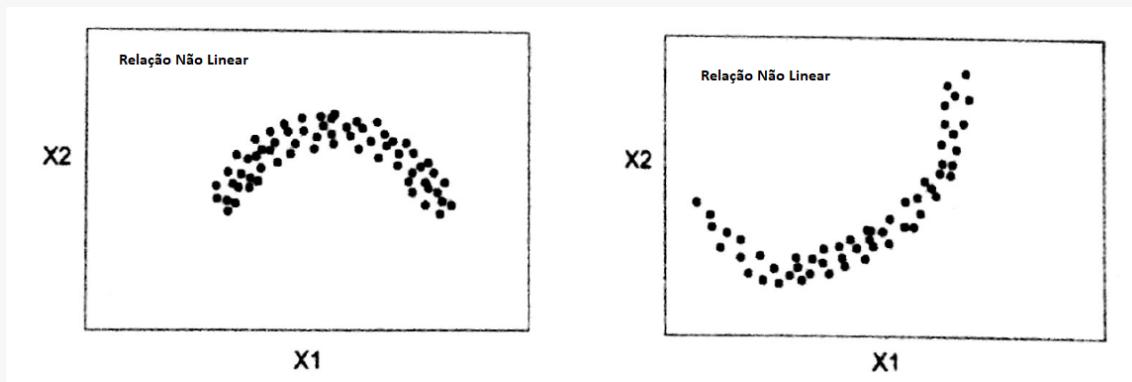
É importante ressaltar que o coeficiente de correlação apenas expressa a correlação linear entre variáveis. As Figuras 38 e 39 apresentam valores de correlação para alguns gráficos de dispersão.

Figura 38 – Diferentes valores de coeficientes de correlação.



Fonte: Mingoti (2005).

Figura 39 – Dados não linearmente relacionados.



Fonte: Mingoti (2005).

Exemplo de aplicação e análise de correlação

Agora vamos aplicar os conceitos aprendidos para determinar o relacionamento entre algumas variáveis de interesse. Para isso, será utilizado o banco de dados Auto MPG Data Set, da Universidade da Califórnia, que apresenta uma pesquisa da eficiência energética de automóveis em 1983. Nesse exemplo, será utilizada a linguagem Python para encontrar a correlação entre as diferentes variáveis. Os códigos para a execução desse exemplo podem ser encontrados no link: <https://colab.research.google.com/drive/1NsqfMsfYLJiFGfVaD5QrOc5EpukUyPI>.

Inicialmente, é realizada a definição das bibliotecas. A Figura 40 apresenta as bibliotecas utilizadas para realizar a correlação entre variáveis.

Figura 40 – Bibliotecas utilizadas.

```
[10] #definição da biblioteca a ser utilizada
     import pandas as pd # biblioteca utilizada para trabalhar com bancos de dados
     import matplotlib.pyplot as plt
```

Posteriormente, é necessário ler o banco de dados que será utilizado para análise. A Figura 41 mostra como os dados são lidos através da biblioteca Pandas.

Figura 41 – Leitura do banco de dados

```
[3] #carrega o banco de dados
mpg_data = pd.read_csv('auto-mpg.data', delim_whitespace=True, header=None,
                      names = ['mpg', 'cylinders', 'displacement','horsepower',
                               'weight', 'acceleration', 'model_year', 'origin', 'name'],
                      na_values='?') #realiza o a leitura do banco de dados
```

A Figura 42 apresenta as 5 primeiras linhas desse conjunto de dados.

Figura 42 – Ilustração do banco de dados

```
[13] # realiza a visualização das primeiras 5 linhas do banco de dados
mpg_data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

Em todos os algoritmos de análise de dados é necessário compreender o banco de dados utilizado, bem como identificar quais os tipos de variáveis constituem esse conjunto de dados. Assim, é possível identificar dados que estejam faltando e separar dados numéricos de categóricos. A Figura 43 apresenta a função que utilizada para realizar essa análise.

Figura 43 – Análise superficial do banco de dados.

```
[4] mpg_data.info() #verifica se existem valores nulos e os tipos de variáveis
[4]:> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
mpg           398 non-null float64
cylinders     398 non-null int64
displacement  398 non-null float64
horsepower    392 non-null float64
weight         398 non-null float64
acceleration  398 non-null float64
model_year    398 non-null int64
origin         398 non-null int64
name           398 non-null object
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

Pelo resultado apresentado pela Figura 43 é possível verificar que existem dados que não são numéricos. Portanto, para realizar uma análise de correlação, nesse exemplo, as colunas “model_year” e “origin” serão excluídas.

Para encontrar a correlação utilizando o Pandas basta aplicar o comando “corr” sobre as colunas desejadas. A Figura 44 mostra a correlação apenas entre duas variáveis e a matriz de correlação entre cada um dos dados utilizando o método de “Spearman”.

Figura 44 – Análise de correlação

```
[5] mpg_data[['mpg']].corr(mpg_data['weight']) # função utilizada para encontrar a correlação entre variáveis
[5]:> -0.8317409332443352

[12] mpg_data.drop(['model_year', 'origin'], axis=1).corr(method='spearman') #retira as colunas model_year e origin e encontra a matriz de correlação
[12]:>
```

	mpg	cylinders	displacement	horsepower	weight	acceleration
mpg	1.000000	-0.821864	-0.855692	-0.853616	-0.874947	0.438677
cylinders	-0.821864	1.000000	0.911876	0.816188	0.873314	-0.474189
displacement	-0.855692	0.911876	1.000000	0.876171	0.945986	-0.496512
horsepower	-0.853616	0.816188	0.876171	1.000000	0.878819	-0.658142
weight	-0.874947	0.873314	0.945986	0.878819	1.000000	-0.404550
acceleration	0.438677	-0.474189	-0.496512	-0.658142	-0.404550	1.000000

O método de Spearman é utilizado para medir o relacionamento monotônico de duas variáveis. Ao invés de realizar o relacionamento linear,

como na correlação de Pearson (Equação 5), o método de Spearman verifica se as variáveis em questão estão variando juntas. Isso não implica que as variáveis devem ser modificadas a taxas constantes, como ocorre na relação linear.

A Figura 45 apresenta a matriz de correlação utilizando o coeficiente de Pearson (Equação 5). Comparando as Figuras 44 e 45 é possível ver que os coeficientes encontrados para cada um dos métodos apresentam valores diferentes. Assim, é interessante realizar uma análise gráfica sobre os dados. Para isso, é utilizado o gráfico de dispersão presente na Figura 46.

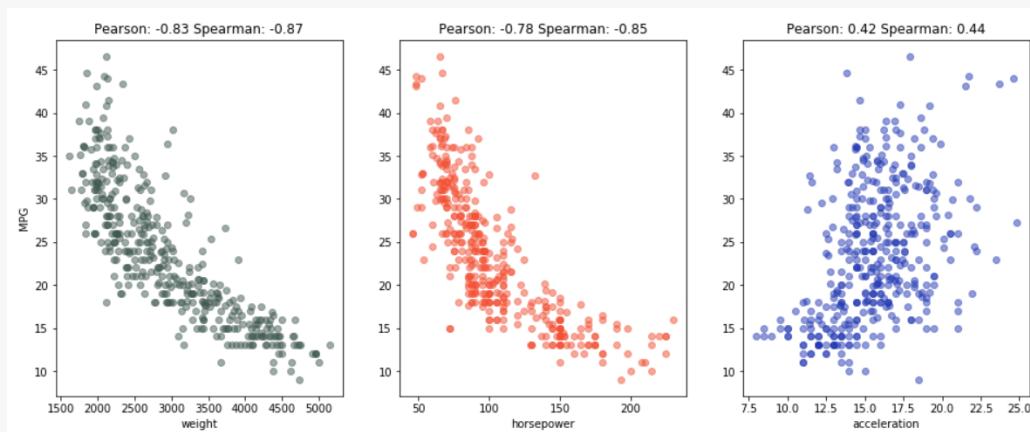
Figura 45 – Análise de correlação de Pearson

```
[11] mpg_data.drop(['model_year', 'origin'], axis=1).corr(method='pearson').style.format("{:.2}").background_gradient(cmap=plt.get_cmap('coolwarm'), axis=1)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration
mpg	1.0	-0.78	-0.8	-0.78	-0.83	0.42
cylinders	-0.78	1.0	0.95	0.84	0.9	-0.51
displacement	-0.8	0.95	1.0	0.9	0.93	-0.54
horsepower	-0.78	0.84	0.9	1.0	0.86	-0.69
weight	-0.83	0.9	0.93	0.86	1.0	-0.42
acceleration	0.42	-0.51	-0.54	-0.69	-0.42	1.0

Analizando a Figura 46, é possível verificar que a relação entre MPG e horsepower não é estritamente linear, mas estão altamente correlacionados. Isso explica o maior valor obtido pelo coeficiente de Spearman sobre o de Pearson.

Figura 46 – Gráficos de dispersão para análise de correlação de Pearson e Spearman.

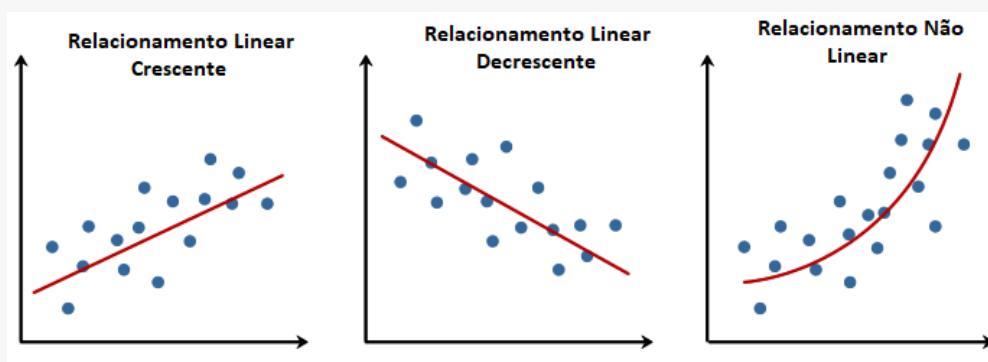


Análise de Regressão

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 certa quantidade que está a ser manipulada numa experiência. Já uma variável dependente representa certa 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 47 exibe alguns modelos de regressão.

Figura 47 – Exemplos de modelos de regressão.



Fonte: adaptado de Mingoti (2005).

Régressão Linear

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

afetam linearmente a variável dependente. A Equação 7 mostra o modelo de regressão linear.

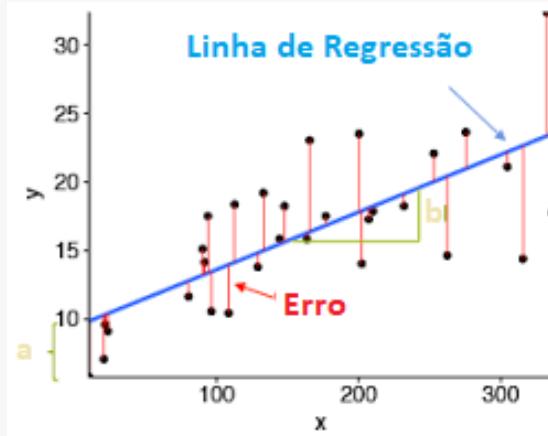
- Equação 7:

$$Y = a + bX$$

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 7 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 48 mostra o valor de erro (medida - prevista). O quadrado da soma de todos os erros (linhas verticais em vermelho) é a função a ser minimizada.

Figura 48 – Erro minimizado 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. O coeficiente de determinação expressa o quanto a variância da variável resposta (dependente) é explicada pela variável preditora (independente). A equação 8 apresenta essa relação.

- Equação 8:

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

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 7 apresenta os valores utilizados nesse problema.

Tabela 7 – 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 49 apresenta essa definição.

Figura 49 – 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 50 mostra como foram definidas as variáveis presentes na Tabela 7.

Figura 50 – 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 51 mostra como é realizada a definição da equação linear a ser otimizada.

Figura 51 – 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 52 mostra os valores iniciais aplicados aos parâmetros “a” e “b” da reta.

Figura 52 – 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 53 apresenta como deve ser realizado esse processo de otimização.

Figura 53 – 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 54 mostra esse procedimento.

Figura 54 – Cálculo 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 55 apresenta os cálculos realizados.

Figura 55 – Cálculo 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(parametrosOptimizados[0],parametrosOptimizados[1]))
```

