Capítulo 1

Machine Learning aplicado à Saúde

André Filipe de Moraes Batista, Alexandre Dias Porto Chiavegatto Filho

Laboratório de Big Data e Análise Preditiva em Saúde (LABDAPS)
Faculdade de Saúde Pública da Universidade de São Paulo

Abstract

The increase in the complexity of computationally-treated challenges as well as the large amount of data generated by different activities fosters the need for sophisticated computational tools that are capable of inducing knowledge from past experience to solve real problems, an area known as Machine Learning. These algorithms learn to induce a function or hypothesis capable of solving a problem from previous data that represent instances of the problems to be solved. In the area of Health Informatics, the data available in electronic health records can serve as input for Machine Learning models aimed at improving diagnoses and predictions, leading to better treatments for patients. This mini-course will present the current landscape of the use of Machine Learning techniques applied in the health area, contextualizing the main algorithms currently used

Resumo

O aumento na complexidade dos problemas a serem tratados computacionalmente, assim como no volume de dados gerados por diferentes atividades, fomenta a necessidade de ferramentas computacionais sofisticadas que sejam capazes de induzir uma hipótese a partir da experiência passada para solucionar problemas reais. A esse cenário dá-se o nome Machine Learning. Algoritmos de Machine Learning aprendem a induzir uma função ou hipótese capaz de resolver um problema a partir de dados que representam observações dos problemas a ser resolvido. Na área de Informática em Saúde, os dados disponibilizados nos registros eletrônicos podem servir de insumos para modelos de Machine Learning voltados para o aprimoramento de diagnósticos e predições, possibilitando um melhor tratamento para os doentes. Este minicurso irá apresentar o panorama atual do uso das técnicas de Machine Learning aplicadas na área de saúde, contextualizando os principais algoritmos utilizados.

1.1. Introdução

Nos últimos anos a ciência tem se deparado com uma enorme quantidade de dados provenientes das mais diversas fontes como, por exemplo, a Internet, as mídias sociais, os dispositivos de coleta de dados, etc. Diversas agências de pesquisas estão financiando projetos que buscam promover a gestão, o acesso e a descoberta de conhecimento a partir de tais dados. O cenário formado por amplos conjuntos de dados em larga escala, normalmente gerados e consumidos em tempo real é conhecido como "dilúvio de dados" (em inglês, *Data Deluge*) (SELTZER; ZHANG, 2009; MATTMANN, 2013).

O termo Big Data tem sido utilizado para representar esta metáfora. De acordo com Mattmann (2013), existem três dimensões que classificam um conjunto de dados como big data, sendo elas: o *volume* dos dados que um sistema recebe, processa e/ou dissemina; a *variedade*, isto é, o número e a complexidade dos tipos de dados manipulados; e a *velocidade* com o qual os dados são criados e/ou disponibilizados para outros usos.

O uso de Big Data em saúde é a grande novidade da área nos últimos anos (CHIAVEGATTO FILHO, 2015). Hospitais e governos têm acumulado uma quantidade imensa de dados sobre diagnósticos clínicos e tratamento de doenças. Há a necessidade para que esses dados sejam utilizados para o benefício da saúde dos pacientes, com o objetivo de melhorar a tomada de decisão e contribuir para a melhoria da saúde pública.

Devido aos grandes e crescentes volume de dados, a Inteligência Artificial (IA) ganhou notória popularidade nas últimas décadas. Machine Learning (ML) é uma das áreas da Inteligência Artificial que vem apresentando maior destaque na área da saúde e em outras áreas do conhecimento, apoiando-se na teoria de que os computadores podem aprender com dados sem programação manual para executar determinadas atividades.

Tendo em vista a sua inerente complexidade, a atenção à saúde ainda é dominada por incertezas, com frequentes mudanças de protocolos e práticas clínicas. O uso de modelos preditivos de machine learning tem o potencial de auxiliar na tomada de decisão nos diversos momentos da atenção à saúde, especialmente no diagnóstico, intervenção e acompanhamento de problemas de saúde (OBERMEYER; LEE, 2017).

De acordo com Obermeyer e Emanuel (2016), os modelos de Machine Learning abordam problemas de maneira análoga a médicos que estão no período de residência: aprendendo as regras a partir dos dados. Tais modelos iniciam com observações no nível do paciente, percorrendo um grande número de variáveis, na busca por predizer resultados de maneira confiável. Tal processo não é muito diferente de um modelo tradicional de regressão: há um resultado a ser modelado, covariáveis e uma função estatística relacionando-os. Porém, o que faz dos modelos de Machine Learning promissores é a capacidade de lidar com um grande número de preditores – às vezes, inclusive, superior ao número de observações – e combiná-los de maneiras não-lineares e altamente interativas, obtendo resultados superiores aos métodos tradicionais.

Este Capítulo tem por objetivo apresentar os principais conceitos de Machine Learning, juntamente com a implementação de modelos de aprendizado supervisionados, aplicados em banco de dados sintéticos para a predição de risco cardiovascular (LADERAS et al., 2018). Os modelos foram implementados fazendo-se uso da biblioteca Scikit-Learn (BUITINCK et al., 2013) da linguagem de programação Python, sobre a qual os

principais fundamentos são apresentados.

1.2. Panorama do uso de Machine Learning na Área da Saúde

Diversos estudos têm sido desenvolvidos no âmbito de machine learning aplicado à saúde. O crescimento das pesquisas nessa área ocorrem conjuntamente ao aumento da demanda por métodos que possam facilitar diagnósticos e otimizar o tempo dos profissionais de saúde.

Weng et al. (2017) objetivaram comparar modelos resultantes de quatro algoritmos de machine learning com recomendações estabelecidas pelo Colégio Americano de Cardiologia para predizer doença cardiovascular em 10 anos, utilizando dados de uma coorte prospectiva de 378.256 pacientes. O modelo de machine learning de redes neurais apresentou o melhor desempenho dentre os modelos analisados, resultando em 355 predições corretas adicionais de doença cardiovascular quando comparado ao modelo baseado nas recomendações do Colégio Americano de Cardiologia.

O estudo de Green (2018) teve como objetivo compreender os determinantes de saúde utilizando machine learning para construir modelos preditivos para a ocorrência de doenças (e.g., hipertensão) em 1 e em 5 anos a partir do baseline. O modelo desenvolvido fez uso dos dados da pesquisa social *Understanding Society*, que coletou dados pessoais, sociais, de saúde, biomarcadores e genéticos de cerca de 6800 indivíduos. Os resultados obtidos indicam que o uso de dados de saúde para a construção de modelos preditivos apresentou o melhor resultados dentre os testes realizados, com uma acurácia de predição de 71%. Dentre os dados de saúde, a atividade física e a presença de algumas condições de saúde foram fortes preditores individuais.

No contexto brasileiro, Olivera et al. (2017) desenvolveram modelos preditivos de diabetes não diagnosticada a partir de dados de 12.447 adultos entrevistados para o Estudo Longitudinal de Saúde do Adulto (ELSA), utilizando cinco algoritmos de machine learning. A frequência de diabetes não diagnosticada foi de 11%. Entre os 403 indivíduos do conjunto de dados de teste que tinham diabetes não diagnosticada, 274 foram identificados como casos positivos.

1.3. Machine Learning

O aumento na complexidade dos problemas a serem tratados computacionalmente, assim como no volume de dados gerados por diferentes atividades, fomenta a necessidade de ferramentas computacionais sofisticadas que sejam capazes de induzir uma hipótese a partir da experiência passada para a solução de problemas reais. A este cenário dá-se o nome de Machine Learning (ML).

Por meio das técnicas de Machine Learning, computadores são programados para aprender com a experiência passada. Para isso, empregam um princípio de inferência denominado indução. Dessa forma, algoritmos de ML aprendem a induzir uma função ou hipótese capaz de resolver um problema a partir de dados que representam observações do problema a ser resolvido (FACELI et al., 2011).

Além do volume de aplicações que se beneficia das técnicas de ML, outros fatores têm favorecido a expansão dessa área, como o desenvolvimento de algoritmos cada vez

mais eficazes e eficientes, e a elevada capacidade dos recursos computacionais atualmente disponíveis.

