INTELIGÊNCIA ARTIFICIAL APLICADA - MACHINE LEARNING

AULA 3

Prof. Antonio Willian Sousa

CONVERSA INICIAL

Nesta aula, apresentaremos o funcionamento dos métodos de aprendizagem supervisionada, começando com um algoritmo clássico chamado *árvore de decisão*. Vamos detalhar como os dados devem ser preparados para o processo de treinamento desse tipo de algoritmo e como se dá o seu funcionamento e treinamento. Também vamos mostrar como utilizar um modelo já treinado e apresentar alguns outros algoritmos semelhantes.

TEMA 1 – PREPARAÇÃO DOS DADOS

Anteriormente, conversamos sobre as bases de dados utilizadas para aprendizagem, os *datasets*. Também falamos de como esses dados podem ser obtidos e a forma como eles normalmente são utilizados, sendo essencial avaliar o *dataset* a ser utilizado de acordo com o tipo de problema que se deseja resolver.

1.1 Obtendo os dados

A obtenção dos dados, como já dissemos, está intrinsecamente relacionada ao problema que se deseja resolver. Assim, o processo de escolha desses dados é essencial. Há vários repositórios que disponibilizam dados para os mais diversos tipos de problemas, porém, muitas vezes, poderemos nos encontrar em uma situação na qual os dados do problema são tão específicos que será necessário criar o próprio *dataset* do zero.

Ao obter ou construir um *dataset*, devemos nos atentar para os aspectos da quantidade de dados e para a qualidade destes. Se a quantidade de dados obtidos é pouca, o modelo até poderá aprender, mas a sua capacidade de generalização pode ser seriamente prejudicada. Todavia, também devemos nos atentar para a qualidade desses dados, pois dados incompletos ou com informações incorretas até podem ser utilizados para treinar um modelo, mas certamente este terá sérias deficiências em seu funcionamento, as quais serão proporcionais à qualidade dos dados que lhe foram fornecidos no processo de treinamento. Assim, devemos sempre considerar que a qualidade dos dados vem antes da quantidade, mas o ideal é que os dois aspectos estejam sempre presentes: quantidade e qualidade.

O trabalho de preparação dos dados se tornou tão essencial para o treinamento de modelos de aprendizagem de máquina, que uma grande parcela de todo o processo é gasta nessa etapa. Assim, reduzir o tempo de preparação dos dados é uma necessidade comum e imperativa.

Entre os problemas com os quais podemos nos defrontar ao selecionar os dados, podemos destacar:

- buscar e identificar dados de relevância para o seu problema com base em fontes externas, por meio da exploração e análise de diversos *datasets*;
- converter conjuntos de arquivos em formato txt, json ou xml em dados tabulares para facilitar a investigação dos dados, bem como a sua manipulação e o fornecimento destes para os algoritmos de aprendizagem;
- determinar quais colunas ou atributos armazenados em arquivos CSV ou tabelas de bancos relacionais são relevantes para o problema.

Devemos considerar ainda que nem sempre os dados, mesmo que sejam obtidos de modo organizado e separado, estão de uma forma que nos permita utilizá-los de imediato. Situações em que os dados estão distribuídos em largos conjuntos de arquivos menores é algo bem comum nessa área, sendo necessário estar preparado para contornar situações dos mais diversos tipos. Assim, algumas considerações devem ser feitas durante a etapa de coleta dos dados que pode ajudar a obter dados melhores e ganhar tempo nas etapas subsequentes. Entre algumas das considerações que devemos fazer ao coletar dados, as seguintes podem ser de grande ajuda:

- ter certeza de como os dados estão armazenados e de como poderemos utilizá-los – se eles foram armazenados em tabelas de bancos relacionais, se possuem relação temporal e como estes estão disponibilizados, e se os formatos e codificações não serão um problema;
- verificar se todos os dados de que precisamos estão disponíveis e, se caso não estiverem, assegurar se temos como produzir esses dados ou simulá-los – caso isso não seja possível, é importante verificar a possibilidade de eliminá-los do processo de aprendizagem ou de utilizar alguma outra abordagem;

- verificar se precisaremos de todos os dados que estão disponíveis – apesar de a quantidade de dados ser um fator determinante na qualidade do modelo, excesso de dados pode tornar o processo de aprendizagem lento e até mesmo impreciso.

Após obter os dados, precisamos avaliar de forma preliminar a sua qualidade, buscando por *outliers*, exceções, inconsistências, dados faltantes, além de analisar o balanceamento e tendências que os dados possam apresentar. Se precisarmos criar um modelo de classificação de clientes direcionado pelo gênero e 90% dos dados se referem a clientes do sexo feminino assim como apenas 10% se referem a clientes do sexo masculino, é muito provável que esse modelo será treinado com uma alta capacidade de reconhecer os clientes do gênero feminino, mas baixa para reconhecer os clientes do gênero masculino. Dizemos que o modelo possui uma tendência, um desvio ou um *bias*.

Uma vez que os dados já foram avaliados e que problemas, falhas e inconsistência foram detectados, é hora de avaliar quais partes dos dados serão úteis e quais não serão, executando ações de pré-processamento neles.

1.2 Pré-processando os dados

Os passos de pré-processamento dos dados podem incluir diversas ações diferentes. Dentre as ações mais comuns a serem encontradas, temos:

- formação dos dados: os dados podem estar armazenados em formato diferente daquele de que necessitamos ou, simplesmente, com uma codificação diferente;
- padronização de valores: alguns dados podem estar com representações distintas. Uma coluna contendo valores em moeda pode vir representada contendo a indicação “R$ 10,00”, “$ 10,00” ou “10,00”. Será necessário executar uma padronização desses valores, dando preferência àquelas em que se tenha apenas números e em que a indicação do tipo de moeda possa ficar em uma coluna em separado;
- limpeza dos dados: nesse tipo de ação pode ser necessário remover registros inteiros ou completar os dados faltantes, sendo de grande importância avaliar quais valores são faltantes e também avaliar se esses dados são realmente importantes para o problema;

- evitar dados duplicados ou repetidos: para dados do tipo texto, devemos sempre nos certificar de que não há repetição de dados, como nas situações em que os dados de primeiro nome, último nome e nome completo estão representados em colunas distintas de dados, quando poderia ser utilizada apenas uma representação;
- melhorar a qualidade dos dados: dados representados por variáveis contínuas podem apresentar *outliers* ou grandes desvios nas faixas de valores. Esses desvios podem ser descobertos por meio de análise de histogramas, mas os ajustes desses dados devem ser feitos com cautela para não criar uma situação em que os dados não reflitam a realidade.