Y = 1320.5325666669085X -6627.651716729711

Por último, podemos realizar o “plot” dessa equação e regressão. A Figura 56 apresenta o gráfico com os pontos utilizados e a reta que minimiza o erro médio quadrático.

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 deste [link](#).

Figura 56 – Gráfico de saída para o modelo de regressão

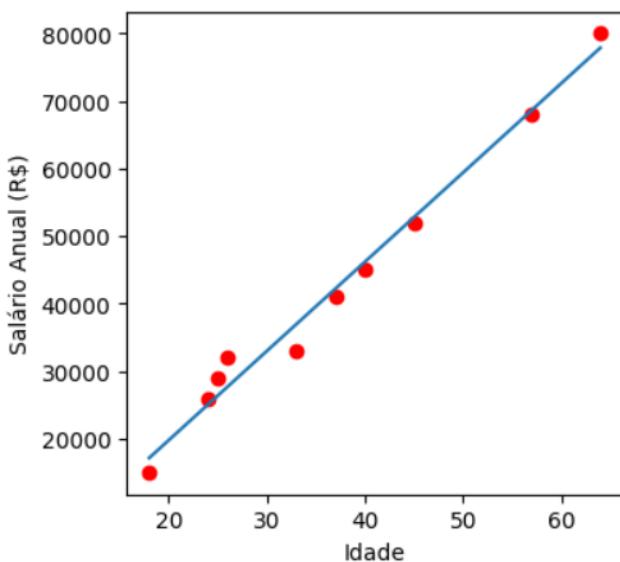
```
#realiza o plot da figura
f = plt.figure(figsize=(4, 4), dpi=100) #indica o tamanho da figura
axes = f.add_subplot(111) #cria os objetos para o subplot

# frealiza 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, *parametrosOptimizados) # 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$)')
```



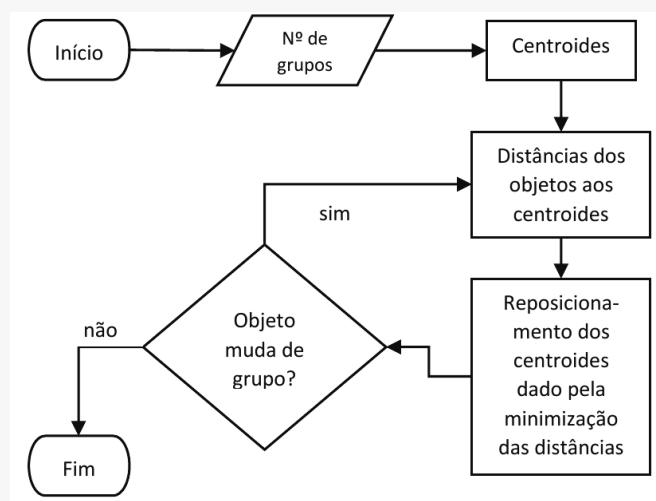
Nesta apostila, serão abordados os algoritmos supervisionados *k-Nearest Neighbors* (KNN), árvore de decisão e SVM (Support Vector Machine), além do algoritmo não supervisionado K-means. Algoritmos supervisionados são aqueles que utilizam um banco de dados que possui valores de entrada e saída. Esses dados são utilizados para realizar o treinamento do modelo. A partir do treinamento é possível que esses algoritmos consigam capturar as características do conjunto de dados e, quando novos dados forem apresentados, é possível realizar a previsão dos valores de saída. Já os algoritmos não supervisionados, não possuem a fase de treinamento, pois o banco de dados utilizado não apresenta o mapeamento entre dados de entradas e saídas. Esses algoritmos trabalham

com um banco de dados sem conhecer a saída provável. Normalmente, são utilizados em etapas anteriores à aplicação dos algoritmos supervisionados.

K-means

O algoritmo K-means é utilizado para encontrar grupos em um conjunto de dados. Para isso, o usuário deve definir o número de grupos que devem ser encontrados. Esse número 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 2 centroides a serem encontrados no conjunto de dados, ao final da execução do K-means, devem existir dois grupos diferentes de dados.

Figura 57 – 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. A Figura 58 apresenta o conjunto de dados desenvolvido.

Figura 58 – Conjunto 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 2 diferentes agrupamentos de dados nesse conjunto. A Figura 59 apresenta o procedimento para criar o objeto desse algoritmo e aplicá-lo.

Figura 59 – 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
centroides = kmeans.cluster_centers_  #encontra as coordenadas dos centroides
print(centroides)
```

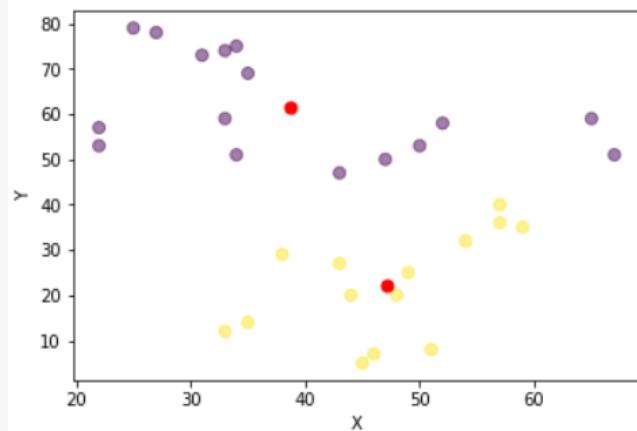
[38.75	61.625]
[47.07142857	22.14285714]]

A Figura 60 mostra o resultado final após a aplicação do agrupamento realizado pelo K-means. Os códigos para essa aplicação podem ser acessados através deste [link](#).

Figura 60 – 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')



KNN

O KNN é algoritmo supervisionado normalmente utilizado para realizar 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 do 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 ponto. A Figura 61 mostra o cálculo das distâncias entre cada um dos dados de um *dataset*.

Figura 61 – 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	?

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

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

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

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

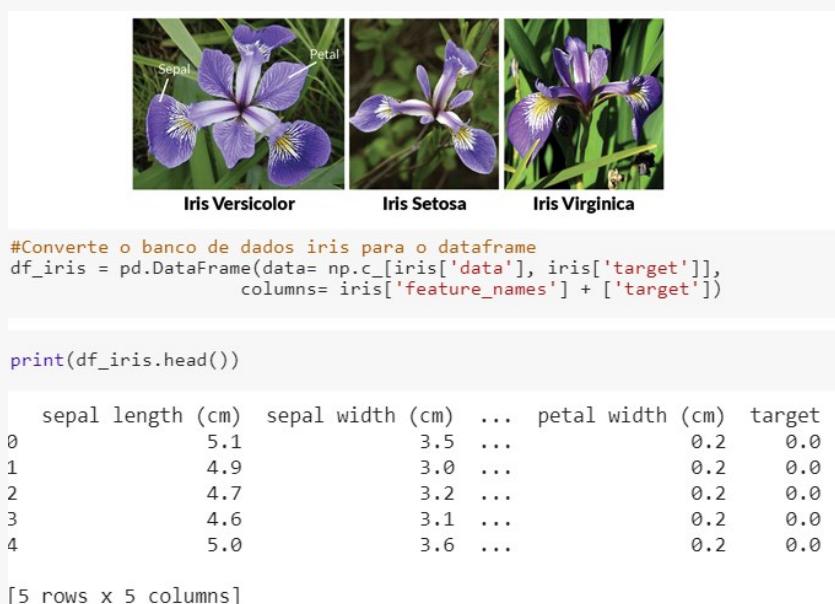
$$d_E(\#1, \#11) = \sqrt{56.25}$$

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

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 Iris (Setosa, Virginica e Versicolor). A Figura 62 exemplifica esse banco de dados.

Figura 62 – Exemplo do banco de dados Iris.



Como esse é um algoritmo supervisionado, é necessário dividir o banco de dados em instâncias para treinamento e teste. A Figura 63 apresenta esse procedimento.

Figura 63 – 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
```

Para que o algoritmo consiga efetuar a tarefa desejada, 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 64.

Figura 64 -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 efetuar a construção e treinamento do modelo. A Figura 65 apresenta esse processo utilizando a biblioteca sklearn. Para esse exemplo, foram escolhidos 5 vizinhos para realizar a classificação dos dados de teste do modelo.

Figura 65 – 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 mostrada na Figura 66.