As tarefas de ML podem ser divididas em duas categorias: **preditivas** e **descritivas**. Para as preditivas, a meta é encontrar uma função a partir dos dados de treinamento que possa ser utilizada para prever um rótulo ou valor que caracterize um novo exemplo, com base nos valores de seus atributos de entrada. Os algoritmos utilizados nessa tarefa seguem o paradigma de **aprendizado supervisionado**. Esse termo vem da simulação da presença de um "supervisor externo", que conhece a saída desejada para cada exemplo. Com isso, o supervisor externo pode avaliar a capacidade da hipótese induzida de predizer o valor de saída para novos exemplos (FACELI et al., 2011).

Em tarefas descritivas, a meta consiste na exploração ou descrição de um conjunto de dados. Os algoritmos utilizados nessas tarefas não fazem uso do atributo de saída (variável de interesse). Tais algoritmos seguem o paradigma de **aprendizado não supervisionado**. Uma tarefa descritiva de agrupamento de dados, por exemplo, tem por meta encontrar grupos de objetos semelhantes no conjuntos de dados (FACELI et al., 2011).

Os métodos de ML consistem em algoritmos computacionais para relacionar todos ou alguns elementos de um conjunto de variáveis preditoras a um resultado. Para estimar o modelo, eles buscam de forma estocástica ou determinística o melhor ajuste. Esse processo de busca por um melhor ajuste do modelo aos dados difere entre os algoritmos existentes. No entanto, ao longo desse processo, cada algoritmo tenta equilibrar dois interesses: **viés** (*bias*) e **variância**. O viés é o ponto até o qual as previsões ajustadas pelo modelo correspondem aos valores verdadeiros. Um modelo com alto viés pode não ter a complexidade necessária para classificar as observações corretamente. Variância é a sensibilidade das previsões na presença de alguma alteração nos dados de entrada. Embora se busca reduzir tanto o viés quanto a variância, esses dois objetivos geralmente estão em conflito: o viés diminuído pode aumentar a variância e vice-versa. Por exemplo, pode-se criar um algoritmo que prevê corretamente todas as mortes em um conjunto de dados. No entanto, esse modelo pode estar vinculado demasiadamente aos detalhes individuais do conjunto de dados de treinamento (baixo viés), modelando fortemente o 'ruído estatístico'. O modelo teria um desempenho ruim quando aplicado em um novo conjunto de dados, apresentando portanto uma alta variância (FACELI et al., 2011).

Este capítulo irá apresentar métodos supervisionados utilizados para a modelagem preditiva de respostas quantitativas e categóricas de interesse para a área da saúde. No aprendizado supervisionado, um conjunto de dados dispõe de n observações, cada uma representada por um vetor de mensurações das m variáveis independentes (ou variáveis preditoras), bem como da mensuração correspondente da variável dependente (resposta de interesse ou *output*). Quando a resposta de interesse é uma variável quantitativa, trata-se de um modelo de *regressão*. Quando se tratar de uma variável qualitativa ou categórica, tem-se um modelo de *classificação*. A maior parte dos modelos de aprendizado supervisionado podem ser aplicados tanto para a predição de variáveis qualitativas quanto para variáveis quantitativas.

Qualquer que seja o tipo da variável de interesse, o algoritmo supervisionado modela uma função de maneira indutiva, isto é, a partir das informações contidas nos dados

utilizados para treinamento. O objetivo é predizer o valor da variável de interesse da maneira mais precisa possível, além de dotar o modelo de uma boa capacidade de generalização, isto é, ter um bom desempenho diante de dados que não foram utilizados para seu treinamento.

Ao longo do aprendizado, os parâmetros da função preditora são ajustados aos dados, de modo a minimizar o erro de predição e aumentar a capacidade de generalização. Além dos parâmetros automaticamente definidos pelos algoritmo, existem também os **hiperparâmetros**. Os hiperparâmetros de um algoritmo devem ser ajustados de modo que o algoritmo evite o sobreajuste (em inglês, *overfitting*), quando o modelo gerado se torna muito especializado no conjunto de treinamento, obtendo baixo desempenho quando confrontado com novos dados. O contrário também é necessário ser evitado, ou seja, quando o algoritmo não se ajusta adequadamente ao padrão, obtendo um baixo desempenho frente ao conjunto de treinamento (*underfitting*). Dessa forma, busca-se minimizar a probabilidade de erros de predição para dados futuros, reduzindo o viés indutivo, assim como reduzindo a variância do mesmo quando da previsão de novos casos.

O trabalho de Olson et al. (2017) buscou avaliar 13 algoritmos de classificação sob 165 conjuntos de dados na área de bioinformática de modo a prover um conjunto de recomendações para o uso de modelos de preditivos. Os modelos preditivos baseados em árvores e que fazem uso das técnicas de comitê de modelos (*ensamble*) apresentaram os melhores resultados. Os autores enfatizam que o processo de ajuste dos hiperparâmetros dos modelos preditivos pode contribuir significativamente para o desempenho dos modelos preditivos.

1.3.1. Processo de Aplicação das Técnicas de Aprendizado Supervisionado

Faz-se necessário apresentar o *workflow* utilizado para a aplicação das técnicas de ML, ilustrado na Figura 1.1. Inicialmente, o conjunto de dados é dividido em dois sub-conjuntos: treinamento e teste. O sub-conjunto de treinamento será utilizado para o ajuste do modelo preditivo aos dados, enquanto que o sub-conjunto de teste é utilizado para verificar a capacidade de generalização do modelo diante de novas observações. Ao longo do ajuste e avaliação do modelo preditivo, apenas o sub-conjunto de treinamento é apresentado ao modelo, ficando como último passo a aplicação do conjunto de teste, para a obtenção de uma estimativa mais precisa sobre o desempenho da predição futura do modelo.

Os dados originais em sua forma bruta normalmente precisam ser adequados aos modelos preditivos, de modo a otimizar o desempenho de tais modelos. A tais tarefas dá-se o nome de pré-processamento dos dados. Entre as principais tarefas de pré-processamento, estão incluídas (FACELI et al., 2011; SAKR et al., 2017; GÉRON, 2017; KUHN; JOHNSON, 2013):

- Transformação dos dados originais: boa parte dos algoritmos de predição apresenta melhor resultado quando as variáveis preditoras estão sob uma mesma escala. Para isto, técnicas de padronização (como a normalização z-score) podem ser aplicadas, em que os valores originais da variável são redimensionados para terem média igual a 0 e desvio-padrão igual a 1. A partir da análise da estatística de simetria dos dados,