Uma atenção especial deve ser dada aos processos de reparação de dados faltantes, pois tratar esses dados de forma incorreta pode conduzir a modelos imprecisos. Assim, há situações em que preencher os dados de forma incorreta pode ser desastroso, mas deixar de preencher esses dados para que possam ser considerados na aprendizagem, também. Por exemplo, se for necessário um modelo para sugerir itens a serem comprados por torcedores de futebol e, dentro do seu conjunto de dados, há registros sem a indicação do time do cliente, não preencher essa informação ou desconsiderá-la pode ser ruim para o modelo. Outro exemplo é quando temos alguma informação baseada em valores, como o preço de venda ou o total de compras, que, caso não sejam preenchidos, podem tornar o modelo deficiente. Agora, se forem preenchidos com valores como 0 ou, por algum erro nos dados, contenha valores negativos, isso também pode atrapalhar o processo de aprendizagem.

1.2 Extração de características

Após a conclusão do processo de limpeza, formatação e padronização dos dados, é hora de realizar a extração das características que serão entregues ao modelo. Aqui, além da influência na escolha das características por conta do problema que estamos tentando resolver, também se deve levar em conta o tipo de algoritmo que será utilizado para aprendizagem. As transformações sobre os dados que podem ser feitas não são claras e pode-se dizer que esse processo é um misto de ciência e arte, fortemente influenciado pela sua experiência com processos semelhantes.

Dentre os processos que são comumente realizados, podemos destacar:

- normalização ou escalonamento: o seu conjunto de dados pode conter atributos em diferentes unidades e escalas e isso pode ser prejudicial. Assim, deve-se considerar sempre que possível que os dados apresentem uma escala similar. Isso evita que valores absolutos muito altos criem tendências inadequadas no comportamento do modelo;
- agregação: algumas informações podem ser mais significativas se forem apresentadas de forma agregada e não isoladamente. Informações como a quantidade de ligações, ou de reservas, ou de compras que um determinado cliente fez podem ser apresentadas como uma contagem de ocorrências ao invés de registros isolados;
- decomposição: alguns dados podem fornecer informações mais importantes se forem decompostos em múltiplas partes. Um campo contendo um valor de data e hora em determinadas situações pode ser de maior relevância se o valor de hora, do mês ou do ano for considerado de forma isolada;
- composição: algumas características podem ser obtidas por meio de outros dados prévios. Uma diferença entre a data de reserva de um bilhete aéreo e a sua utilização efetiva pode ser mais útil para o processo de aprendizagem que apenas as datas isoladas. Da mesma forma, inferir o dia da semana no qual a reserva e a utilização do bilhete foram feitas, gerando um dado novo, pode ser de grande utilidade.

A etapa de extração de características demanda um bom tempo de dedicação, pois, como já citamos anteriormente, o seu modelo será tão bom quanto o conjunto de características que vamos escolher, sendo necessário realizar uma análise criteriosa dos dados disponíveis e de aspectos como temporalidade, localização e distribuição dos dados. Isso pode ser feito por meio da análise de relação entre os atributos e estudos exploratórios do conjunto de dados.

1.3 Separação dos conjuntos de treinamento e testes

Após a determinação das características que serão utilizadas, é necessário realizar a divisão dos dados em dois conjuntos: um para executar o treinamento do modelo de aprendizagem e outro para avaliar o seu desempenho.

É primordial evitar sobreposição entre esses conjuntos, para que dados utilizados em treinamento não figurem no conjunto de testes.

Uma regra comum na divisão dos dados entre conjunto de treinamento e testes é utilizar a proporção 80-20, com 80% dos dados para treinamento e 20% para testes. Esses conjuntos, idealmente, devem conter variabilidade de exemplares e devem ser fornecidos ao algoritmo de treinamento de forma que os vetores de características sejam de regiões distintas do espaço vetorial de características e não de regiões muito próximas umas das outras – contíguas.

Uma estratégia que pode ser utilizada é gerar os conjuntos de forma aleatória, mas mantendo a proporção. Esta pode ser uma boa estratégia, mas, como em todos os processos anteriores, deve ser considerado o problema a ser resolvido. Se o modelo apresenta dependência temporal em relação aos tempos mais recentes, de nada serviria incluir dados muito antigos no conjunto de testes. Considere situações também em que necessitamos avaliar diferentes regiões do país. Se no modelo for utilizado como treinamento dados de todo o país, mas no conjunto de testes haver dados de apenas algumas regiões ou estados, não teremos como avaliar a qualidade do modelo de forma mais ampla.

Considerar a variabilidade dos seus dados de treinamento e de testes é fundamental, mas devemos lembrar que, apesar de quantidade de dados não ser melhor que qualidade, ter poucos dados pode ser um problema. Para isso, há um artifício utilizado para situações em que a quantidade dos dados é pouca, chamado de *validação cruzada*. A estratégia consiste em dividir o conjunto de treinamento em K partes. Assim, podemos realizar o treinamento em K-1 etapas, no qual a cada etapa uma das partes é utilizada como o conjunto de testes e o restante como o conjunto de treinamento. Essa proporção também pode ser alterada. A seguir apresentamos um trecho de código Python que faz a carga e divisão de uma base de dados em um conjunto de treinamento e testes, bem como executa a divisão para validação cruzada.