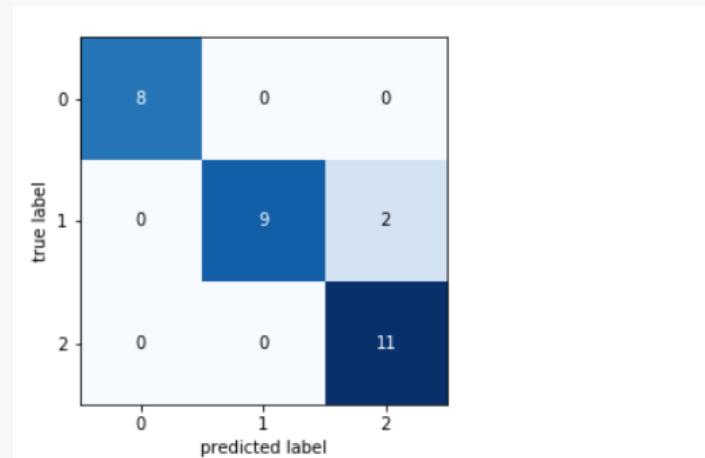
Figura 66 – Previsão realizada pelo modelo KNN treinado.

```
#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 67 apresenta a matriz de confusão para o KNN.

Figura 67 – 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 67, é possível perceber, por exemplo, que o KNN classificou corretamente 11 instâncias como

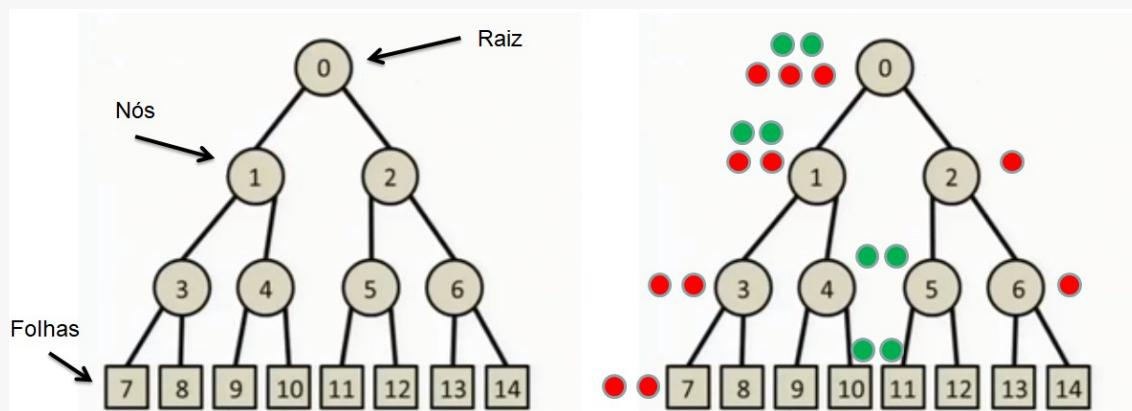
pertencentes à classe 2 e 2 instâncias foram erroneamente classificadas como pertencentes à classe 2, quando deveriam ter sido classificadas como pertencentes à classe 1. Portanto, a matriz de confusão apresenta-se como uma excelente métrica para comparar a classificação de modelos.

Os códigos para esse exemplo podem ser acessados através deste [link](#).

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 e 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 dele as instâncias são repassadas para os nós intermediários até que cheguem às folhas da árvore. A Figura 68 apresenta um exemplo de árvore.

Figura 68 – 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 o final do processo, nas folhas, o conjunto de dados pode ser separado.

Para o exemplo de utilização da árvore de decisão, também será empregado o banco de dados Iris. A Figura 69 apresenta como é criado e treinado o modelo de classificação utilizando árvores de decisão.

Figura 69 – 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)

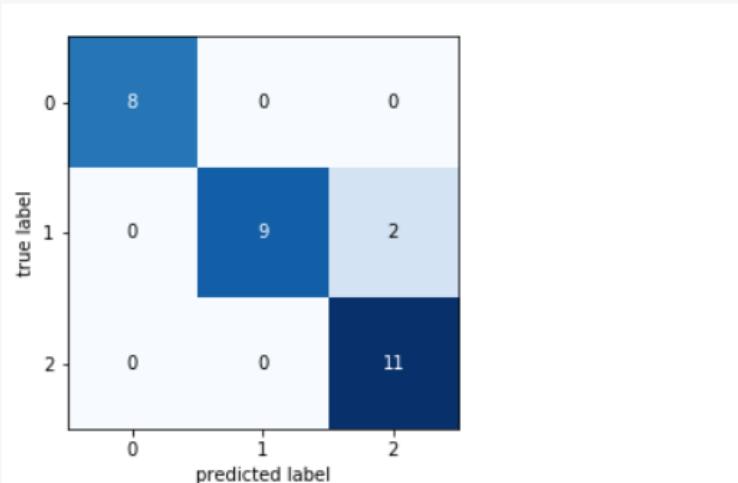
#Realiza a previsão de classificação
y_pred = clf.predict(X_test)
```

A Figura 70 mostra a matriz de confusão para a classificação do banco de dados Iris utilizando o algoritmo árvore de decisão.

Pela Figura 70 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.

Figura 70 – Matriz de confusão para o algoritmo árvore de decisão

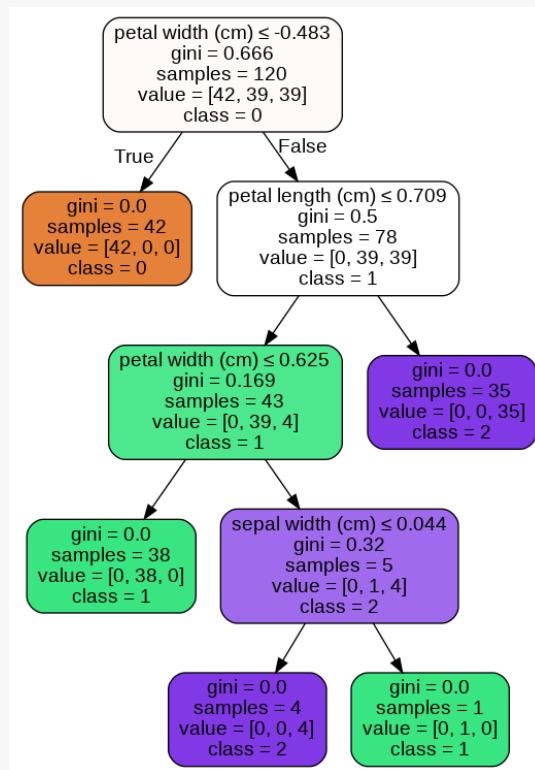
```
#Avaliando o modelo  
  
#realiza o plot da matriz de confusão  
matriz_confusao = confusion_matrix(y_test, y_pred)  
from mlxtend.plotting import plot_confusion_matrix  
  
fig, ax = plot_confusion_matrix(conf_mat=matriz_confusao)  
plt.show()
```



A Figura 71 apresenta todo o processo de decisão realizado pelo algoritmo até encontrar o resultado expresso na Figura 70.

Os códigos para essa aplicação podem ser encontrados através deste [link](#).

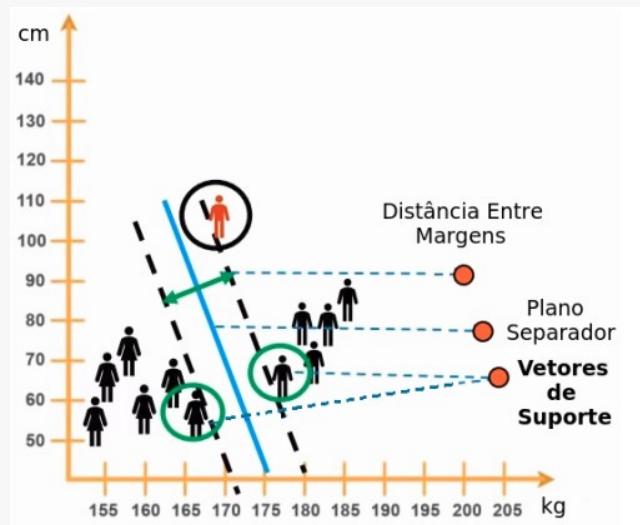
Figura 71 – Processo de decisão realizado pela árvore de decisão



SVM

Os *Support Vector Machines* (SVM) também pertencem à classe de algoritmos supervisionados. Esses algoritmos são utilizados, normalmente, para realizar a classificação de um conjunto de elementos. O princípio de funcionamento dele 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 72 mostra graficamente o hiperplano (para duas dimensões é uma reta) de separação.

Figura 72 – Separação realizada pelo SVM.



Para a aplicação do SVM, também é utilizado o *dataset Iris*. A Figura 73 mostra como deve ser construído, treinado e a previsão de classificação do algoritmo SVM, através da biblioteca sklearn.

Figura 73 – 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)
```

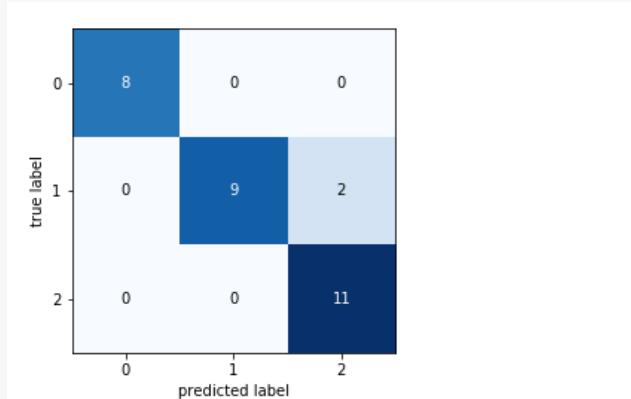
A Figura 74 apresenta a matriz de confusão para a aplicação do algoritmo SVM. Por meio dessa figura, é possível verificar que os resultados também foram similares aos demais algoritmos de classificação. O SVM, em bancos de dados mais complexos, normalmente possui melhor resultado que os outros algoritmos demonstrados nas seções anteriores. Os códigos para essa construção podem ser encontrados através deste [link](#).

Figura 74 – Matriz de confusão para o SVM.