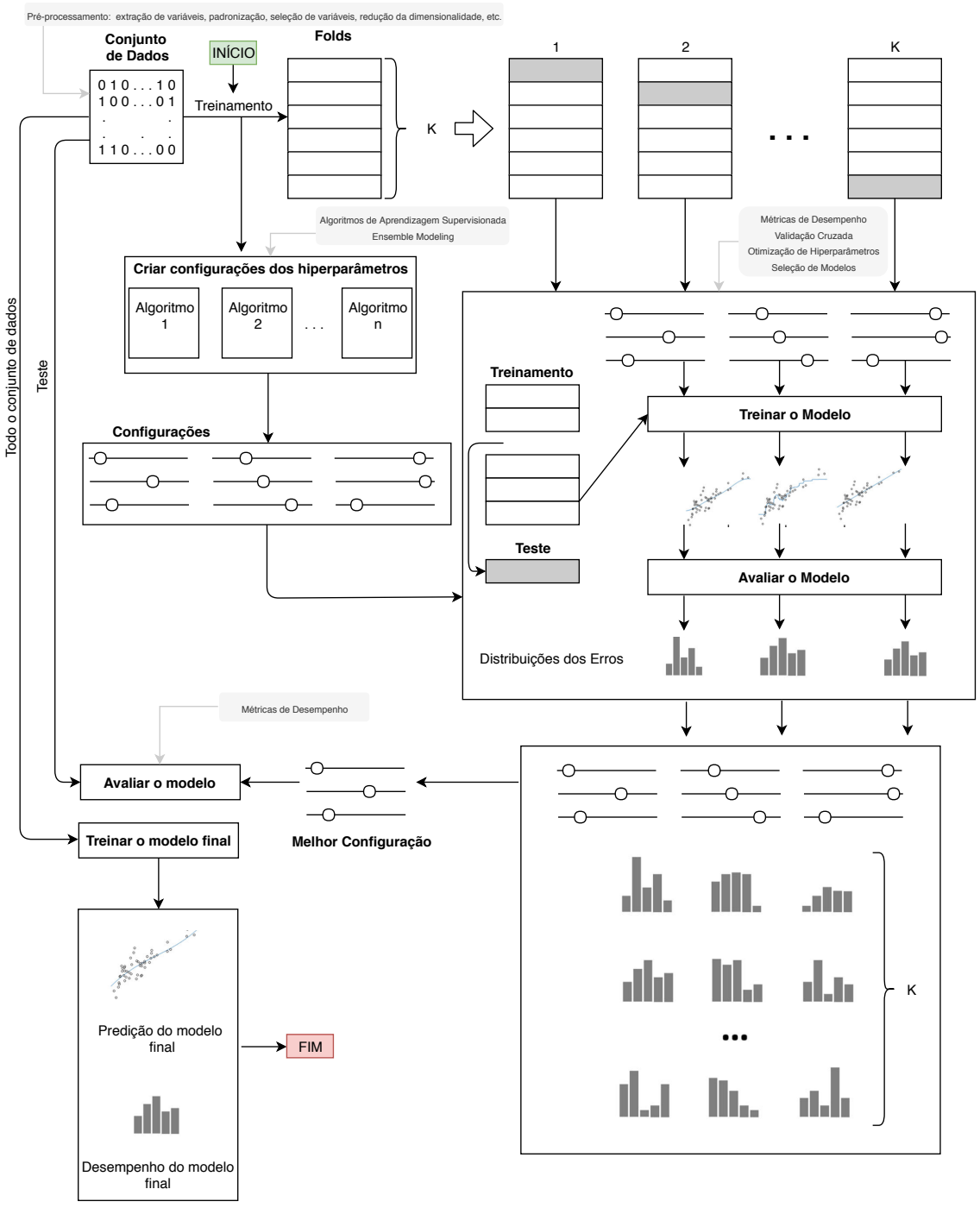

Figura 1.1. *Workflow* de aplicação das técnicas de machine learning para a análise preditiva. Adaptado de Borboudakis et al. (2017).

ou da razão entre o maior e o menor valor da variável preditora, pode ser feito o diagnóstico da necessidade de transformação dos dados;

- Remoção de preditores altamente correlacionados: normalmente os modelos preditivos apresentam desempenho inferior quando preditores altamente correlacionados estão medindo a mesma informação subjacente, além do problema do tempo desnecessariamente perdido para modelar variáveis semelhantes. Normalmente remove-se os preditores altamente correlacionados, de modo a garantir que as correlações de pares de variáveis preditoras estejam abaixo de um determinado limiar. Há também a possibilidade de utilização de técnicas para redução da dimensionalidade (como a análise de componentes principais ou o método dos mínimos quadrados parciais);

- Exclusão de preditores não informativos: variáveis preditoras que apresentam distribuição degenerada ou variância próxima a zero podem ser excluídas do conjunto de dados, resultando no aumento do desempenho do algoritmo;

- Tratamento de variáveis preditoras qualitativas: Se as variáveis qualitativas são ordinais, é importante convertê-las para números inteiros. Se tais variáveis forem nominais, é interessante transformá-las em um conjunto de variáveis indicadoras (denominadas *dummies*), as quais são representadas por valores binários. Por exemplo, para a variável qualitativa idr que representa o período do dia em que ocorreu um evento (manhã, tarde ou noite, por exemplo), três variáveis indicadoras serão criadas: $idr_{manha}, idr_{tarde}, idr_{noite}$, de modo que para cada instância apenas uma delas apresente o valor 1, enquanto as outras irá apresentar o valor 0, mapeando assim todos os valores possíveis da variável qualitativa;

- Tratamento para dados ausentes: boa parte dos conjuntos de dados apresentam variáveis que possuem valores ausentes. Como técnicas para tratamento dessas ausências, pode-se optar por remover a variável preditora, remover a observação (linha) do conjunto de dados que possui tal ausência, ou ainda fazer uso da imputação de valores interpolados, como a média, mediana ou o valor mais frequente de um preditor. Há também a possibilidade do uso de técnicas de imputação para determinar o valor a ser inserido como, por exemplo, por meio de modelos de regressão linear ou do algoritmo k-NN. Para o caso de variáveis categóricas dummificadas é também possível criar uma nova categoria específica para os valores faltantes.

Todas as tarefas de pré-processamento dos dados devem ser feitas no conjunto de treinamento. Os dados de teste não devem ser incluídos na consideração desses ajustes. Por exemplo, suponha que um valor ausente de uma variável preditora será substituído pela média dos valores dessa variável. Para o cálculo dessa média, apenas o conjunto de dados de treinamento deve ser levado em consideração. Quando tal procedimento for feito no conjunto de teste, ele será feito com base na média do conjunto de treinamento, não levando em conta as observações do conjunto de teste. Busca-se, dessa forma, evitar que o modelo tenha um conhecimento prévio sobre o conjunto de teste, o que "contaminaria" o modelo desenvolvido (KAUFMAN et al., 2012).

Normalmente após as tarefas de pré-processamento, inicia-se a fase de ajuste do modelo ao conjunto de dados de treinamento. Cada vez mais está se fazendo uso de

técnicas de reamostragem (pela limitação no número de observações disponíveis para treinamento/teste, por exemplo), assim como técnicas de comitê de modelos preditivos (*ensemble*), na qual um ou mais algoritmos de ML são utilizados para a realização da predição. Como cada algoritmo possui um conjunto de hiperparâmetros a serem ajustados, diversas configurações de hiperparâmetros são avaliadas, de modo a obter qual é a configuração que maximiza o desempenho de um modelo preditivo (GÉRON, 2017; KUHN; JOHNSON, 2013).

Dentre as técnicas de reamostragem, destaca-se a validação cruzada *k-fold*, na qual o conjunto de treinamento é dividido em k partes de tamanhos aproximadamente iguais, em que $k-1$ partes serão utilizadas para treinar o modelo preditivo, enquanto que a parte remanescente será utilizada para a validação do modelo e estimativa de seu desempenho. O processo se repete até que todas as partes tenham participado tanto do treinamento quanto da validação do modelo, resultando assim em k estimativas de desempenho que serão resumidas, normalmente, pelo cálculo da média e do erro padrão (GÉRON, 2017; KUHN; JOHNSON, 2013).

Para modelos de *ensamble*, métricas como *out-of-bag error* podem ser utilizadas para verificar o desempenho dos preditores. Uma vez que o desempenho de todos os preditores utilizados na fase de treinamento for conhecido, é possível determinar a melhor configuração dos hiperparâmetros, configurando-se assim o modelo final, o qual será testado com o conjunto de teste, possibilitando o cálculo de seu desempenho (GÉRON, 2017).

O último passo consiste no treinamento do modelo final sob todo o conjunto de dados, estabelecendo-se dessa forma um modelo preditivo a ser utilizado na predição da variável dependente a partir de novos dados (FACELI et al., 2011).

1.3.2. Avaliação do Desempenho

Há diversas métricas para a avaliação do desempenho de modelos preditivos com base no paradigma de aprendizado supervisionado. Tais métricas buscam mensurar o desempenho das predições decorrente do modelo ajustado, avaliando o quanto ele reproduz o valor observado para a resposta de interesse.

Para modelos de regressão, a métrica mais utilizada é a de erro quadrático médio (MSE, do inglês *Mean Squared Error*), de modo a quantificar o quanto o valor predito ($\hat{y}_i$) para a resposta de uma observação se aproxima de seu valor observado, y_i. O MSE é dado por:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1}$$

A raiz quadrada desse valor possibilita obter o erro quadrático médio na mesma unidade da variável de interesse. Um modelo que apresenta respostas preditas muito próximas das observadas irá apresentar um baixo MSE. A comparação entre múltiplos modelos, sob as mais variadas configurações de hiperparâmetros também podem ser feitas por meio da métrica de MSE.

Para modelos de predição por classificação, normalmente se realiza uma tabulação cruzada das classes observadas e preditas em uma *matriz de confusão*, apresentada na Figura 1.2.