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
import pandas as pd

 # Lê o dataset de carros
 carros = pd.read_csv("carros.csv")

# Seleciona apenas as colunas necessárias e separado
# em vetor de características e labels
```

```
X = carros.loc[:, 'ano':'preco_venda'].to_numpy()
y = carros['label'].to_numpy()

# Divide o conjunto em treinamento e teste, na proporção 80-20
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.20, random_state=42)

# Alternativamente, também pode dividir o conjunto em 3 folds
kf = KFold(n_splits=3)
idx_folds = kf.split(X)

for idx_treino, idx_teste in idx_folds:
  X_treino, X_teste = X[idx_treino], X[idx_teste]
  y_treino, y_teste = y[idx_treino], y[idx_teste]
```

Para finalizar a etapa de preparação dos dados, faz-se necessário reforçar que se exige que os dados sejam analisados de forma criteriosa. Devemos sempre analisar os dados e buscar descobrir suas particularidades, ao invés de simplesmente tentar aplicar estratégias de que não se sabe se serão frutíferas ou não. Isso pode poupar muito tempo de desenvolvimento e recursos, pois os processos de treinamento podem ser bem demorados e, quanto mais demorado, maior o custo computacional e, consequentemente, financeiro.

Uma vez que tenhamos os conjuntos de treinamento e de testes separados, com todas as ações de pré-processamento que consideramos necessárias executadas, podemos entregar os dados para o algoritmo de aprendizagem escolhido e treinar o modelo.

TEMA 2 – ÁRVORES DE DECISÃO

Nesta seção, apresentaremos o algoritmo de árvore de decisão e como este pode ser usado para o treinamento de um modelo de aprendizagem. As árvores de decisão são alguns dos algoritmos de aprendizagem mais conhecidos, bem como dos mais antigos, sendo muitas vezes referenciados como algoritmos clássicos.

O funcionamento do algoritmo de uma árvore de decisão imita a forma como um humano faria a separação de um largo conjunto de dados baseado em vários atributos. Por conta disso, esses algoritmos apresentam a vantagem de serem simples de entender, permitindo interpretar os seus resultados.

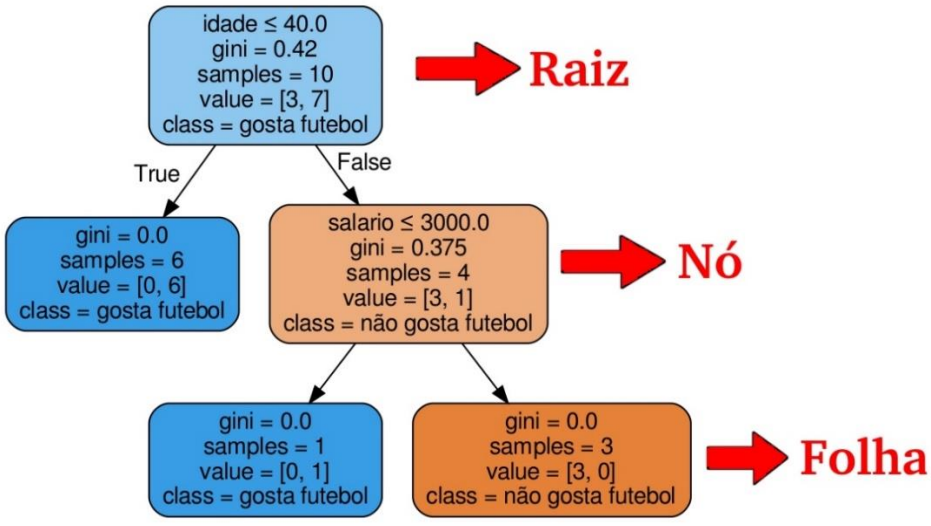
2.1 Como funciona

Uma árvore de decisão é uma estrutura na qual os nós representam os atributos ou características, as arestas representam as regras de decisão e cada folha da árvore representa uma categoria ou classe. Esse último caso se aplica quando a árvore é utilizada em processos de classificação. A ideia principal é processar todo o conjunto de dados e, aos poucos, dividir o processo de separação das instâncias. Na Tabela 1, temos um exemplo de uso de uma árvore de decisão para classificar um conjunto de dados de clientes de acordo com o seu gosto de futebol.

Tabela 1 – Exemplo de uso de uma árvore de decisão

	nome	idade	nacionalidade	altura	peso	sexo	gosta de futebol	salario
0	Aldo	22	0	1.70	75	0	1	2500
1	Beto	52	1	1.75	80	0	1	2500
2	Bia	31	1	1.50	65	1	1	4500
3	Carlos	65	1	1.95	86	0	0	6500
4	Diego	17	0	1.81	95	0	1	1500
5	Daina	54	0	1.65	80	1	0	3500
6	Carol	30	0	1.90	105	1	1	1200
7	Fabia	25	1	1.85	65	1	1	2245
8	Felix	49	0	1.71	75	0	0	25000
9	Gaia	26	1	1.81	80	1	1	8000

Figura 1 – Exemplo de árvore de decisão



No exemplo, o nosso vetor de características é formado por todos os dados que descrevem os clientes, com exceção do nome, por não se tratar de um valor numérico. Os valores dessa classe serão os dados fornecidos para a característica “gosta de futebol”. Cada um dos componentes da árvore pode ser descrito da seguinte forma:

- raiz: é o nó inicial da árvore, sendo nó-pai de todos os demais, representando todo o conjunto de dados;
- nó: são os subconjuntos que podem ser divididos em conjuntos menores ou em folhas;
- folhas: também podem ser referenciados como nós terminais e se caracterizam por não possuírem nenhum nó-filho.

Em cada nó da árvore, encontramos as seguintes informações:

- atributos utilizados para separação: no exemplo anterior, a idade com a regra “idade <= 40.0” no primeiro nível e “salario <= 3000.0” no segundo nível;
- samples: indica a quantidade de amostras que se enquadram na condição daquele nó específico;
- gini: é uma métrica que quantifica a pureza do nó. Um valor de gini igual a zero implica que as amostras daquele nó pertencem a uma mesma classe;
- value: é uma lista que indica quantas amostras se enquadram em cada uma das categorias;
- class: indica a categoria ou classe à qual o conjunto de amostras daquele nó pertence.

Podemos notar que nas folhas não há condições de divisão do conjunto de dados e que nos nós podemos encontrar regras baseadas em diferentes atributos. Assim, a árvore é montada de acordo com o conjunto de dados utilizados e o seu tamanho dependerá dos parâmetros utilizados na sua construção.

2.2 Tipos de algoritmos

Existem diferentes algoritmos que implementam o conceito das árvores de decisão, cada um deles com as suas características próprias e com

diferenças nas formas de determinar a separação e profundidade dos nós. As implementações mais comuns são:

- *Iterative Dichotomiser* 3 (ID3): desenvolvido por Ross Quinlan, em 1986, é um algoritmo que cria uma árvore de múltiplos caminhos, buscando o ganho máximo de informação para categoria. Ao final, é executado um processo de poda para dar ao modelo capacidade de generalização;
- C4.5: algoritmo sucessor do ID3 que removeu a restrição de os atributos serem categóricos, utilizando as árvores de saída do ID3 e convertendo-as em um conjunto de *if-elses*. O processo de poda foi substituído por remoção de regras *if-else*;
- C5.0: Evolução do C4.5, que utiliza menos memória e gera um conjunto de regras melhorado e com maior acurácia;
- *Classification and Regression Trees* (Cart): suporta objetivos numéricos, ou seja, pode ser utilizado tanto para classificação como para regressão, além de construir árvores binárias, baseadas nos atributos e nos ganhos de informação.

Utilizaremos a implantação Cart de árvores de decisão. Ao utilizar uma biblioteca que implemente algoritmos de aprendizagem, é preciso avaliar quais as implementações estão presentes nela, pois o tipo de implementação tem especificidades tanto no treinamento quanto na fase de predição.

TEMA 3 – TREINAMENTO DE UM MODELO DE ÁRVORE DE DECISÃO

Nesta seção, como já cumprimos as etapas de preparação dos dados, separação dos conjuntos de treinamento e de testes e escolha de um algoritmo para utilizar, estamos prontos para iniciar o treinamento de um modelo sobre o conjunto de dados.

3.1 Treinando um classificador

Um treinamento com um *dataset* desse tipo representa um problema real, com o qual vamos nos deparar ao trabalhar com dados reais. Todo o pré-processamento e a divisão dos subconjuntos de treinamento e testes foram feitos nesse conjunto de dados. A Tabela 2 apresenta as informações sobre os dados.

Tabela 2 – Informações sobre os dados

***Dataset* de automóveis**	
Total de registros do *dataset*	11.913
Tamanho do conjunto de treino	9.427
Tamanho do conjunto de testes	2.357
Quantidade de características	14
Quantidade de classes	48

A Tabela 3 exibe uma amostra do *dataset*, seus atributos e *labels*.

Tabela 3 – *Dataset* de dados de automóveis

ano	combustivel	hp	cilindros	tipo_transmissao	tracao	num_portas	categoria	tamanho	estilo	km_por_l_estrada	km_por_l_cidade	popularidade	preco_venda	label
2011	gasolina premium	335.0	6.0	MANUAL	tracao_traseira	2.0	Audio de Fabrica,Luxo,Alta-Performance	compacto	Cupê	26	19	3916	46135	BMW 1 Series M
2011	gasolina premium	300.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Performance	compacto	Conversível	28	19	3916	40650	BMW 1 Series
2011	gasolina premium	300.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Alta-Performance	compacto	Cupê	28	20	3916	36350	BMW 1 Series
2011	gasolina premium	230.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Performance	compacto	Cupê	28	18	3916	29450	BMW 1 Series
2011	gasolina premium	230.0	6.0	MANUAL	tracao_traseira	2.0	Luxo	compacto	Conversível	28	18	3916	34500	BMW 1 Series
2012	gasolina premium	230.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Performance	compacto	Cupê	28	18	3916	31200	BMW 1 Series
2012	gasolina premium	300.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Performance	compacto	Conversível	26	17	3916	44100	BMW 1 Series
2012	gasolina premium	300.0	6.0	MANUAL	tracao_traseira	2.0	Luxo,Alta-Performance	compacto	Cupê	28	20	3916	39300	BMW 1 Series
2012	gasolina premium	230.0	6.0	MANUAL	tracao_traseira	2.0	Luxo	compacto	Conversível	28	18	3916	36900	BMW 1 Series
2013	gasolina premium	230.0	6.0	MANUAL	tracao_traseira	2.0	Luxo	compacto	Conversível	27	18	3916	37200	BMW 1 Series

O trecho de código a seguir mostra um processo de classificação de um conjunto de dados de automóveis utilizando um algoritmo de árvore de decisão do tipo Cart. O objetivo do modelo é determinar por meio das características de um veículo o fabricante; assim, a coluna de *labels* contém um código para cada fabricante distinto que aparece no *dataset*.