```
#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()
```





> Capítulo 4



Capítulo 4. Capítulo 4. Redes Neurais Artificiais

Neste capítulo, serão abordados os conceitos fundamentais sobre redes neurais artificiais e Deep Learning. Ao final, espera-se que o aluno seja capaz de:

- Identificar as características das redes neurais artificiais;
- Identificar as diferenças entre Deep Learning e uma rede neural artificial sem Deep Learning;
- Aplicar o Deep Learning para resolver algumas classes de problemas.

Redes Neurais Artificiais e Deep Learning

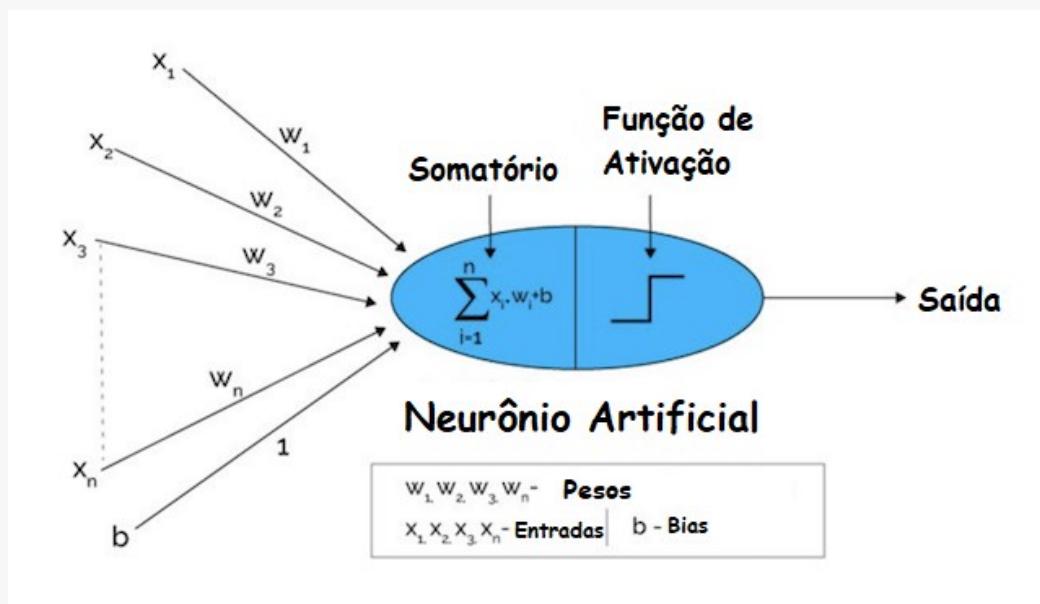
Redes neurais artificiais são sistemas de computação com nós interconectados inspirados no funcionamento do sistema nervoso central humano. Usando algoritmos, elas podem reconhecer padrões escondidos e correlações em dados brutos, agrupá-los, classificá-los e, com o tempo, aprender e melhorar continuamente.

A unidade fundamental de uma rede neural artificial é o perceptron. A Figura 75 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 que, normalmente, ocorre por meio de um processo conhecido como backpropagation.

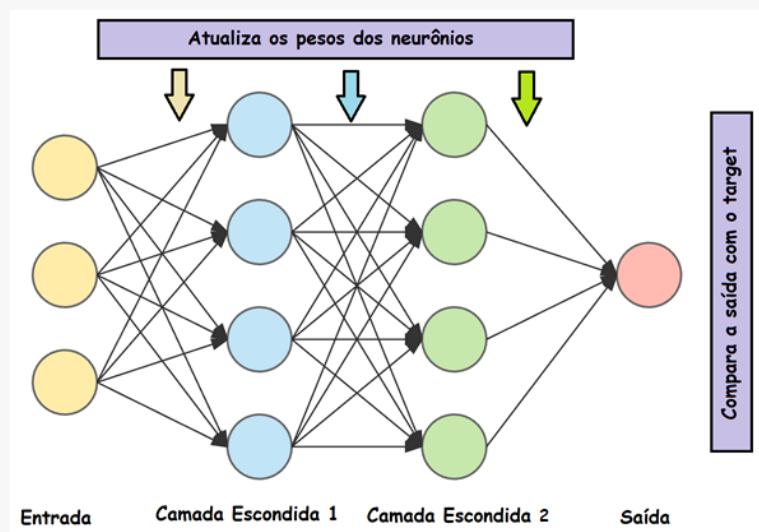
O backpropagation consiste em duas etapas. Na primeira etapa, os dados são apresentados à rede. Desse modo, ela realiza o processo do cálculo de todos os pesos e encontra uma saída estimada. Essa etapa é

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 75 – Exemplo de um perceptron



Quando vários perceptrons estão conectados, temos a construção de uma rede neural artificial. A Figura 76 apresenta um exemplo de rede Perceptron Multi-camadas (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 76 – 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, consequentemente, mais pesos para serem ajustados. Assim, é possível que a rede consiga “aprender” características mais complexas sobre o banco de dados. Entretanto, com isso, também a complexidade computacional para a treinamento do modelo será maior.

Para a construção a 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 77 apresenta a forma como a MLP é criado no sklearn.

Figura 77 – 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 78 mostra como ele é realizado através do Sklearn.

Figura 78 – Procedimento para realizar o treinamento da rede MLP

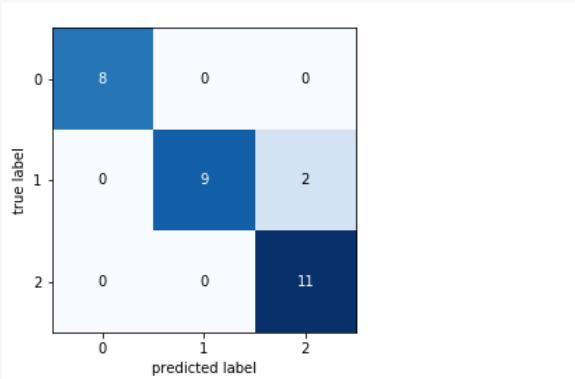
```
#realiza o fit do modelo
clf.fit(X_train,y_train)

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 79.