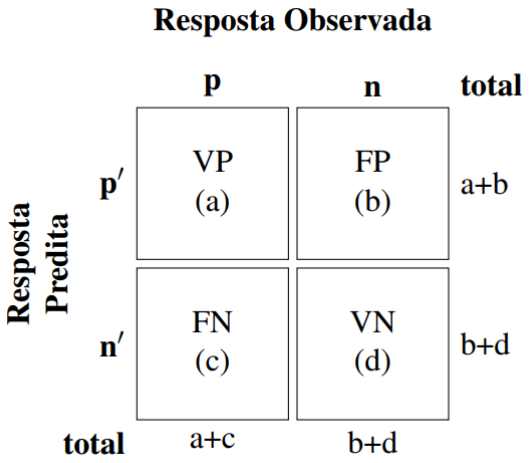

Figura 1.2. Exemplo de Matriz de Confusão para Algoritmos de Classificação.

A partir dessa matriz, diversas métricas podem ser mensuradas, apresentadas na Tabela 1.1.

Tabela 1.1. Métricas de desempenho derivadas da matriz de confusão.

Métrica (nomenclatura)	Cálculo	Descrição
Acurácia; Exatidão; Precisão	$ACC = \frac{VP+VN}{VP+VN+FP+FN}$	Proporção de observações classificadas corretamente
Taxa de falsos positivos; Fall-out	$FPR = \frac{FP}{FP+VN}$	Proporção de observações classificadas incorretamente como verdadeiras
Sensibilidade; taxa de verdadeiro positivos; recall	$TPR = \frac{VP}{VP+FN}$	Proporção de verdadeiros positivos corretamente identificados
Especificidade; taxa de verdadeiro negativos	$TNR = \frac{VN}{VN+FP}$	Proporção de verdadeiros negativos corretamente identificados
F-score (F1)	$F1 = \frac{2*TP}{2*TP+FP+FN}$	Média harmônica entre a acurácia e o recall de um classificador

Uma técnica adotada para resumir o desempenho do modelo em diferentes situações é a construção da curva *Receiver Operating Characteristic* (ROC). Essa curva avalia diferentes pontos de limiar para discriminação do que é considerado valor positivo:

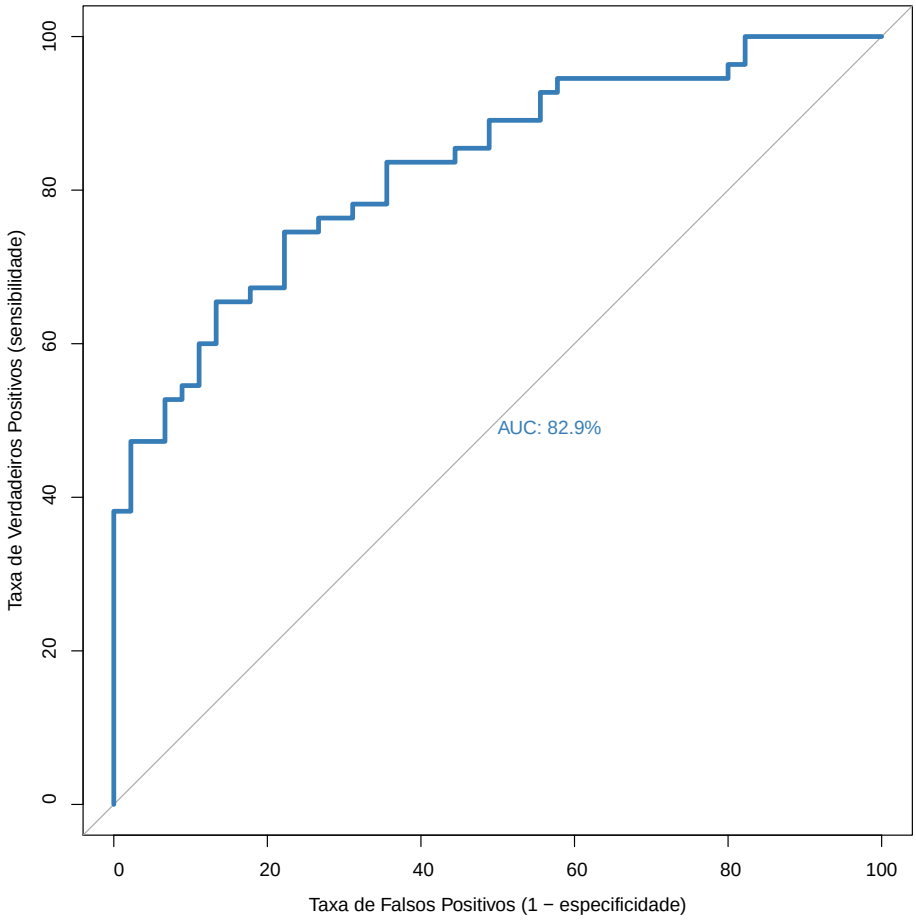

Figura 1.3. Exemplo de uma curva ROC.

enquanto o eixo vertical indica a sensibilidade (taxa de verdadeiros positivos), o eixo horizontal indica a taxa de falsos positivos (1 - sensibilidade), onde cada ponto no espaço representa os respectivos valores obtidos de uma matriz de confusão (GÉRON, 2017).

A Figura 1.3 apresenta um exemplo da curva ROC para um determinado algoritmo de classificação. A diagonal que corta o espaço ROC ao meio é conhecida como a linha de zero discriminação, onde permanecem resultados de decisões aleatórias. Um classificador satisfatório deve, no mínimo, apresentar uma curva acima dessa linha. O cálculo da área sob a curva obtida, conhecido como AUC (*area under curve*) é uma métrica pode ser utilizada para avaliar a probabilidade que um modelo possui de discriminar o exemplo positivo de um par escolhido aleatoriamente. Quanto mais próximo de 1 for o valor da AUC, melhor será o algoritmo de classificação.

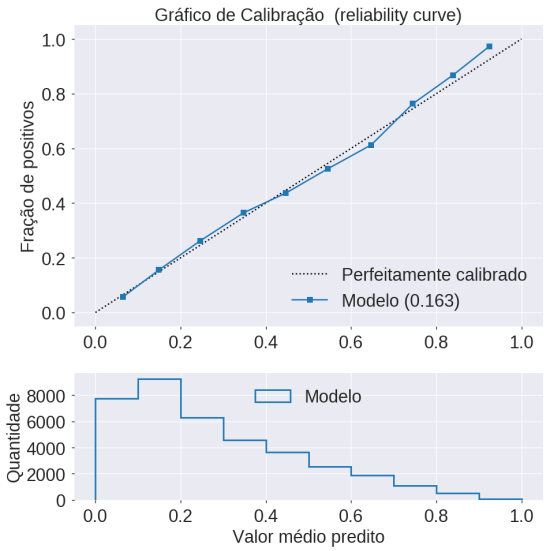

Figura 1.4. Exemplo de curva de calibração para um modelo preditivo.

Ao realizar a classificação, muitas vezes é necessário prever não apenas a classe, mas também a probabilidade associada. Dizemos que um algoritmo apresenta boa calibração quando pode ser utilizado como um estimador de probabilidade. Uma maneira de obter o valor da calibração de um modelo é pela mensuração do erro dado por $\hat{P}(+|\mathbf{x}_i)$ e $P(+|\mathbf{x}_i)$, onde $\hat{P}$ e P representam a probabilidade de ocorrência do evento predito pelo modelo e a classe verdadeira do exemplo, respectivamente. Dessa forma, o erro pode ser calculado como a diferença quadrática entre $\hat{P}(+|\mathbf{x}_i)$ e a classe verdadeira do exemplo (y_i), conforme a equação abaixo, conhecida como *Brier Score* (bs):

$$bs(y,\hat{P}) = \frac{1}{n}\sum_{1}^{n}(\hat{P}(+|\mathbf{x}_i) - y_i)^2 \tag{2}$$

Visualmente, faz-se uso do gráfico da curva de calibração do modelo. A Figura 1.4 apresenta um exemplo de curva de calibração. O diagrama agrupa as predições em compartimentos de acordo com a probabilidade obtida do modelo (eixo horizontal). A frequência com que o evento foi observado para esse subgrupo de predições é então plotada no eixo vertical. Para uma perfeita calibração, a probabilidade do evento e a frequência de ocorrência devem ser iguais, e os pontos plotados devem estar na diagonal. Assim, por exemplo, quando o modelo declara que há uma probabilidade de 25% de ocorrência de um evento, estará calibrado se tal ocorrência se concretizar em 25% das observações que tiveram tal probabilidade.