```
import pandas as pd
from sklearn import tree
from sklearn.model_selection import train_test_split

# Lê o dataset de carros
carros = pd.read_csv("carros_categorical.csv")


# Seleciona apenas as colunas necessárias
X = carros.loc[:, 'ano':'preco_venda'].to_numpy()
y = carros['label'].to_numpy()

# Divide o conjunto em treinamento e teste, na proporção 80-20
```

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y,
                    test_size=0.20,
                    random_state=42)
# Executa a definição e treinamento do modelo
clf = tree.DecisionTreeClassifier(criterion = "gini", splitter = "best", max_depth = 5,
                     min_samples_split = 5, min_samples_leaf = 2,
                          max_features = 2)
clf = clf.fit(X_treino, y_treino)
```

É possível notar que, durante a definição do modelo a ser treinado `tree.DecisionTreeClassifier()`, vários parâmetros são passados para a chamada. Esses parâmetros se referem ao modelo e definem aspectos do comportamento do algoritmo escolhido. A seguir descrevemos cada um deles e como podem afetar o modelo obtido:

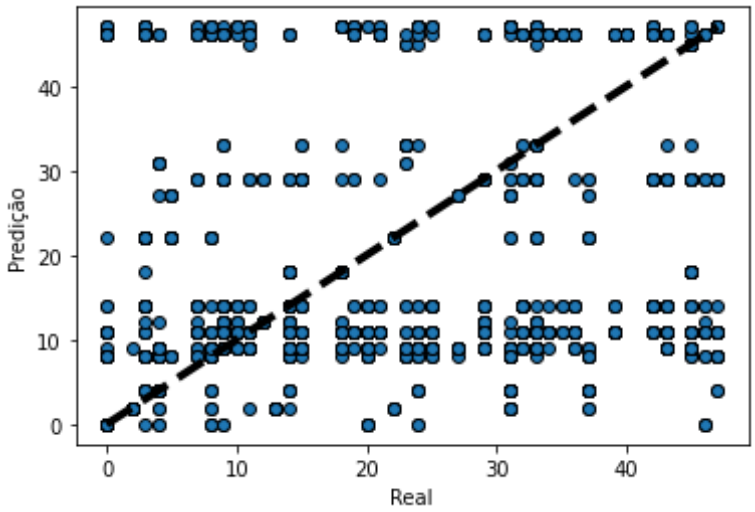
- criterion: define como a impureza das divisões será medida, sendo a mais comum a gini, mas também pode ser utilizado valor de entropia;
- splitter: define como o algoritmo deve buscar as características que utilizará para executar as divisões. As opções mais comuns são “best” e “random”, cada um dependendo do tamanho do vetor de características;
- max_depth: determina a profundidade máxima da árvore. Esse parâmetro deve ser definido de acordo com a necessidade e por meio do comportamento do modelo, pois profundidades muito grandes podem gerar modelos com *overfitting* (“sobreajuste”) dos dados, que é quando o modelo se ajusta aos dados de treinamento que é incapaz, ou faz com deficiência a predição em conjuntos de dados que nunca lhe foram apresentados;
- min_samples_split: quantidade mínima de uma amostra que um nó deve conter para que possa executar uma divisão;
- min_samples_leaf: quantidade mínima de amostras para considerar um nó como nó-folha, fazendo com o que este não sofra mais divisões. Esse parâmetro também pode ser usado para controlar o crescimento da árvore;
- max_features: número máximo de atributos que deve ser considerado ao executar a busca pela melhor divisão. Normalmente, recomenda-se utilizar todos os atributos, mas isso varia de acordo com o cenário, sendo