Figura 79 – 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()
```

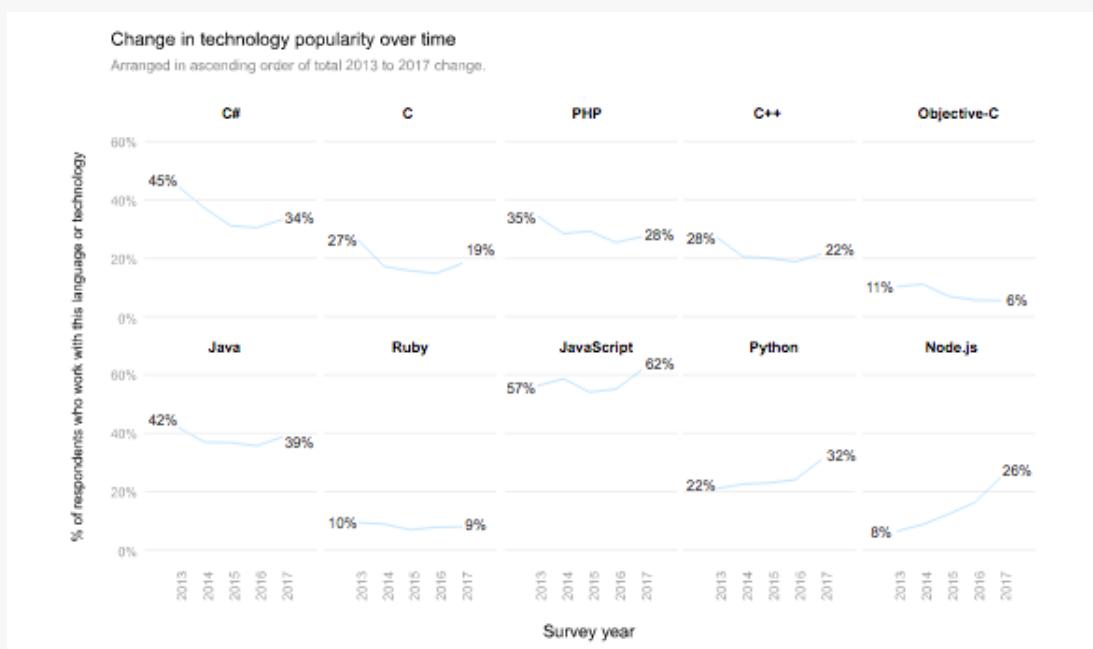


Como pode ser visto, o resultado obtido é similar a todos os outros encontrados pelos demais algoritmos. As redes neurais artificiais são utilizadas em sistemas mais complexos, em que as características do conjunto de dados necessitam de uma análise mais apurada. Os códigos para a construção dessa aplicação podem ser encontrados através deste [link](#).

Anexo A. Introdução ao Python

Python é uma linguagem de programação, concebida na década de 1980, desenvolvida por Guido van Rossum no *Centrum Wiskunde and Informatica* (CWI) na Holanda. Python foi proposta para ser a sucessora da linguagem ABC. Segundo dados da TIOBE (2019), a linguagem Python continua a ser a linguagem de programação que mais cresce no mundo. Segundo projeções, dentro de 3 a 4 anos o Python deve ser a linguagem de programação mais utilizada no mundo, superando a JAVA e C. A Figura A.1, apresenta a variação de popularidade das diferentes linguagens de programação durante os anos.

Figura A.1 – Variação da popularidade das linguagens de programação.



Fonte: OVERFLOW, 2017.

Algumas características que alavancam o crescimento dessa linguagem são:

- Facilidade de aprendizado:

Como essa linguagem apresenta uma programação em alto nível e que se assemelha com a língua inglesa escrita, implementar algoritmos torna-se uma tarefa menos custosa.

- Linguagem portátil e expansividade:

Com o Python é possível integrar componentes do JAVA e dotNet, além de possibilitar a integração com bibliotecas do C e C++. O Python também é suportado pela maioria das plataformas existentes, por exemplo, MAC, Windows, Linux etc.

- Escalabilidade:

Utilizando a linguagem Python é possível criar pequenos programas para prototipagem ou desenvolver grandes sistemas. Existem várias bibliotecas para o desenvolvimento de aplicações.

- Grande comunidade:

Python é utilizada por grandes empresas e por diferentes centros de pesquisas. Engenheiros de software, matemáticos, cientistas de dados, biólogos e praticamente todas as áreas da ciência moderna utilizam das funcionalidades da linguagem Python para desenvolver projetos. Assim, existe uma extensa comunidade com conhecimento em várias áreas que auxiliam na construção de bibliotecas e na solução de problemas.

Python foi desenvolvida para ser uma linguagem interpretada e de script. Entretanto, utilizando os conceitos de programação orientada a objetos e os construtores, a linguagem Python pode ser utilizada como qualquer outra linguagem, naturalmente, orientada a objeto. Funcionalidades como classes, objetos e métodos, além dos princípios de

abstração como encapsulamento, herança e polimorfismo, estão presentes nessa linguagem. Assim, utilizar os conceitos de orientação a objeto para desenvolver aplicações é algo similar a outras linguagens.

O desenvolvimento da linguagem Python foi realizado utilizando os preceitos de que um código simples e fluido é mais elegante e fácil de utilizar que códigos difíceis de interpretar e que empregam otimizações prematuras. Portanto, Python apresenta-se como uma linguagem fácil de interpretar, com alto nível de abstração em que a experiência do usuário é o foco da aplicação.

Como mostrado anteriormente, Python é uma linguagem que preza pela simplicidade e elegância da programação. Para se ter uma ideia desses conceitos, você pode acessar os 20 princípios de desenvolvimento dessa linguagem, diretamente, pelas linhas de comando. Assim, você verá os preceitos do Zen of Python.

Existem duas principais versões do Python: 2.x e 3.x. O x significa a atualização da versão. Essas duas versões apresentam, praticamente, as mesmas funcionalidades, entretanto, existe incompatibilidade entre algumas funções. O exemplo mais comum de incompatibilidade é o print. No Python 2.x ela é um statement, já no Python 3 foi transformada em função. Durante todo o nosso curso, vamos utilizar o Python 3. A seguir, são apresentadas duas opções de utilização do Python 3. A primeira corresponde em instalar em sua própria máquina todos os pacotes necessários. A segunda consiste em utilizar uma conta do Google e a plataforma Google Colaboratory. Utilizando qualquer uma das opções, você estará apto a implementar todos os exemplos de aplicações apresentados nesta disciplina.

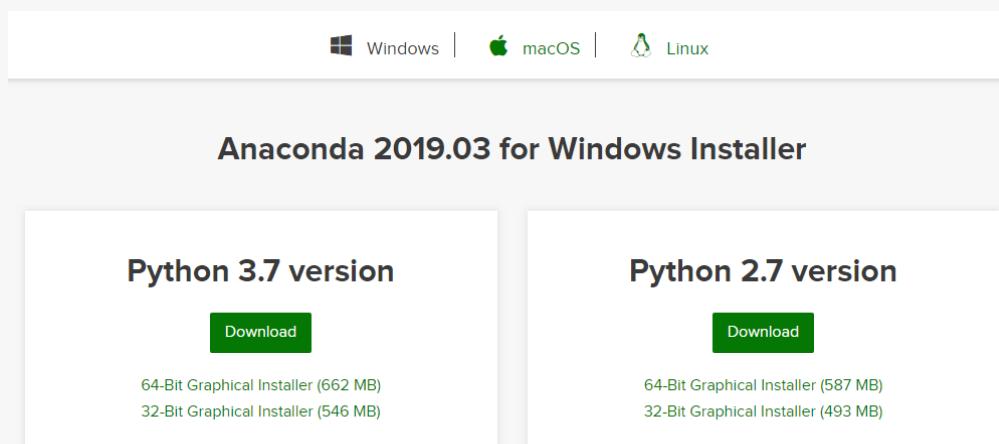
Instalação e Configuração do Ambiente

Existem vários ambientes integrados de desenvolvimento (IDE) para Python. Sugiro utilizar a distribuição Python Anaconda que já possui a IDE Spyder para desenvolvimento. Anaconda é uma distribuição desenvolvida especialmente para aplicações científicas. Com a instalação do Anaconda, vários pacotes como o scikit learn e numpy já vem instalados e configurados. Assim, o seu computador já está praticamente configurado para a utilização de todos os códigos desta disciplina.

Para a instalação do Anaconda, siga os passos a seguir:

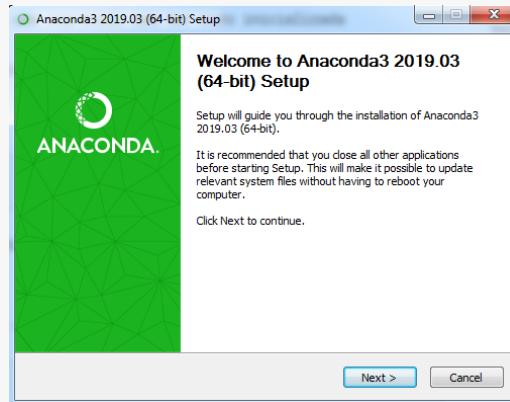
- Baixe e instale o Anaconda (Windows) pelo link: <https://www.anaconda.com/distribution/>.

Figura A.2 – Tela de Download do Anaconda.



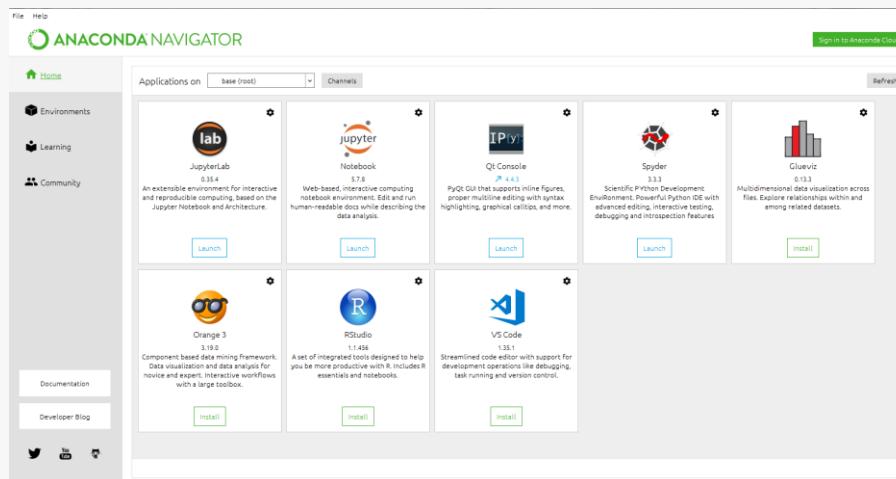
- Escolha a versão Python 3.x.
- Será baixado para a sua máquina o instalador do Anaconda com 700MB, aproximadamente.
- Inicie o instalador do Anaconda.

Figura A.3 – Tela Inicial de instalação do Anaconda.



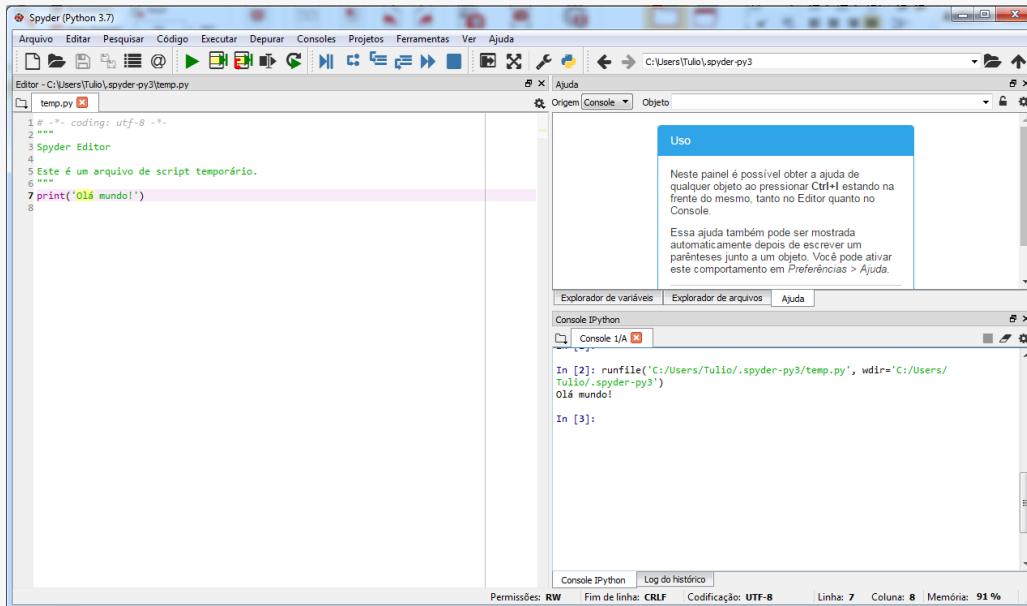
- Recomendo utilizar a instalação padrão da distribuição.
- Após a instalação, você verá uma tela igual a essa:

Figura A.4 – Tela inicial do Anaconda.



- Para iniciar a IDE Spyder, simplesmente, clique no ícone correspondente, presente na Figura A.4.
- Pronto, você já pode executar o print ('Olá mundo!').

Figura A.5 – Olá mundo em Python.



- Para instalar outros pacotes Python no Anaconda, vá ao terminal (prompt de comando) do Anaconda e utilize o comando:
 - pip install nomedopacote
 - Ex.: instalação do pacote Keras.

Figura A.6 – Prompt de comando do Anaconda.

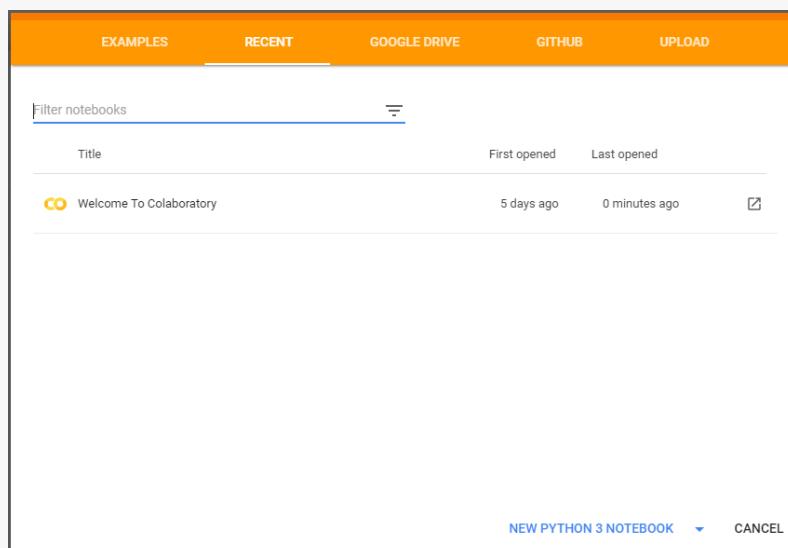
```
(base) C:\Users\Tulio>pip install keras
Collecting keras
  Downloading https://files.pythonhosted.org/packages/5e/10/aa32dad071ce52b55022
66b5c659451cf6ffcbf14e6c8c4f16c0ff5aab/Keras-2.2.4-py2.py3-none-any.whl (312kB)
    100% : [██████████] 317kB 701kB/s
Requirement already satisfied: scipy>=0.14 in c:\users\tulio\anaconda3\lib\site-
packages <from keras> (1.2.1)
Requirement already satisfied: pyyaml in c:\users\tulio\anaconda3\lib\site-packages <from keras> (5.1)
Requirement already satisfied: h5py in c:\users\tulio\anaconda3\lib\site-packages <from keras> (2.9.0)
Collecting keras-applications>1.0.6 <from keras>
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdcf62877ae9102
edf6342d71b28fbfd9dea3d2f96a882ce099b03f/Keras_Applications-1.0.8-py3-none-any.w
hl (500kB)
    100% : [██████████] 51kB 2.0MB/s
Requirement already satisfied: six>=1.9.0 in c:\users\tulio\anaconda3\lib\site-p
ackages <from keras> (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in c:\users\tulio\anaconda3\lib\site-
packages <from keras> (1.16.2)
Collecting keras-preprocessing>=1.0.5 <from keras>
  Downloading https://files.pythonhosted.org/packages/28/6a/8c1f62c37212d9fc441a
7e26736df51ce6f0e38455816445471f10da4f0a/Keras_Preprocessing-1.1.0-py2.py3-none-
```

Segunda Forma de Utilização dos Códigos

Caso você não deseje instalar componentes em sua máquina, uma opção é utilizar o Google Colaboratory. Para acessar o Google Colab é necessário apenas ter uma conta Google e acessar diretamente em seu browser. A Figura A.7 apresenta a tela inicial do Google Colab.

Para a utilização desse ambiente, não é necessário instalar um programa adicional. Entretanto, para realizar o upload de algum dataset, por exemplo, são necessários comandos extras. Todos os comandos necessários para realizar o upload dos dataset estão presentes nos códigos disponibilizados. Para iniciar um novo colab, basta clicar em NEW PYTHON 3 NOTEBOOK e começar a desenvolver a sua aplicação utilizando Python 3.

Figura A.7 – Tela inicial do Google Colaboratory.



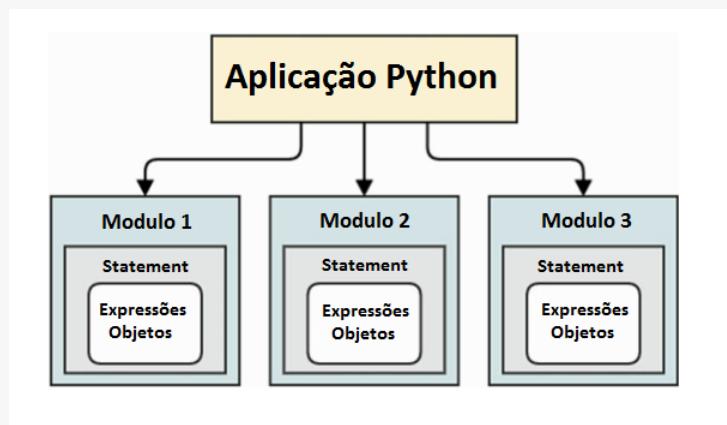
Estrutura e Sintaxe do Python

Para a construção de qualquer programa em qualquer linguagem de programação, é necessário conhecer a estrutura que governa o desenvolvimento e as hierarquias de construção. Em Python, podemos dizer que tudo é um objeto. Funções, estrutura de dados, classes e tipos são

objetos. A Figura A.8 apresenta um diagrama que representa essa hierarquia.

Como dito anteriormente, a estrutura básica em Python são os objetos. Todas as operações são realizadas por meio da manipulação de objetos. Em Python, assim como outras linguagens de programação, possui palavras reservadas.

Figura A.8 – Hierarquia de um programa em Python.



Fonte: adaptado de SARKAR (2016).

A lista com algumas dessas palavras e descrição são apresentadas na Tabela A.1.

Tabela A.1 – Exemplo de algumas palavras-chave e descrição.

Palavra-Chave	Descrição	Exemplo de Utilização
and	Operador lógico AND	(5==5 and 1==2) == False
as	Utilizado como sinônimo de algum objeto/referência	import matplotlib.pyplot as plt
class	Utilizada para criar uma classe (POO)	class GeraNumerosAleatorios()
def	Define funções	def adicao(a,b)
del	Deleta referências	del erro
elif	Condicional Else-if (então-se)	If num==1: print('1') elif num==2: print('2')
else	Condicional Else (então)	If num==1: print('1') else: print('nao e 1')
False	Booleano falso	False==0
for	Loop for	for num in array: print('num')
from	Importa componentes específicos dos módulos	from sklearn import metrics
if	Condicional if	if num==1: print('1')
import	Importa um modulo existente	import numpy
in	Procura ou realiza um loop sobre algum objeto existente	for num in array \ if x in y
is	Utilizado para checar igualdade	type('string') is str
not	Operador lógico not	not 1==2
or	Operador lógico or	1 or 2 ==1
return	Retorna objeto(s) de uma função existente	return a,b
while	O loop while	while True: print('valor')

Fonte: adaptado de SARKAR (2016).

Estrutura e Tipos de Dados

Como dito anteriormente, todos os elementos em Python são objetos. Cada objeto possui propriedades específicas que são utilizadas para distingui-los. Essas propriedades são:

- Identidade: é única. É criada no momento em que o objeto é criado. Normalmente, é representado pelo endereço de memória do objeto.
- Tipo (type): determina o tipo de objeto.
- Valor (value): representa o valor real (atual) armazenado pelo objeto.

A Figura A.9 apresenta as propriedades de um objeto string que foi criado. Por ela é possível perceber a ação dessas propriedades na prática.

Figura A.9 – Apresentação das propriedades de um objeto.