1.4. Algoritmos de ML

A seguir são apresentados os conceitos básicos dos seguintes algoritmos de aprendizado supervisionado: Regressão Linear, Regressão Logística, K-Vizinhos mais Próximos, Árvores de Decisão, Random Forest, Gradient Boosted Trees e Redes Neurais Artificiais.

1.4.1. Regressão Linear

Os modelos preditivos de regressão linear visam verificar se existe uma relação entre duas ou mais variáveis. O problema, portanto, se encontra na predição de valores contínuos para a variável de interesse (MAGALHÃES; LIMA, 2002).

Dado um conjunto com n observações e $(m+1)$ variáveis, cujos valores são representados por $(y_i, x_{i1}, x_{i2}, \ldots, x_{im})$, $i = 1, 2, \ldots, n$, o modelo de regressão linear estabelece uma relação entre a variável resposta ou de interesse $(\hat{y})$ e as variáveis explicativas ou preditoras $(x_{i1}, x_{i2}, \ldots, x_{im})$, da seguinte forma:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_m x_{mi} + e_i \tag{3}$$

onde $\beta_0, \ldots, \beta_m$ são chamados coeficientes e representam o aumento/decréscimo médio esperado para quaisquer mudanças na respectiva variável preditora, independentemente das restantes preditoras. e_i é o o erro ou parte não capturada pelo modelo, independente e identicamente distribuído.

A estimação dos coeficientes $\beta_0, \ldots, \beta_m$ é realizada de forma a minimizar a soma dos quadrados das diferenças entre cada ponto observado (y) e o seu valor estimado $(\hat{y})$ pela reta. Esta diferença é conhecida como resíduo. Em outras palavras, o que pretende-se é predizer um valor para y, denominado $\hat{y}$, minimizando o erro quadrático médio (MSE), dado por:

$$\begin{aligned} MSE(\hat{y}, y) &= \sum_{i=1}^{n} (e_i)^2 \\ &= \sum_{i=1}^{n} \left(\hat{y}(x_i) - y(x_i)\right)^2 \\ &= \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_m x_{im})]^2 \end{aligned} \tag{4}$$

onde n é o número de observações de treinamento, utilizadas na elaboração do modelo.

A Figura 1.5 apresenta um exemplo de ajuste do modelo de regressão linear simples, ou seja, com uma variável de interesse e uma variável preditora, para um determinado conjunto de dados, cuja raiz do erro médio quadrático (RMSE, do inglês *Root Mean Square Error*) é de 26,84. O RMSE é um medida utilizada para verificar a qualidade do ajuste, sendo dada na mesma escala da varável de interesse (y). Quanto menor o valor do RMSE, melhor a qualidade do ajuste.

1.4.2. Regressão Logística

O modelo de regressão logística possui sua representação gráfica em formato de "S", sendo esta conhecida como curva logística. Este modelo estabelece uma relação entre a

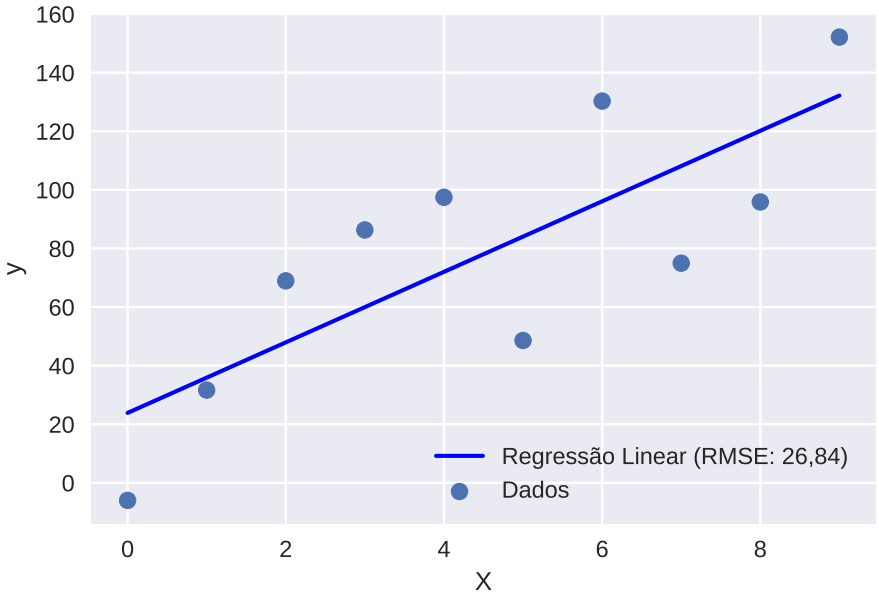

Figura 1.5. Exemplo de regressão linear simples.

probabilidade de ocorrência da variável categórica de interesse e as variáveis preditoras. A Figura 1.6 apresenta um exemplo da curva logística.

Uma das expressões mais simples da função que descreve a curva logística é dada por:

$$f(x) = \frac{1}{1 + e^{(-x)}} \tag{5}$$

onde e é conhecido como número de Euler (aproximadamente 2,72).

Em Machine Learning, a regressão logística é utilizada para tarefas de classificação, ou seja, em que a variável de interesse é categórica. Nesse caso, a variável de interesse assume valores 0 ou 1, onde 1 é geralmente utilizado para indicar a ocorrência do evento de interesse. Uma nomenclatura bastante utilizada para os valores 0 e 1 é classe negativa e classe positiva, respectivamente.

De forma simplificada, o modelo logístico pode ser definido da seguinte maneira:

$$\hat{y} = f(Z) = P(y = 1|Z) = \frac{1}{1 + e^{(-Z)}} \tag{6}$$

onde $P(y = 1|Z)$ é a probabilidade de ocorrência da classe positiva dado Z. Sendo Z:

$$Z = ln(\frac{p}{1-p}) = \alpha + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_k X_{ik} \tag{7}$$

onde p indica a probabilidade de ocorrência do evento de interesse, X é o conjunto das variáveis preditoras e α e β são parâmetros do modelo, os quais estão estimados pela

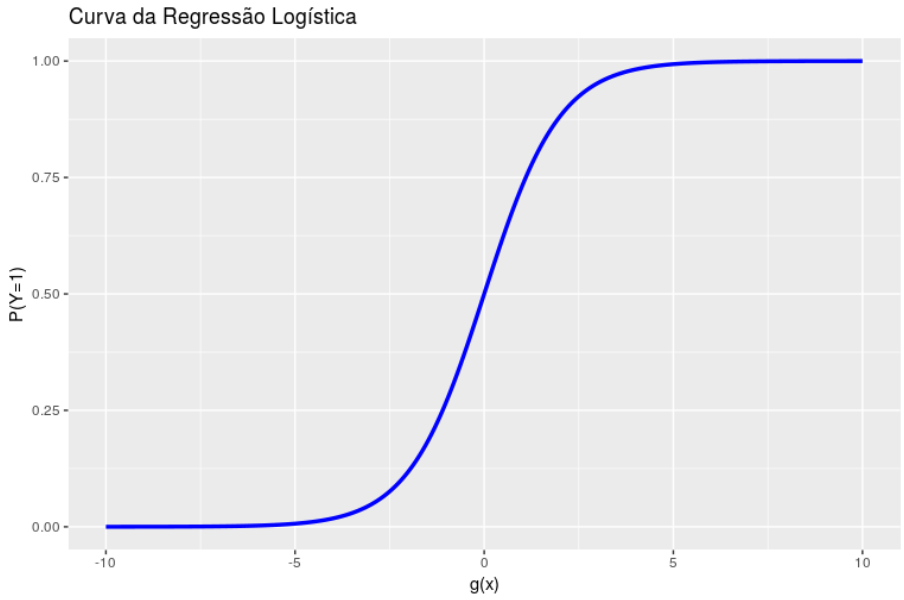

Figura 1.6. Exemplo de regressão logística.

máxima verossimilhança (FAVERO et al., 2009), encontrando uma combinação de coeficientes que maximiza a probabilidade de ocorrência do evento de interesse.