uma boa ideia fazer um ajuste fino nesse parâmetro para melhorar os resultados obtidos.

Agora que já temos um modelo treinado, precisamos avaliar o seu desempenho utilizando o conjunto de dados de teste. Durante esse processo é fornecido ao modelo treinado cada uma das instâncias de treinamento para que ele indique qual o fabricante daquele automóvel. Como temos os *labels* que indicam o fabricante correto, podemos comparar o valor cuja predição foi feita pelo modelo com o valor correto. Essa comparação permite avaliar o desempenho do modelo por meio de algumas métricas e direcionar um novo processo de treinamento. O código a seguir executa o processo de predição no conjunto de testes e gera um gráfico de comparação entre os valores corretos e valores da predição do modelo.

```
# Executa a predição do conjunto de testes
y_pred = clf.predict(X_teste)
# Plota gráfico de comparação
fig, ax = plt.subplots()
ax.scatter(y_teste, y_pred, edgecolors=(0, 0, 0))
ax.plot([y_teste.min(), y_teste.max()],
   [y_teste.min(), y_teste.max()], 'k--', lw=4)
ax.set_xlabel('Real')
ax.set_ylabel('Predição')
plt.show()
```

Figura 2 – Comparação entre os valores corretos e valores da predição do modelo



A Figura 2 indica um desempenho ruim do modelo, pois quanto mais próximo da linha diagonal estiverem os pontos do gráfico, melhor será o desempenho. Isso porque haverá coincidência entre o que o modelo está predizendo e o valor real. Assim, o modelo necessita de um novo processo de treinamento para melhorar o seu desempenho.

O processo de treinamento deve ser conduzido por meio da avaliação constante do seu desempenho e, para isso, são necessárias métricas que indiquem esse desempenho do modelo. Isso porque, ainda que saibamos que o desempenho é ruim, não temos uma noção clara de como e por quê o modelo estava produzindo resultados incorretos.

3.2 Avaliando o desempenho de um modelo

Uma das medidas mais comuns que podem ser utilizadas para avaliar a qualidade de um modelo é acurácia. Ela indica o percentual de concordância entre os valores preditos pelo modelo e os valores reais do conjunto de dados de teste. Ainda que os valores de acurácia indicados por um modelo sejam altos, esse modelo pode apresentar um desempenho ruim quando apresentado a dados diferentes daqueles do conjunto de testes. Assim, para ter um entendimento melhor do desempenho de um modelo, duas outras métricas são comumente utilizadas: precisão e *recall*[1].

Antes de definirmos o que é precisão e *recall*, precisamos definir como os resultados de classificador podem ser categorizados. O Quadro 1 apresenta as possibilidades de classificação para cada uma das categorias de um modelo.

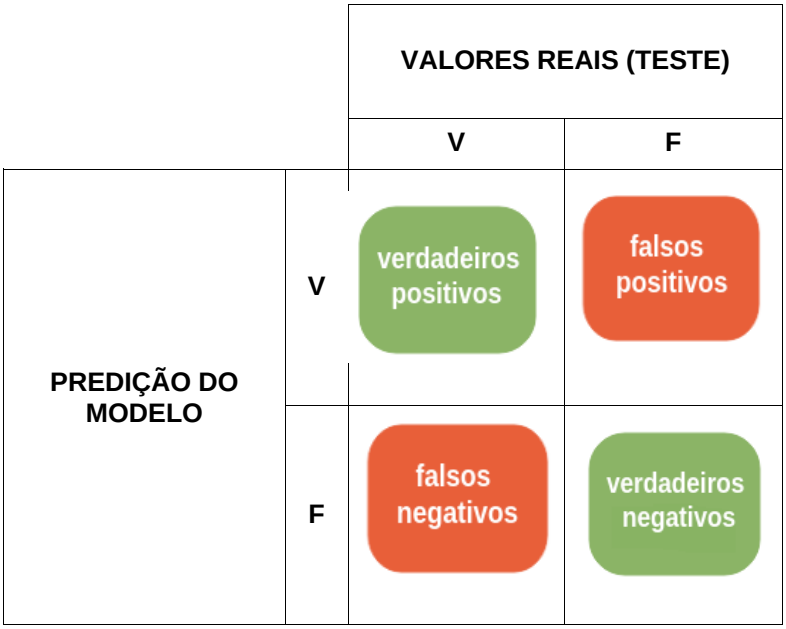
Quadro 1 – Possibilidades de classificação para cada uma das categorias de um modelo

DEFINIÇÃO	SIGNIFICADO
Verdadeiros positivos	Instâncias preditas como positivas pelo modelo, que realmente são positivas.
Verdadeiros negativos	Instâncias preditas como negativas pelo modelo, que realmente são negativas.
Falsos positivos (ou erro tipo I)	Instâncias preditas como positivas pelo modelo, que na verdade são negativas.

[1] Em estatística, é comum encontrarmos o termo *sensibilidade* ou *revocação*.

falsos negativos (ou erro tipo II)	Instâncias preditas como negativas pelo modelo, que na verdade são realmente são positivas.

Figura 3 – Predição do modelo e valores reais



Os conceitos de verdadeiro e falso, quando relacionados à predição do modelo, indicam respectivamente quando o modelo prediz que uma instância pertence à uma classe e quando não pertence. Quando temos diversas classes em um problema, essa avaliação é feita para cada uma das categorias em análise.

Por meio da categorização dos resultados que o modelo pode fornecer, podemos definir de maneira mais formal os conceitos de precisão, *recall* e acurácia conforme a seguir.