```
[3] minha_string = "Disciplina Aplicações em Deep Learning" #armazena a string  
[4] id(minha_string) #identificador do objeto  
⇒ 139908227640056  
[5] type(minha_string) #tipo de objeto  
⇒ str  
[6] minha_string #mostra a string armazenada  
⇒ 'Disciplina Aplicações em Deep Learning'
```

Tipos de Dados em Python

Em Python, existem vários tipos de dados. Nesta apostila, vamos falar apenas dos mais importantes para o desenvolvimento de nossas aplicações. Quando existir a necessidade, as estruturas que são apresentadas neste tópico serão discutidas no momento da apresentação dos conceitos.

Dados Numéricos

Existem, basicamente, 3 tipos de dados numéricos em Python. Inteiros (integers), pontos flutuantes (floats) e números complexos. A Figura A.10 apresenta algumas operações básicas com números inteiros.

Figura A.10 – Algumas operações básicas com números inteiros.

```
[ ] numero_inteiro= 1234 # representação de numero inteiro
[ ] type(numero_inteiro)
[ ] <int>
[ ] numero_inteiro + 10 # adição
[ ] <int> 1244
[ ] numero_inteiro - 10 #subtração
[ ] <int> 1224
[ ] numero_inteiro * 2 # multiplicação
[ ] <int> 2468
[ ] numero_inteiro / 2 # divisão
[ ] <float> 617.0
```

Para os números em pontos flutuantes, podemos representá-los por meio de valores decimais ou em notação de expoente (e ou E seguido de +/- indicando o sentido da notação). A Figura A.11 apresenta alguns exemplos de operações utilizando pontos flutuantes.

Figura A.11 – Algumas operações básicas com ponto flutuante.[19] $1.5 + 3.7$

↳ 5.2

[20] $12e3 + 5e3$

↳ 17000.0

[21] $2.5e-3$

↳ 0.0025

[22] $2.5e-3 + 0.0025$

↳ 0.005

Os números complexos são representados por dois componentes distintos. Eles possuem uma parte real, representado por um float, e uma parte imaginária, representada por um float seguido da letra j. A Figura A.12 apresenta algumas operações realizadas com números complexos.

Figura A.12 – Algumas operações básicas com números complexos.[24] `numero_complexo = 2 + 5j # representação de um número complexo`[25] `numero_complexo + 3`

↳ (5+5j)

[26] `numero_complexo + 6j`

↳ (2+11j)

[30] `type(numero_complexo)`

↳ complex

[28] `numero_complexo.imag`

↳ 5.0

[29] `numero_complexo.real`

↳ 2.0

Strings

As strings são sequências de caracteres utilizadas para representar texto. As strings podem armazenar dados textuais ou bytes como informação. Como dito anteriormente, as strings são objetos que possuem vários métodos já implementados que permitem manipulá-las de maneira mais conveniente. Uma característica das strings é que elas representam objetos imutáveis, ou seja, toda operação realizada com uma string cria uma nova. A Figura A.13 apresenta algumas operações realizadas com strings. Como pode ser visto, existem várias operações que podem ser realizadas sobre os objetos string.

Figura A.13 – Algumas operações com strings.

```
[2] s1 = "Esta é a disciplina"  
  
[3] s2 = " Aplicações em Deep Learning"  
  
[4] print(s1,s2)  
↳ Esta é a disciplina Aplicações em Deep Learning  
  
[5] print(s1+'\n'+s2)  
↳ Esta é a disciplina  
Aplicações em Deep Learning  
  
[6] ''.join([s1,s2]) #concatena a lista de strings  
↳ 'Esta é a disciplina Aplicações em Deep Learning'  
  
[7] s1[::-1] # percorre a string de maneira inversa  
↳ 'anilpicsid a é atsE'
```

Listas

Em Python, listas podem ser um conjunto de objetos heterogêneos ou homogêneos. Nas listas, os objetos são ordenados seguindo a sequência em que foram adicionados. Cada um dos elementos possui o seu próprio

índice. Cada objeto pode ser acessando através da referência ao índice que possui. A Figura A.14 apresenta alguns exemplos de como as listas podem ser criadas. Na Figura A.15 são mostradas algumas operações sobre listas.

Figura A.14 – Exemplos criação e acesso às listas.

```
[8]  l1= ['azul','branco', 'amarelo', 'vermelho']  #exemplo de criação de listas
[9]  l2= list ([1,'coxinha',10,'pasteis',0.25e-3])  #exemplo de criação de listas
[10] l3= [1,2,3,['a','b','c'], ['IGTI','ADL']]
[12] print(l1,l2)
    ['azul', 'branco', 'amarelo', 'vermelho'] [1, 'coxinha', 10, 'pasteis', 0.00025]
[13] print(l3)
    [1, 2, 3, ['a', 'b', 'c'], ['IGTI', 'ADL']]
[14] #acessando itens das listas
    l1
    ['azul', 'branco', 'amarelo', 'vermelho']
[15] l1[0]
    'azul'
[16] l1[0]+' '+l1[2]
    'azul amarelo'
[17] #acessando intervalos de itens
    l2[1:3]
    ['coxinha', 10]
```

Figura A.15 – Exemplos de listas e operações que podem ser executadas.

```
[28] numeros= list(range(10)) # gera uma lista contendo os numeros de 0 a 9
      numeros[:]

      ↗ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[29] numeros[2:5]

      ↗ [2, 3, 4]

[30] #concatenando listas
      numeros*2

      ↗ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[32] numeros+12

      ↗ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 'coxinha', 10, 'pasteis', 0.00025]

[33] #acessando listas aninhadas
      l3

      ↗ [1, 2, 3, ['a', 'b', 'c'], ['IGTI', 'ADL']]

[34] l3[3]

      ↗ ['a', 'b', 'c']

[35] l3[4]

      ↗ ['IGTI', 'ADL']

[36] l3[3][1]

      ↗ 'b'

[37] #adicionando elemetos à lista
      l3.append('Aplicações em Deep Learning')

[38] l3

      ↗ [1, 2, 3, ['a', 'b', 'c'], ['IGTI', 'ADL'], 'Aplicações em Deep Learning']

[39] #retirando um elemento da lista
      l3.pop(3)

      ↗ ['a', 'b', 'c']

      ↗ l3

      ↗ [1, 2, 3, ['IGTI', 'ADL'], 'Aplicações em Deep Learning']
```

Conjuntos (sets)

Os conjuntos ou “sets” são um conjunto de objetos únicos e imutáveis. Para criá-lo, é utilizado o método set() ou {}. A Figura A.16 apresenta exemplos de criação e operações com esses conjuntos.

Figura A.16 – Exemplos de conjuntos e operações que podem ser executadas.

```
[41] l1 = [1,2,3,3,3,4,5,5,5,6,6,6,7,7,7,7,8,8]. #cria uma lista com numeros repetidos  
[42] set(l1)    #cria um conjunto através do método set().  
⇒ {1, 2, 3, 4, 5, 6, 7, 8}  
[43] s1 = set(l1).  
[44] #verifica se um elemento pertence ao conjunto  
     1 in s1  
⇒ True  
[45] 10 in s1  
⇒ False  
[49] s2 = {3,5,10,11,12}  #cria um conjunto através do {}  
[50] #operações com conjuntos  
     s1 - s2      # diferença entre conjuntos  
⇒ {1, 2, 4, 6, 7, 8}  
[51] s1 | s2    #união de conjuntos  
⇒ {1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12}  
[52] s1 & s2  #interseção de conjuntos  
⇒ {3, 5}  
▶ s1 ^ s2 # elementos que não aparecem, ao mesmo tempo, em ambos  
⇒ {1, 2, 4, 6, 7, 8, 10, 11, 12}
```

Dicionários (dict)

Dicionários são conjuntos em Python mutáveis e não ordenados. Eles possuem acesso através de um valor chave ‘key’. As chaves são sempre estruturas imutáveis. Eles são criados através do método dict(). A estrutura dos dicionários é similar a um JSON. A Figura A.17 mostra alguns exemplos de criação e operações com dicionários.

Figura A.17 – Exemplos de dicionários e operações que podem ser executadas.