Reescrevendo o modelo, obtém-se:

$$\hat{y} = f(Z) = \frac{1}{1+e^{(-Z)}} = Prob(y = 1 \mid X_1, X_2, \ldots, X_k) = \frac{1}{1+e^{-(\alpha - \sum B_i X_i)}} \tag{8}$$

Dessa forma, um modelo de regressão logística utilizado para a classificação binária de uma variável de interesse irá predizer a probabilidade de pertencer à classe positiva. Tem-se, portanto, que $\hat{y}$ é dado por:

$$\hat{y} = \begin{cases} 0, & se\ Prob(y = 1 \mid X) < 0,5 \\ 1, & se\ Prob(y = 1 \mid X) \geq 0,5 \end{cases} \tag{9}$$

onde X é o vetor das variáveis preditoras.

1.4.3. K-vizinhos mais próximos

O algoritmo de "K-vizinhos mais próximos", conhecido por k-NN (*k-nearest neighbors*), é um algoritmo de classificação supervisionado não-paramétrico e baseado em instâncias. Este classificador é ajustado aos dados de treinamento (x, y) de modo a obter uma função $h : X \rightarrow Y$ de modo que, quando diante de uma nova observação x, prediz com uma acurácia aceitável em relação a y.

O fato de ser não-paramétrico indica que não há premissas por parte do algoritmo em relação à distribuição dos dados de treinamento. Por ser baseado em instâncias, o k-NN armazena os exemplos de treinamento para serem utilizados no momento da predição.

Quando diante de uma nova instância, o modelo irá utilizar as observações de treinamento para classificá-la.

O algoritmo k-NN assume que todas as observações correspondem a pontos em um espaço n-dimensional. Os "vizinhos mais próximos" de uma instância são mensurados por métricas de distância, tais como: distância Euclidiana, distância de Hamming, distância de Manhattan, etc. A título de exemplo, será apresentada a distância Euclidiana, definida por:

$$d(x,x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \cdots + (x_n - x'_n)^2} \tag{10}$$

Dado um número inteiro positivo ímpar k, e uma instância a ser classificada x, sob uma métrica de distância d, o classificador k-NN executa as seguintes ações:

- Para todo o conjunto de observações de treinamento, calcula-se a distância d em relação a x;
- As k observações de treinamento mais próximas de x são agrupadas em um conjunto A;
- A probabilidade condicional para cada classe é definida por:

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in A} I(y^{(i)} = j) \tag{11}$$

  onde $I(x)$ é uma função que retorna 1 se seu argumento for verdadeiro e 0, caso contrário.

- Por último, a instância x é então classificada como a classe que apresentou maior probabilidade condicional.

O hiperparâmetro necessário para ajuste do modelo é o valor de k. Quando k apresenta um valor pequeno, o modelo não consegue observar a distribuição geral das classes no conjuntos de dados, o que leva para uma situação de baixo viés, porém alta variância, tendo então o modelo apresentando sobreajuste.

Já quando k é um valor muito alto, mais elementos irão compor o mecanismo de votação (vizinho mais próximo), de modo que o modelo poderá ficar sujeito a *outliers*. Altos valores de k levam a um maior viés, porém uma menor variância quando diante de novas observações. O trabalho de Jiang et al. (2007) sugere o uso da técnica de validação cruzada para estimar o valor adequado de k.

1.4.4. Árvores de Decisão

Os modelos preditivos baseados em árvores são comumente utilizados para tarefas de classificação, embora também possam ser utilizados para tarefas de regressão. Considerado um dos mais populares algoritmos de predição, o algoritmo de árvore de decisão proporciona uma grande facilidade de interpretação.

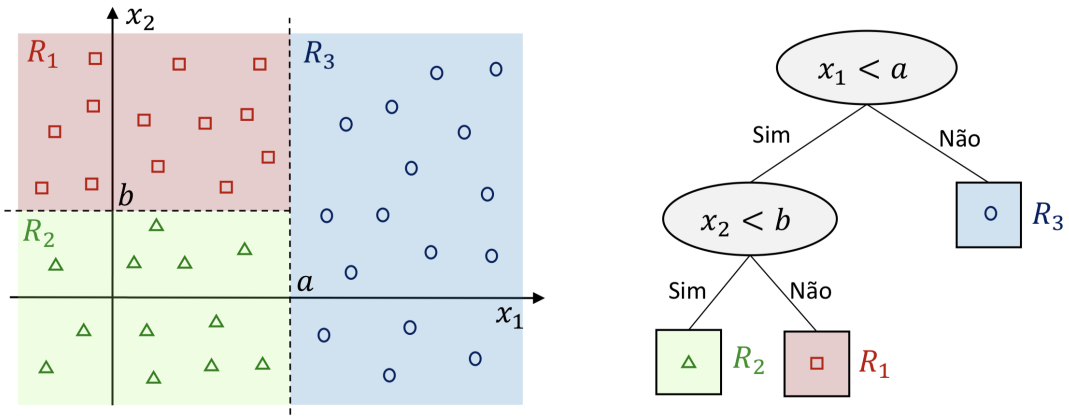

Figura 1.7. Árvore de decisão aplicada a uma tarefa de classificação. Na imagem da direita tem-se a árvore de decisão, ao passo em que a imagem da esquerda ilustra como a árvore está discriminando as classes a serem preditas (MAYRINK et al., 2016).

Uma árvore de decisão é uma estrutura hierárquica na qual cada nó interno representa uma decisão em um dos atributos do conjunto de dados, e cada ramo representa uma tomada de decisão. Nessa árvore, as folhas representam o rótulo da classe predita. A Figura 1.7 apresenta um exemplo de árvore de decisão, onde as variáveis x_1 e x_2 representam os atributos do conjunto de dados (variáveis preditoras) e os símbolos $\square$, $\triangle$ e $\circ$ indicam os rótulos de classe observados na amostra.

É possível observar que a partir do aprendizado efetuado nos atributos preditores, a árvore de decisão constrói regiões de discriminação das classes do conjunto de dados.

Uma vez construída a árvore, dado um conjunto de atributos preditores para os quais se deseja conhecer o valor da classe, tais atributos são testados por meio da árvore de decisão, onde um caminho é traçado a partir da raiz até um nó folha, o qual irá representar a classe predita.

O processo de dividir o conjunto de dados D em partições, por meio de um atributo que virá a ser um nó da árvore, busca gerar subdivisões de D em que, idealmente, cada partição tenha mais dados pertencentes a uma mesma classe. Na busca por otimizar esse resultado, o critério de seleção do atributo que irá particionar os dados será pelo o atributo que resulta em um maior número de partições que se aproximam do objetivo proposto. Dizemos então que este atributo tem uma maior capacidade de resumir D.

Há na literatura alguns algoritmos propostos para a construção da árvore, tais como ID3 (*Iterative Dichotomiser*), C4.5 e CART (*Classification and Regression Tree*). O algoritmo ID3 elenca os nós da árvore por meio da métrica do ganho de informação. Já o algoritmo CART faz normalmente uso do índice de Gini (KUHN; JOHNSON, 2013).