Figura 4 – Conceitos

$$\text{Precisão} = \frac{\text{verdadeiros positivos}}{\text{verdadeiros positivos} + \text{falsos positivos}}$$

$$\textit{Recall} = \frac{\text{verdadeiros positivos}}{\text{verdadeiros positivos} + \text{falsos negativos}}$$

$$\text{Acurácia} = \frac{\text{verdadeiros positivos} + \text{verdadeiros negativos}}{\text{total de instâncias}}$$

A precisão indica o percentual dos resultados obtidos que são relevantes. Já o *recall* indica o percentual do total de resultados relevantes classificados de forma correta pelo modelo, de maneira que um modelo bem treinado deve buscar um equilíbrio entre precisão e *recall*. Uma maneira de avaliar um modelo considerando essas duas métricas é utilizar o escore F1, que é definido como a média harmônica entre a precisão e o *recall*, conforme a equação a seguir:

$$F1 = 2 \times \frac{\text{precisão} * \text{recall}}{\text{precisão} + \text{recall}}$$

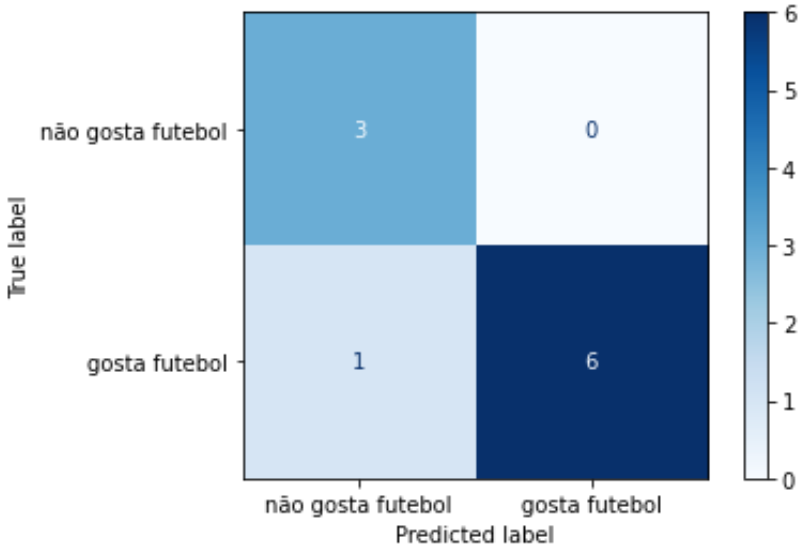
Seguindo o exemplo das métricas anteriormente apresentadas em um problema com apenas duas classes, considere o exemplo de um modelo treinado no conjunto de dados de clientes para determinar aqueles que gostam e não gostam de futebol. Os valores das métricas são os seguintes (Tabela 4):

Tabela 4 – Valores das métricas

	precisão	***recall***	**f1-score**
Não gosta futebol	0.67	0.67	0.67
Gosta futebol	0.86	0.86	0.86

Além dos valores das métricas apresentadas, outra ferramenta para avaliar o desempenho do modelo é a matriz de confusão. Essa matriz apresenta os resultados quantitativos dos valores reais das classes das instâncias e os valores preditos pelo modelo. Assim, os valores dos resultados do modelo de classificação de clientes que gostam de futebol e de que não gostam será apresentado conforme mostra os valores da Figura 5.

Figura 5 – Matriz de confusão de modelo de classificação de clientes



De acordo com as métricas e com o que podemos ver na matriz de confusão, o modelo se confunde com uma instância, gerando um falso negativo. Por esse motivo, temos um valor de f1-score (0.67) para a classe “não gosta futebol”, sendo um bom indicador de um novo treinamento do modelo ou de uma acréscimo de dados para melhorar a qualidade da predição.

3.2 Melhorando um modelo

Na Seção 3.1, executamos o processo de treinamento de um modelo de predição de fabricantes de carros, cujo desempenho não pareceu ser suficiente quando avaliado com o conjunto de dados de testes. Os resultados ruins das predições indicaram que seria necessário um novo treinamento do modelo, assim o procedimento a se fazer é buscar novos parâmetros que permitam uma melhora daquele modelo.

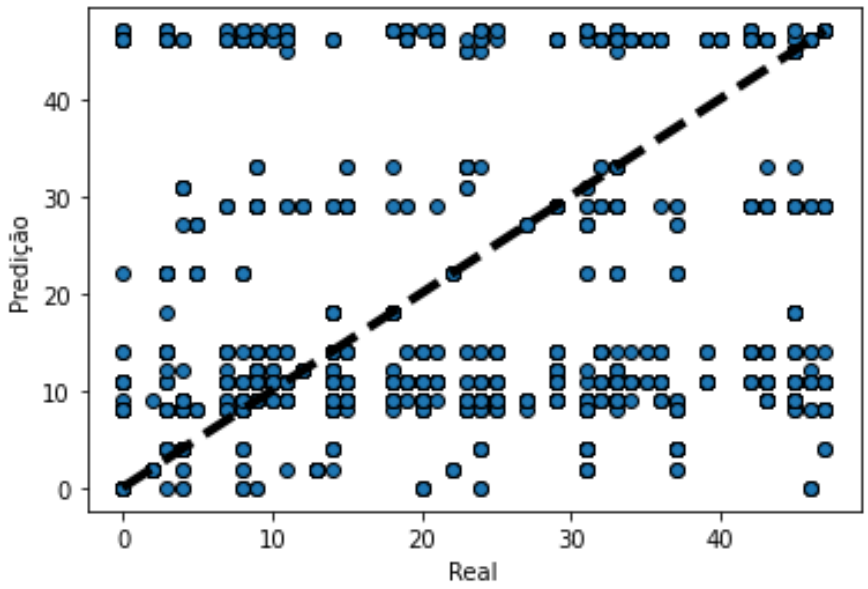
Para ilustrar as diferenças que se pode obter ao alterar os parâmetros de definição de um modelo de aprendizagem, apresentamos a seguir dois cenários, com duas listas de diferentes parâmetros de um mesmo modelo treinado em um único conjunto de dados.