```
[55] d1 = {'azul': 3, 'branco': 4, 'verde': 6, 'preto': 1}
[56] #recuperando itens de um dicionário
      d1.get('azul')
[57] d1['azul']
[58] 3
[59] # adicionando valores
      d1['amarelo']=4
      d1
[60] d1.keys()  #recuperando as chaves
[61] d1.values()  # recuperando os valores
[62] dict_keys(['azul', 'branco', 'verde', 'preto', 'amarelo'])
[63] dict_values([3, 4, 6, 1, 4])
[64] #cria dicionário utilizando o método dict()
      d2 = dict({'laranja': 47,'roxo': 21,'azul':58})
      d2
[65] {'azul': 58, 'laranja': 47, 'roxo': 21}
[66] #atualizando o dicionário d1 utilizando o d2
      d1.update(d2)
      d1
[67] {'amarelo': 4,
      'azul': 58,
      'branco': 4,
      'laranja': 47,
      'preto': 1,
      'roxo': 21,
      'verde': 6}
```

Tuplas

Tuplas, assim como as listas, também são sequências de objetos. Entretanto, as tuplas são imutáveis. São utilizadas, normalmente, para representar uma coleção de objetos ou valores. A Figura A.18 apresenta as formas de criação e operações que podem ser realizadas com as tuplas.

Figura A.18 – Exemplos de tuplas e operações que podem ser executadas.

```
[67] #criando tuplas
t1= (1,)
t1

↳ (1,)

[68] #identificação da tupla
id(t1)

↳ 140097858444088

[69] # modificar o conteúdo da tupla, modifica o endereço
t1 = t1 + (2,3,4,5,6)
t1

↳ (1, 2, 3, 4, 5, 6)

[70] id(t1)

↳ 140097840249928

[71] # tuplas são imutáveis
t1[2]= 1000

↳ -----
TypeError                                 Traceback (most recent call last)
<ipython-input-71-a5f2253b0903> in <module>()
----> 1 t1[2]= 1000

TypeError: 'tuple' object does not support item assignment

SEARCH STACK OVERFLOW

[72] tupla = ([['IGTI','Disciplina'],['Aplicações','Deep Learning']])
tupla[0]

↳ ['IGTI', 'Disciplina']

[74] l1,l2 = tupla
print(l1,l2)

↳ ['IGTI', 'Disciplina'] ['Aplicações', 'Deep Learning']
```

Controle de Fluxo

Python proporciona várias formas de criar controles de fluxo para os programas. Alguns exemplos de controle de fluxo disponíveis são:

- Condicionais If-elif-else;
- Loop for;
- Loop while;
- Loop break, continue e else;

- Try-except.

Condicionais

A Estrutura Condicional possibilita a escolha de um grupo de ações e estruturas a serem executadas quando determinadas condições são ou não satisfeitas. Em Python, essas estruturas possuem o seguinte formato:

```
if <verifica se a condição 1 é verdadeira (True)>    #condição mandatória  
  
<bloco 1 de      #somente é executado se a #condição 1 for verdadeira.  
  
elif <verifica se a condição 2 é verdadeira (True)> #condição opcional  
  
<bloco 2 de só # é executado se a condição 1 é #falsa e a 2 for verdadeira  
  
else:  
  
<bloco 3 de códigos>          #executado apenas quando 1 e 2 forem falsas
```

A Figura A.19 mostra alguns exemplos de utilização desses condicionais para comandar o fluxo de ações dentro de um código.

Figura A.19 – Exemplos de construção de condicionais.

```
[76] variavel = 'azul'  
      if variavel =='azul':  
          print("Variável é AZUL")
```

⇒ Variável é AZUL

```
[78] variavel = 'branco'  
      if variavel =='azul':  
          print("Variável é AZUL")  
      elif variavel == 'branco':  
          print('Variável é BRANCO')
```

⇒ Variável é BRANCO

```
[79] variavel = 'verde'  
      if variavel =='azul':  
          print("Variável é AZUL")  
      elif variavel == 'branco':  
          print('Variável é BRANCO')  
      else:  
          print('Variável é VERDE')
```

⇒ Variável é VERDE

Construção de Loops

Assim como em outras linguagens, as duas principais estruturas de repetição em Python são: **for** e **while**. Essas estruturas são utilizadas para que um bloco de comandos seja executado repetidas vezes. Em Python, existe a opção de colocar o comando **else** ao final da execução do loop. Assim, depois de finalizada a execução da estrutura de repetição (sem a utilização do comando **break**), os comandos que estiverem dentro do **else** serão executados.

#Loop for

for item **in** lista: #realiza o loop para cada elemento na lista

<bloco de código> #executado repetidamente

else: #comando opcional, executado apenas se o loop terminar #normalmente (sem a utilização do **break**)

#Loop while

while <condicional>: #repete até que a condição seja satisfeita

< bloco de código> # executado repetidamente

else: #comando opcional, executado apenas se o loop terminar normalmente (sem a utilização do **break**)

A Figura A.20 exemplifica a utilização dos comandos de repetição **for** e **while**.

Figura A.20 – Exemplos de construção de Loop for e while.

```
[80] #ilustrando loops
sequencia = range(0,5)
for num in sequencia:
    print(num)

C> 0
1
2
3
4

[82] sum = 0
for num in sequencia:
    sum+= num
print(sum)

C> 10

[83] sequencia = range(0,5)
for num in sequencia:
    print(num)
else:
    print('loop terminou normalmente')

C> 0
1
2
3
4
loop terminou normalmente

[84] sequencia = range(0,5)
for num in sequencia:
    if num <3:
        print(num)
    else:
        break
else:
    print('loop terminou normalmente')

C> 0
1
2

[85] #ilustrando loop while
num = 5
while num >0:
    print(num)
    num -=1      #necessário para evitar o loop infinito

C> 5
4
3
2
1
```

Funções

As funções podem ser definidas como um bloco de códigos que será executado apenas quando for invocado. Todas as funções precisam ter uma assinatura def e um nome específico para serem invocadas. Dentro das funções, existem códigos que devem ser executados após ela ser invocada. O pseudocódigo a seguir mostra como o bloco funcional é criado.

```
def exemploFuncao (parâmetros): # parâmetros = valores de entrada  
  
< bloco de códigos>  
  
return valores           #bloco opcional para retorno da função
```

A Figura A.21 mostra a implementação de algumas funções simples.

Figura A.21 – Exemplos de construção de funções em Python.

```
[87] #exemplo de uma função que retorna o quadrado de um número  
def retornaQuadrado(numero):  
    return numero*numero  
  
numero=5  
print("O quadrado de {} é: {}".format(numero,retornaQuadrado(numero)))  
  
⇒ O quadrado de 5 é: 25
```

Classes

Classes em Python são estruturas que permitem utilizar o paradigma de Orientação a Objetos para desenvolver programas que utilizem os conceitos de objetos, encapsulamento, métodos, herança e polimorfismo. Abaixo é apresentado o pseudocódigo para construção de classes em Python.

```
class NomeDaClasse (ClasseBase):  
    variáveis_da_classe    # variáveis compartilhadas com todas as  
    instâncias  
  
    def __init__ (self, ...): #construtor da classe
```

```
#instancia as variáveis que são únicas para cada objeto/instancia  
  
    self.variaveis_unicas = ...  
  
def __str__(self): representação em string do objeto/instancia  
  
    return representação(self)  
  