As árvores de decisão têm alta tendência ao sobreajuste, uma vez que para n observações de treinamento, a árvore pode construir n caminhos diferentes. Isto é, a árvore de decisão mapeou cada instância do treinamento como uma folha distinta. Para evitar situações como essa podem utilizados métodos de poda (*prunning*), o qual é controlado

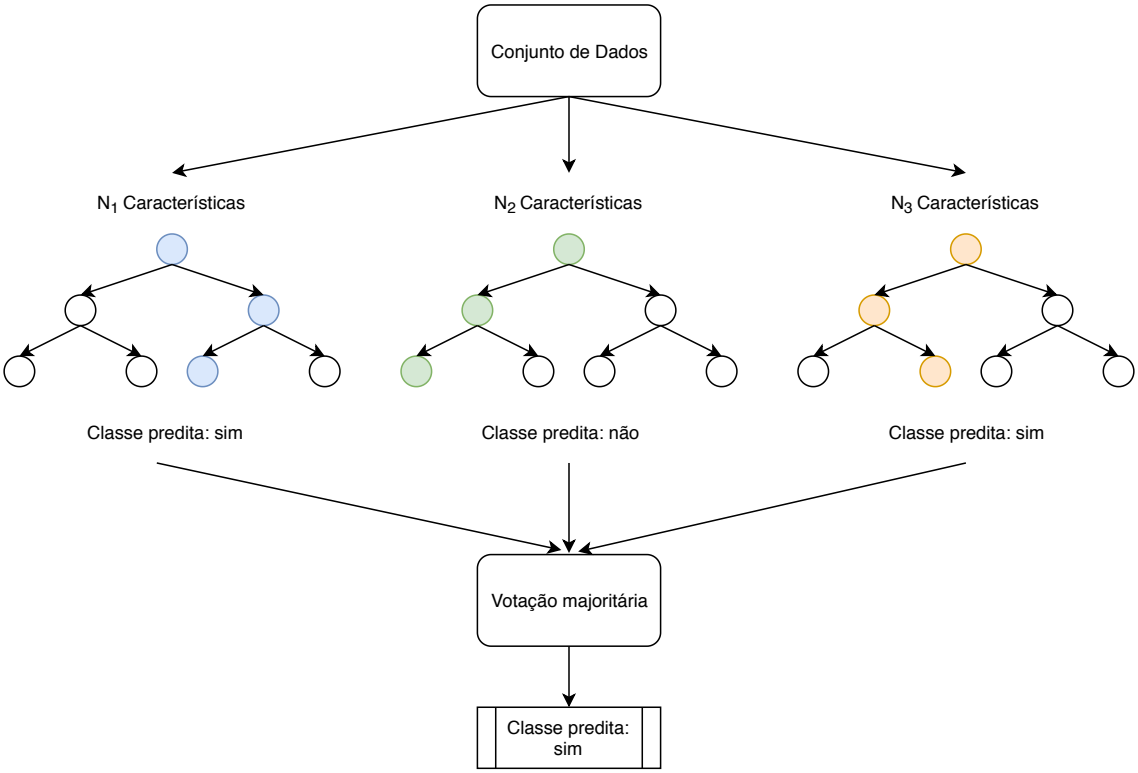

Figura 1.8. Ilustração do algoritmo Random Forest.

como um hiperparâmetro do modelo.

1.4.5. Random Forest e Gradient Boosted Trees

Os algoritmos Random Forest e Gradient Boosted Trees são exemplos de comitês (*ensembles*) de algoritmos de machine learning. Por meio desses algoritmos, diversas árvores de decisão são criadas. Trabalhos recentes demonstram que o uso destes algoritmos tem proporcionado resultados superiores na capacidade de predição quando comparado com outros algoritmos de machine learning (GÉRON, 2017; RASCHKA; MIRJALILI, 2017).

O algoritmo de Random Forest é utilizado para tarefas de regressão e classificação. Trata-se de um ensemble de árvores de decisão que são construídas com base na estratégia de bagging (*boostrap aggregation*). Neste algoritmo são geradas n árvores de decisão. Na ocasião da predição para uma determinada instância, as n árvores fazem a predição e, para os problemas de classificação, a classe majoritariamente predita é considerada a classe da instância.

Ao longo do processo de treinamento, de modo a construir as n árvores de decisão, o algoritmo seleciona, para cada *split* a ser feito em uma árvore, um conjunto p de variáveis preditoras (p é normalmente dado como $\sqrt{m}$, onde m é o número de variáveis preditoras), que será utilizado para a construção da árvore. Cada árvore é treinada com um conjunto de observações de treinamento obtidas por meio de uma seleção aleatória com repetição (técnica conhecida como Booststrap).

A Figura 1.8 ilustra a construção de três árvores de decisão para um problema de

classificação, construídas pelo algoritmo Random Forest. A classe predita da instância em questão é a classe majoritária predita pelo comitê de árvores. O método de bagging, portanto, baseia-se em criar preditores em amostras bootstrap dos dados, e depois agregá-los, ou combiná-los, para formar um preditor, o qual é esperado apresentar um resultado superior quando comparado a um algoritmo tradicional.

A estratégia de bagging proporciona que o algoritmo de Random Forest seja menos suscetível a sobreajuste, assim como viabiliza a redução do viés e da variância do modelo que está sendo construído, uma vez que as árvores de decisão são construídas fazendo-se uso de diferentes subconjuntos das variáveis preditoras.

Com o mesmo intuito de maximizar a acurácia da predição das árvores de decisão, o algoritmo Gradient Boosted Trees (FRIEDMAN, 2001) faz uso da técnica de Boosting.

De acordo com Chambers e Dinsmore (2014), Gradient Boosted Trees é um algoritmo que produz diferentes modelos sequenciais de árvores de decisão, cujos resultados são adicionados para que modelo final seja uma combinação dos anteriores, apresentando uma capacidade de predição muito superior à dos modelos individuais que foram gerados.

A cada iteração do algoritmo, uma árvore é gerada de modo a aprender e minimizar os erros da árvore anterior. A Figura 1.9 ilustra o princípio de funcionamento do algoritmo Gradient Boosted Trees como uma sequência de tacadas de golfe. A primeira tacada (F_0) representa a primeira árvore gerada. Em função da distância da posição bola ($\hat{y}$) até buraco (y), constrói-se uma nova árvore (F_1) com o objetivo de predizer Δ_1, para que $\hat{y} = F_0 + \Delta_1$. Após algumas iterações, um conjunto de árvores foi construído na busca por estimar os erros das árvores anteriores. Dessa forma, para o exemplo em questão, a predição final será dada por: $\hat{y} = F_0 + \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4$, onde F_0 é a árvore inicial, e $\Delta_1, \Delta_2, \Delta_3$ e Δ_4 são valores preditos por árvores geradas para aprender e minimizar os erros das árvores respectivamente anteriores.

1.4.6. Redes Neurais Artificiais

Modelos computacionais de redes neurais artificiais (RNA) são inspirados na estrutura neural de organismos inteligentes, os quais adquirem conhecimento por meio da experiência. As redes possuem unidades de processamento denominadas neurônios artificiais, os quais, por meio de uma rede de interligação, processam os sinais de entrada, na busca pela predição da variável de interesse, que é possível por meio do encaminhamento do sinal de entrada pela rede, simulando um processo de sinapse.

Para Haykin (1999) uma RNA é um sistema massivamente paralelo e distribuído, composto por unidades de processamento simples que possuem a capacidade de armazenar e utilizar conhecimento. As RNAs são uma alternativa para a solução de diversos problemas complexos, uma vez que a representação interna e o paralelismo inerente à sua arquitetura possibilitam um desempenho superior aos modelos convencionais para alguns tipos de problemas (BRAGA; FERREIRA; LUDERMIR, 2007).

Nas RNAs o processamento ocorre nos neurônios artificiais, que estão interconectados. O sinal de entrada é transmitido por meio das conexões entre os neurônios. Para cada conexão, de modo de representar a eficiência da sinapse, um peso associado é armazenado. Dessa forma, o processo de aprendizado de uma RNA se dá por meio