3.2.1 Modelo 1

- **Modelo Árvore de Decisão**
- Parâmetros:

- criterion: gini
- splitter: best
- max_depth: 5
- min_samples_split: 5
- min_samples_leaf: 2
- max_features: 2

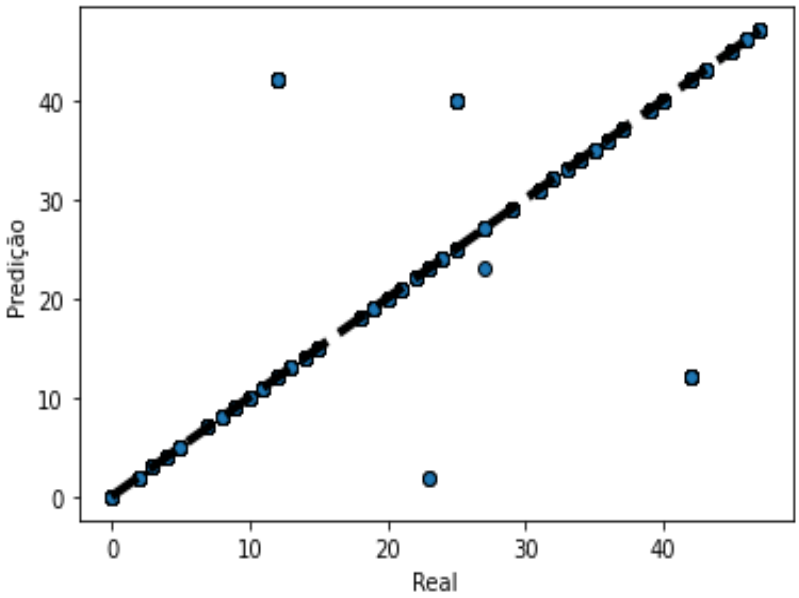
Figura 6 – Modelo 1



3.2.2 Modelo 2

- **Modelo Árvore de Decisão**
- Parâmetros:
  - criterion: gini
  - splitter: best
  - max_depth: 15
  - min_samples_split: 5
  - min_samples_leaf: 2
  - max_features: 14

Figura 7 – Modelo 2



A melhora nos resultados de predição do modelo é considerável, tendo sido obtida pela alteração de dois parâmetros: a profundidade da árvore a ser gerada e a quantidade de atributos a ser considerada na divisão dos nós da árvore. Esse processo de avaliação e busca de um melhor modelo é um trabalho comum em aprendizagem de máquina, sendo necessário muito cuidado e critério na busca de melhores modelos, enquanto se tenta manter um equilíbrio entre capacidade de acertar as predições e a capacidade de generalização para dados nunca antes vistos pelo modelo.

TEMA 4 – UTILIZANDO UM MODELO TREINADO

Agora que dispomos de um modelo treinado e com resultados eficientes, necessitamos armazenar o modelo para uso posterior, pois, do contrário, sempre que fôssemos executar um processo de predição seria necessário treinar o modelo novamente. Há diversos meios de armazenar os modelos treinados, porém, uma maneira simples e direta do Python é a serialização que executa a conversão de um objeto da linguagem em um arquivo binário. Esse método permite que modelos treinados sejam reutilizados, bem como compartilhados.

4.1 “Serializando” um classificador

Processos de “serialização” são comuns em diversas linguagens de programação que oferecem formas distintas de obter esse resultado. No Python,

a serialização é feita por meio da biblioteca Pickle. O código a seguir mostra como fazer o processo de serialização do modelo de árvore de decisão treinado para classificar automóveis.

```
import pickle

with open("clf_automoveis_dt.pickle", 'wb') as f:
pickle.dump(clf, f, pickle.HIGHEST_PROTOCOL)
```

O processo de serialização feito com Pickle é simples, assim como o processo de "desserialização". Contudo, cuidados adicionais devem ser tomados durante o processo de carregamento de um modelo, pois devemos ter certeza de sua origem, sob o risco de executarmos código inapropriado em algum processamento utilizando o objeto "desserializado".

Para permitir a utilização e manutenção do modelo "serializado", é recomendado que algumas informações sejam gravadas no objeto, por exemplo:

- os dados utilizados no treinamento do modelo, pois dessa forma será possível refazer os passos de treinamento e acrescentar dados novos ou explorar os dados anteriormente utilizados;
- código Python utilizado na geração do modelo;
- versão do scikit-learn ou da biblioteca utilizada para gerar o modelo, bem como as suas dependências;
- valores das métricas obtidas no treinamento do modelo;
- versão da biblioteca de serialização utilizada.

O processo de desserialização também é simples de ser efetuado, conforme podemos ver no código a seguir, em que é feito o processo de desserialização do classificador.

```
import pickle
from sklearn import tree

clf = None
with open("clf_automoveis_dt.pickle", 'rb') as f:
clf = pickle.load(f)
```

4.2 Outras formas de serialização

A serialização dos modelos treinados pode ser feita de diversas maneiras. Dentre as outras possiblidades de armazenamento dos modelos de classificação

está o uso da outra biblioteca do Python chamada Joblib. A serialização utilizando essa biblioteca é vantajosa comparada com o método utilizado pela biblioteca Pickle, quando o modelo possui *arrays numpy* de grandes dimensões. O código a seguir mostra podem ser realizados os processos de serialização e desserialização utilizando a Joblib.

```
from sklearn.externals import joblib

# Salva o arquivo serializado do classificador
joblib_file = "clf_automoveis_dt.pkl"
joblib.dump(clf, joblib_file)

# Carrega o classificador a partir de um arquivo
clf_model = joblib.load(joblib_file)

# Calcula a acurácia e executa predição
score = clf_model.score(X_teste, y_teste)
y_pred = clf_model.predict(X_teste)
```

Outra forma de executar a serialização é por meio do uso de JSON. Esse método permite um controle maior sobre os processos de serializar e restaurar os objetos, podendo ser muito útil em situações a que tanto o Pickle quanto a Joblib não atendam. Uma opção que permite obter consistência do processo é estender o classificador e implementar os métodos de salvar e carregar o JSON, assim, obtém-se uma padronização. Todavia, o uso de JSON como forma de serialização pode representar um risco à segurança, pois o arquivo gerado no processo pode facilmente ser alterado em qualquer editor de texto. Por isso, é recomendado que, no momento da geração do arquivo, seja produzido um resumo (*hash*) identificador deste, como forma de garantir a sua integridade.

TEMA 5 – MODELOS DE ÁRVORES DE DECISÃO COMBINADOS

Os modelos baseados em árvore de decisão são simples de treinar e também de refinar, permitindo uma interpretação e avaliação da obtenção dos seus resultados. Entretanto, seu desempenho em problemas nos quais a quantidade de atributos seja muito grande pode ser proibitivo, ainda que a sua velocidade de execução e de treinamento sejam pontos fortes.