def métodos(self, ...):      #criação de métodos da classe  
  
    < bloco de código>
```

O pseudocódigo mostrado anteriormente, indica que a classe NomeDaClasse herda os parâmetros definidos pela classe ClasseBase. O parâmetro `self` é, normalmente, utilizado como o primeiro parâmetro de cada método, pois é ele o responsável por referenciar a instância ou objeto da classe NomeDaClasse e chamar os métodos correspondentes.

Todos os comandos apresentados neste capítulo e em toda a apostila podem ser baixados e executados através deste [link](#).

Agora que você já conhece todos os comandos que serão utilizados no decorrer desta disciplina, podemos começar a mostrar algumas aplicações interessantes que utilizam o Deep Learning para a solução de problemas.

Anexo B. Desenvolvendo aplicações com Keras

Neste capítulo, serão apresentadas algumas aplicações em Deep Learning utilizando a linguagem Python e o framework Keras. Ao final, os alunos estarão aptos a:

- Diferenciar os frameworks Keras, TensorFlow e PyTorch;
- Criar aplicações de Aprendizado de Máquina utilizando o Keras.

Diferenças entre Keras, TensorFlow e Pytorch

Keras, TensorFlow e PyTorch são frameworks para o desenvolvimento de aplicações que utilizam algoritmos ou estratégias de Aprendizado de Máquina. Esses 3 frameworks possuem código aberto e foram desenvolvidos para facilitar a implementação desses algoritmos. Apesar de possuírem a mesma finalidade, apresentam características distintas. A Tabela B.1 apresenta um resumo sobre as diferenças entre esses 3 frameworks.

Tabela B.1 – Comparaçāo entre Keras, TensorFlow e PyTorch

Característica	Keras	TensorFlow	PyTorch
Desenvolvedor	François Chollet	Google	Facebook
Open source	Sim	Sim	Sim
Nível de Abstração	Alto	Alto e Baixo	Baixo
Velocidade	Inferior aos outros	Alta	Alta
Arquitetura	Simples	Não muito simples	Complexa
Facilidade de Programação	Simples	Mais complexa	Bem mais complexa
Debugging	Praticamente, não precisa	Complexo	Não muito complexo
Comunidade	Menor dos três	Grandes empresas	Grande: pesquisadores e empresas
Volume de dados	Menor dos três	Grande volume	Grande volume
Flexibilidade	Menor dos três	Flexível	Grande flexibilidade

A escolha entre qual desses frameworks utilizar para a implementação de aplicações de Deep Learning deve ser realizada com base nos objetivos da aplicação. No caso do nosso curso de aplicações, o objetivo é mostrar ou ilustrar algumas aplicações de Deep Learning sem o intuito de obter melhores performances no desenvolvimento dessas aplicações. Assim, foi

escolhido o Keras como framework para a implementação dos exemplos apresentados nesta disciplina. Caso o foco da disciplina fosse a implementação de algoritmos com a maior performance possível com a utilização de grandes bases de dados, certamente, a escolha seria diferente. Portanto, a escolha sobre qual framework utilizar deve ser realizada observando-se os objetivos que se deseja alcançar.

Entendendo o Keras

O Keras é uma *Application Programming Interface* (API) de desenvolvimento de aplicações em Machine Learning de alto nível que pode utilizar o TensorFlow, Teano ou o Microsoft Cognitive Toolkit (CNTK) para implementar os algoritmos de Deep Learning. O Keras foi desenvolvido para facilitar o desenvolvimento de aplicações com uma interface mais amigável para o usuário. As principais vantagens em utilizar o Keras são:

- Facilidade de implementação:

Com apenas algumas linhas de código é possível criar uma rede neural artificial, definir as entradas e saídas, treinar essa rede e gerar previsões.

- Open source:

Como o Keras possui o código aberto, é possível que exista grande interação entre os desenvolvedores e a comunidade que utiliza esse framework.

- Documentação extensa e completa:

O Keras conta com uma documentação bastante completa e constantemente atualizada. Assim, é mais fácil compreender o funcionamento de toda a estrutura do framework.

- Utiliza o TensorFlow como backend.

Como o Keras pode utilizar o TensorFlow como backend, é possível construir modelos que utilizem o TensorFlow para o treinamento e avaliação da rede neural criada. Portanto, podemos fazer uso da alta performance do TensorFlow utilizando as facilidades de programação do Keras.

Para a construção de redes neurais com o Keras, estão disponíveis dois modos: o modelo Sequencial e o modelo Funcional. No modelo sequencial, a rede é construída em sequência, ou seja, cada uma das camadas é adicionada uma após a outra em uma espécie de pilha. Na primeira camada da rede, é obrigatório definir as dimensões da entrada. A Figura B.1 apresenta um exemplo de criação de uma rede neural utilizando o modelo sequencial.

Figura B.1 – Exemplo de criação de uma rede utilizando o modelo sequencial do Keras.

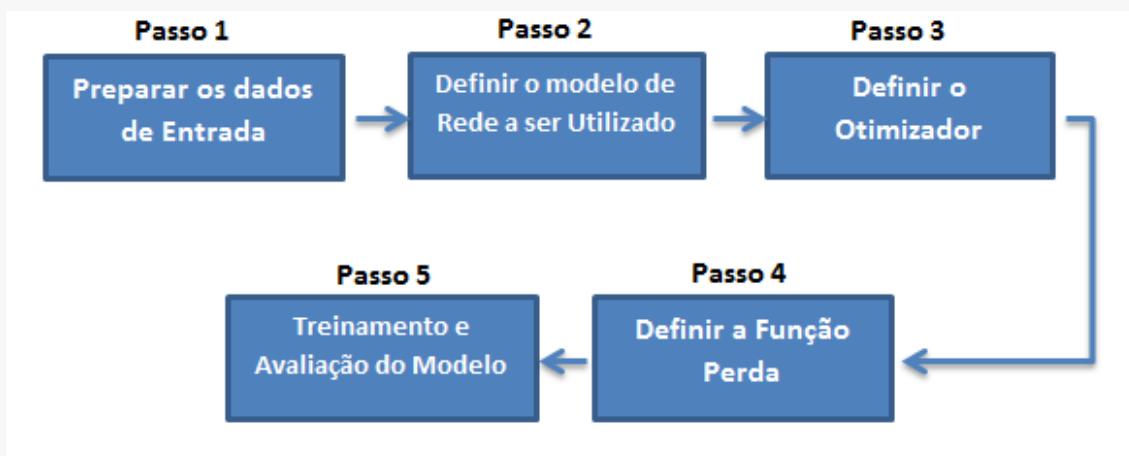
```
model = Sequential()  
  
model.add(Dense(4, activation='relu', input_dim=3))  
model.add(Dense(4, activation='relu',input_dim=4))  
model.add(Dense(4, activation='relu',input_dim=4))  
model.add(Dense(1, activation='relu', input_dim=4))
```

O modelo funcional do Keras apresenta algumas diferenças. Nesse modelo, a ligação entre as camadas ocorre de forma diferente. É necessário indicar qual deve ser a entrada em cada uma das camadas, ou seja, para cada camada adicionada ao modelo, deve-se indicar a saída de qual camada será utilizada como entrada dessa camada criada. Na camada de entrada, também é necessário indicar qual é o formato dos dados utilizados para o treinamento e avaliação do modelo. A Figura B.2 mostra um exemplo de utilização do modelo funcional para a criação de uma rede simples.

Figura B.2 – Exemplo de criação de uma rede utilizando o modelo funcional do Keras.

```
#cria a camada de entrada para a rede  
entrada=Input(shape=(3,))#especifica o tamanho (formato) da entrada  
  
x1=Dense(4,activation='relu')(entrada)# cria a primeira camada escondida e o link com a camada de entrada  
x2=Dense(4,activation='relu')(x1) # cria a segunda camada escondida  
y=Dense(1,activation='softmax')(x2)  
  
#cria o modelo  
model=Model(inputs=entrada,outputs=y.)
```

Para a implementação de todas as aplicações presentes neste curso, será utilizado o modelo sequencial. Além disso, para essas implementações, será adotada a sequência de passos presentes na Figura B.3. A adoção desses passos visa tornar sistemática a aplicação dos conceitos apresentados no decorrer deste curso.

Figura B.3 – Passos a serem seguidos para implementar as redes no Keras.

Referências

- ACTION, Portal. *Estacionariedade.* Disponível em: <<http://www.portalaction.com.br/series-temporais/11-estacionariedade>>. Acesso em: 10 mai. 2021.
- AMAZON. *Model Fit: Underfitting vs. Overfitting.* Disponível em: <<https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>>. Acesso em: 10 mai. 2021.
- ANTOL, Stanislaw et al. Vqa: Visual question answering. In: *Proceedings of the IEEE international conference on computer vision*. 2015. p. 2425-2433.
- ARYAL, Milan. *Object Detection, Classification, and Tracking for Autonomous Vehicle.* 2018.
- BISGAARD, Søren; KULAHCI, Murat. *Time series analysis and forecasting by example.* John Wiley & Sons, 2011.
- BROCK, D. L. *The electronic product code (epc).* Auto-ID Center White Paper MIT-AUTOIDWH-002, 2001.
- BRUNER, Jon; DESHPANDE, Adit. *Generative Adversarial Networks for beginners.* O'Reilly, 2017. Disponível em: <<https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners>>. Acesso em: 13 jun. 2019.
- DAS, Aneek. *The very basics of Reinforcement Learning.* Medium, 2017. Disponível em: <<https://becominghuman.ai/the-very-basics-of-reinforcement-learning-154f28a79071>>. Acesso em: 10 mai. 2021.
- DE AGUIAR, Erikson Júlio et al. Análise de sentimento em redes sociais para a língua portuguesa utilizando algoritmos de classificação. In: *Anais do*

XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. SBC, 2018.

Departamento de Atenção Básica. *Obesidade/Ministério da Saúde, Secretaria de Atenção à Saúde, Departamento de Atenção Básica.* Brasília: Ministério da Saúde, 2006. 108 p. il. - (Cadernos de Atenção Básica, n. 12) (Série A. Normas e Manuais Técnicos)

DONAHUE, Jeffrey et al. Long-term recurrent convolutional networks for visual recognition and description. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015. p. 2625-2634.

EHLERS, Ricardo S. *Análise de séries temporais*. Laboratório de Estatística e Geoinformação. Universidade Federal do Paraná, 2007.

FACURE, Matheus. *Aprendendo Representações de Palavras*. 2017. Disponível em: <<https://matheusfacure.github.io/2017/03/20/word2vec/>>. Acesso em: 10 mai. 2021.

FRAZÃO, Fernando. *Seleção brasileira de futebol enfrenta a Alemanha*. Wikimedia Commons, 2016. Disponível em: <https://commons.wikimedia.org/wiki/File:Sele%C3%A7%C3%A3o_brasileira_de_futebol_enfrenta_a_Alemanha_1039203-20.08.2016_frz-01.jpg>. Acesso em: 10 mai. 2021.

GHAZVININEJAD, Marjan et al. Generating topical poetry. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016. p. 1183-1191.

GOODFELLOW, Ian et al. Generative adversarial nets. In: *Advances in neural information processing systems*. 2014. p. 2672-2680.

ISAKSSON, Robin. *Reduction of Temperature Forecast Errors with Deep Neural Networks*. 2018.

ITER, Dan et al. *Rastreamento de alvo com filtragem de kalman, knn e lstms*. 2016

JOHNSON, Daniel. *Composing Music With Recurrent Neural Networks*. 2015. Disponível em: <<http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>>. Acesso em: 10 mai. 2021.

JOSHI, Ameet V. *Machine learning and artificial intelligence*. Springer, 2020.

LECUN, Yann; CORTES, Corinna; BURGES, Christopher J.c. *THE MNIST DATABASE of handwritten digits*. Disponível em: <<http://yann.lecun.com/exdb/mnist/>>. Acesso em: 10 mai. 2021.

MAO, Hui; CHEUNG, Ming; SHE, James. Deepart: Learning joint representations of visual arts. In: *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017. p. 1183-1191.

MNIH, Volodymyr et al. Playing atari with deep reinforcement learning. In: *arXiv preprint*, 2013.

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre aprendizado de máquina. In: *Sistemas inteligentes-Fundamentos e aplicações*, v. 1, n. 1, p. 32, 2003.

NEDELTCHEV, P. *The internet of everything is the new economy*. Cico, 2014. Disponível em: <<https://blogs.cisco.com/ciscoit/b-ioe-08132014-iae-is-the-new-economy>>. Acesso em: 10 mai. 2021.

OVERFLOW, Stack. *Developer Survey Results (2017)*. 2017. Disponível em: <https://insights.stackoverflow.com/survey/2017?utm_source=medium&utm_medium=referral>

[m=hero&utm_campaign=dev-survey-2017&utm_content=hero-questions#technology-languages-over-time](https://www.xper.com.br/m=hero&utm_campaign=dev-survey-2017&utm_content=hero-questions#technology-languages-over-time). Acesso em: 10 mai. 2021.

PAL, Avishek; PRAKASH, P. K. S. *Practical Time Series Analysis*: Master Time Series Data Processing, Visualization, and Modeling using Python. Packt Publishing Ltd, 2017.

PEDERSEN, Magnus Erik Hvass. *Style Transfer*. 2016. Disponível em: <https://www.zepl.com/viewer/github/Hvass-Labs/TensorFlow-Tutorials/blob/master/15_Style_Transfer.ipynb>. Acesso em: 13 jun. 2019.

TIOBE. *TIOBE Index for May 2021*. Disponível em: <<https://www.tiobe.com/tiobe-index/>>. Acesso em: 10 mai. 2021.

WEISER, Mark. The computer for the 21st century. In: *ACM SIGMOBILE mobile computing and communications review*, v. 3, n. 3, p. 3-11, 1999.

YAN, Rui. i, Poet: Automatic Poetry Composition through Recurrent Neural Networks with Iterative Polishing Schema. In: *IJCAI*. 2016. p. 2238-2244.