Uma opção de uso de árvores de decisão para problemas mais complexos pode ser por meio de um algoritmo de aprendizagem que aproveita os pontos fortes das árvores de decisão e expande esse modelo: as florestas randômicas, ou *random forest*. Esse tipo de modelo se baseia em uma forma de combinar

modelos de predição conhecidos como *ensemble*, que podem ser aplicados tanto para árvores de decisão como para diferentes modelos de predição.

5.1 *Ensembles* de classificadores

Ensembles são métodos que combinam diferentes modelos em um único modelo preditivo, de maneira que todos os modelos gerados atuem como se fosse um único, apesar das predições serem executadas em separado em cada um deles e combinadas para gerar a predição (ou regressão) final.

Os *ensembles* podem ser divididos em duas categorias distintas:

- métodos sequenciais: os modelos de predição atuam de forma sequencial e são combinados de forma a explorar a dependência entre si;
- métodos paralelos: os modelos de predição atuam de forma paralela e são combinados de forma a explorar a independência entre si.

Dependendo da necessidade, os *ensembles* podem ser utilizados para reduzir a variância e, dessa forma, reduzir o *overfitting* do modelo. Essa forma de *ensemble* paralelo é conhecida como *bagging*. Um exemplo muito utilizado desse tipo são as florestas randômicas.

Para reduzir o *underfitting*, por meio da redução do viés (bias) utiliza-se a forma de *ensemble* sequencial do tipo *boosting*. Um exemplo muito utilizado de *ensemble* do tipo *boosting* é o algoritmo AdaBoost. Já para melhorar as predições por meio do uso de *ensembles*, utiliza-se um método denominado *stacking*, em que as saídas dos modelos utilizados no *ensemble* são utilizadas para o treinamento do meta-classificador. Diferentemente dos outros métodos de *ensemble*, todo o conjunto de dados de treino é utilizado pelos modelos que fazem parte da combinação.

Apesar de ser possível combinar diferentes algoritmos de predição, em geral, o mesmo algoritmo é utilizado, com variações nos conjuntos de dados de treino e nos atributos utilizados, bem como nos parâmetros de definição do modelo. Os *ensembles* que utilizam o mesmo algoritmo são referenciados como *ensembles* homogêneos, enquanto aqueles que combinam algoritmos diferentes são referenciados como *ensembles* heterogêneos.

5.2 Florestas randômicas

Random Forest é um classificador que cria um conjunto de árvores de decisão e executa os processos de treinamento desse conjunto, como também utiliza as suas predições para compor o seu resultado final de predição. Esse tipo de técnica é referenciado como *ensemble* de classificadores, em que um conjunto de classificadores é criado, treinado e posto em produção como se fosse um único classificador.

Ao utilizar diferentes classificadores em conjunto, é possível observar diferentes aspectos dos atributos e treinar classificadores especializados em um conjunto de atributos ou em uma porção específica dos dados, aumentando a capacidade de generalização do modelo. Uma desvantagem das florestas de árvores de decisão é o fato de sua interpretação ser difícil por um humano, pois em geral a quantidade de árvores é muito grande, bem como a quantidade de regras, diferentemente de um único modelo de árvore de decisão isolado. Outro aspecto que deve ser considerado é que o tempo de treinamento de um modelo *random forest* é consideravelmente maior que um modelo de árvore de decisão, pois obviamente são treinados diversos modelos de árvores.

5.3 Outros *ensembles* de árvores

Além de *random forest*, vários outros métodos de combinação de classificadores baseados em árvores de decisão podem ser utilizados. Entre as combinações e variações que podemos encontrar, temos:

- *Isolation Forest*: algoritmo baseado em árvores de decisão utilizado para detecção de instâncias anômalas de um conjunto de dados;
- *Extra Trees Classifier*: um metaclassificador que utiliza um conjunto aleatório de árvores de decisão treinados em amostras obtidas dos dados de treinamento e utiliza a média das predições como forma de melhorar o desempenho do modelo;
- AdaBoost com Árvore de Decisão: usa-se o algoritmo AdaBoost para treinar uma árvore de decisão no conjunto completo de dados e, em seguida, treina-se cópias adicionais do modelo de árvore utilizando pesos para as instâncias que não foram classificadas corretamente.

Os algoritmos de árvores de decisão, sejam utilizados sozinhos ou *ensembles*, podem ser utilizados tanto para tarefas de classificação como para tarefas de regressão que exijam a predição de valores contínuos. Enfim, os algoritmos de árvore de decisão podem ser utilizados de forma rápida e com um custo computacional compatível com os resultados que entregam. São uma boa ferramenta, tanto para exploração inicial quanto para utilização em produção como modelo preditivo.

FINALIZANDO

Nesta aula, apresentamos as etapas de preparação de dados necessárias para o treinamento de modelos de aprendizagem supervisionada, ressaltando a importância da qualidade dos dados no modelo de aprendizagem obtido. Também foram apresentados os conceitos e a forma de funcionamento dos algoritmos de árvore de decisão, seu treinamento, utilização e exportação para reutilização futura. Destaque especial foi dado para as maneiras como os algoritmos de predição podem ser combinados por meio de *ensembles* para gerar melhores predições, reduzir *overfitting* e *underfitting*. Ainda vimos que há diferentes algoritmos de árvores de decisão que permitem abordagens diferentes dos problemas de predição.

Todos os conceitos apresentados nesta aula são comuns aos processos de treinamento de outros algoritmos de aprendizagem, bem como de utilização, sendo de fundamental importância o entendimento desses conceitos para os profissionais de aprendizagem de máquina.

REFERÊNCIAS

CUESTA, H. **Practical data analysis**. Birmingham: Packt Publishing Ltd, 2013.

GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow**: Concepts, tools, and techniques to build intelligent systems. Sebastopol: O'Reilly Media, 2019.

MCKINNEY, W. **Python para análise de dados**: tratamento de dados com Pandas, NumPy e IPython. São Paulo: Novatec, 2019.

RICHERT, W. **Building machine learning systems with Python**. Birmingham: Packt Publishing Ltd, 2013.

RUSSEL, S. J.; NORVIG, P. **Inteligência Artificial**: uma abordagem moderna. 3. ed. Tradução de Regina Célia Simille. Rio de Janeiro: Elsevier, 2013.