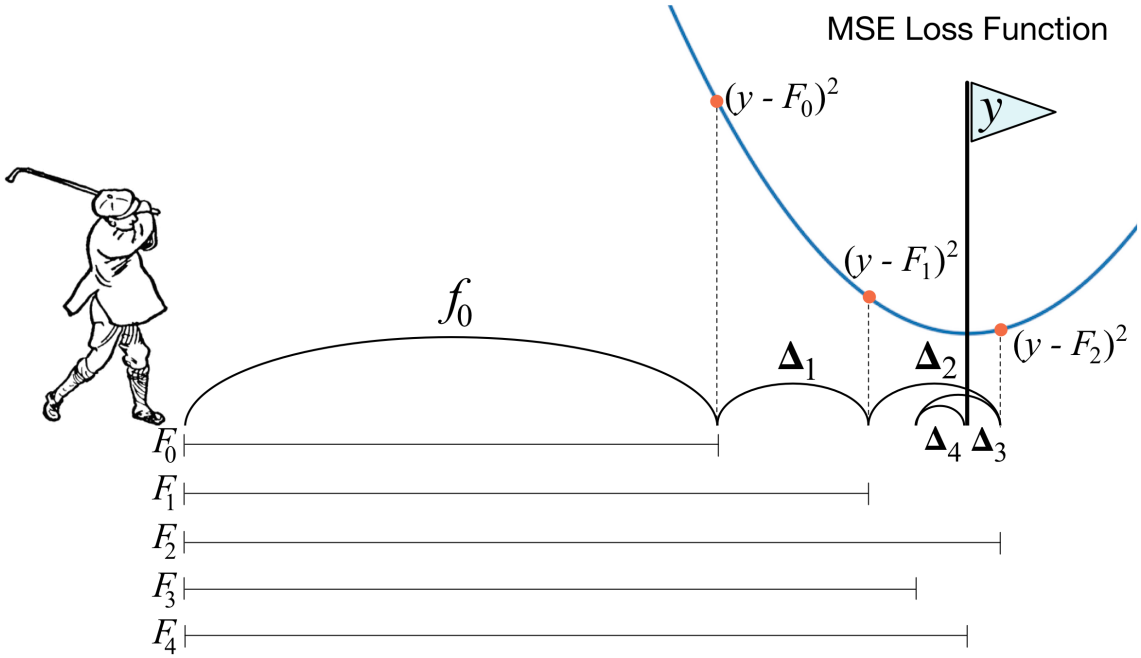

Figura 1.9. Princípio de funcionamento do algoritmo Gradient Boosted Trees (PARR; HOWARD, 2018).

do ajuste dos pesos das conexões entre os neurônios. O treinamento de uma rede neural artificial, portanto, consiste em identificar um conjunto apropriado de pesos de forma que a rede neural se comporte como o desejado, maximizando a sua capacidade de predição e generalização.

Ao longo da construção de modelos de RNAs, diversos pontos devem ser estabelecidos, como (i) a determinação do conjunto de neurônios artificiais, (ii) a escolha da arquitetura da rede neural, isto é, o padrão de conectividade entre os neurônios e as suas respectivas funções de ativação, e (iii) a escolha do método de determinação dos parâmetros da rede, conhecido como algoritmo de aprendizagem.

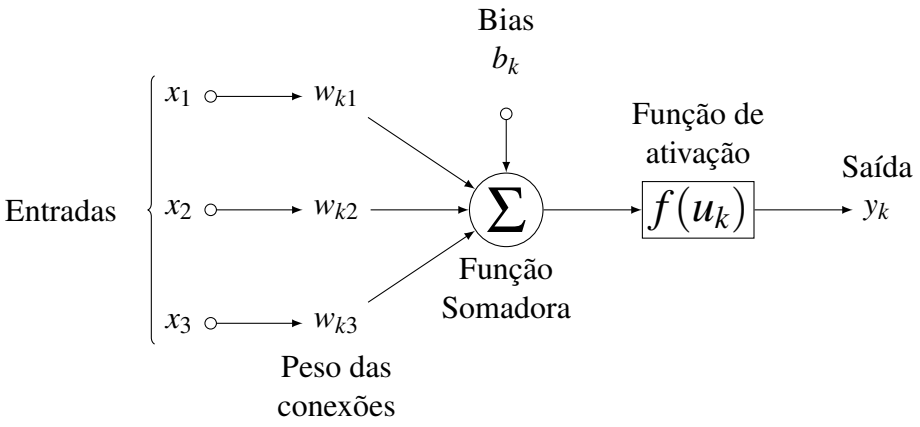

Figura 1.10. Arquitetura de um neurônio artificial.

A Figura 1.10 ilustra a arquitetura de um neurônio artificial. Nela, um neurônio artificial k recebe uma entrada de dados $x_1, x_2, \ldots, x_m$. As conexões recebem valores referentes aos pesos sinápticos, de forma que w_{ki} é o peso sináptico da conexão entre k a entrada x_i.

O neurônio realiza um somatório de todos os sinais sinápticos ponderados pelos respectivos pesos de suas conexões com k. O neurônio também conta um com valor limiar b_k que tem o papel de aumentar ou diminuir a influência do valor das entradas para a ativação do neurônio.

Uma vez calculado o produto interno do vetor de entradas x pelo vetor de pesos w_k, e considerando-se o limiar b_k, tal resultado (u_k) é submetido a uma função de ativação $f(u_k)$, a qual limita a saída do neurônio enquanto introduz não linearidade no modelo. Assim, o valor de saída do neurônio k, denominado y_k, é dado por:

$$y_k = f(u_k) = f\left(\sum_{j=1}^{m} w_{kj} x_j + b_k\right) \tag{12}$$

A notação pode ser simplificada, considerando-se o bias simplesmente como um sinal de entrada de valor $x_0 = 1$ e com peso associado $w_{k0} = b_k$:

$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right) = f(w^T x) \tag{13}$$

A escolha adequada da função de ativação $f(u_k)$ e de suas propriedades contribuem para melhorar o processo de convergência da rede durante seu treinamento. Existem diversas funções matemáticas que são utilizadas como função de ativação. As funções de ativação mais comumente usadas são: função sigmóide logística e a função tangente hiperbólica. Com o advento das redes neurais profundas, a função de ativação retificadora linear (ReLU) também tem sido bastante utilizada. Estas funções estão ilustradas na Figura 1.11.

A RNA passa a ser conhecida como um algoritmo de deep learning (aprendizado profundo) quando sua arquitetura é formada por múltiplas camadas. As camadas internas de uma rede neural são denominadas camadas ocultas. A Figura 1.12 ilustra um exemplo de uma rede multicamada *feedforward*, na qual cada neurônio artificial recebe apenas entradas de unidades situadas na camada imediatamente anterior. Devido ao conjunto de conexões sinápticas e da riqueza de interações neurais, as camadas ocultas são capazes de extrair características complexas dos dados (HAYKIN, 1999).

1.5. Implementando Modelos de Machine Learning em Python

É crescente o aumento da popularidade da linguagem Python em aplicações científicas. Boa parte deste aumento se dá pela diversidade de bibliotecas científicas como Numpy e Scipy. Na área de Machine Learning, a linguagem Python apresenta diversas bibliotecas que permitem a execução de modelos de ML, com destaque para a biblioteca Scikit-Learn (BUITINCK et al., 2013).

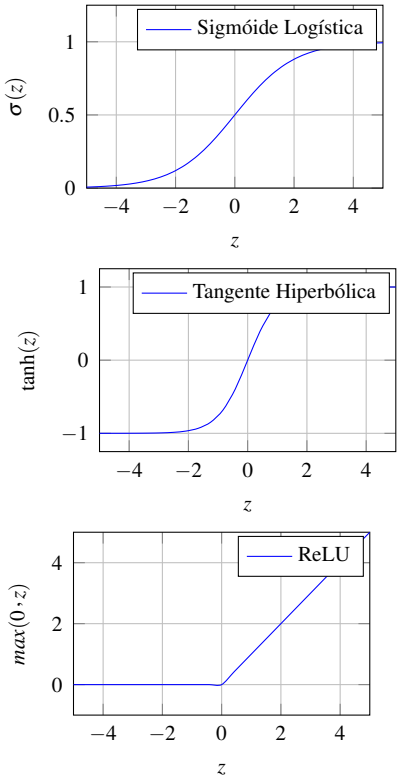

Figura 1.11. Exemplos de Funções de ativação para Redes Neurais Artificiais.

A proposta da biblioteca Scikit-Learn consiste em fornecer um conjunto de funcionalidades padronizadas, de modo a permitir que especialistas das mais diversas áreas possam construir modelos de Machine Learning (BUITINCK et al., 2013).

Além de bem documentados, os modelos de ML implementados na biblioteca Scikit-Learn são padronizados quando ao input de dados e aos métodos disponíveis para a sua execução. Todos os modelos disponíveis na biblioteca aceitam entrada de dados na forma de arrays bidimensionais (observações x características).

Há três componentes básicos: Transformer, Estimator e Predictor. Estes são apresentados a seguir, em conjunto com a Figura 1.13, a qual apresenta como estes componentes estão relacionados entre si e como são aplicados aos conjuntos de treinamento e de teste na busca pelo ajuste de modelos de Machine Learning. Pressupõe-se que o conjunto de dados foi dividido em duas partes: treinamento e teste, para o qual a biblioteca Scikit-Learn dispõe do método `train_test_split`, que pode ser utilizado como segue:

```
1 from sklearn.model_selection import train_test_split
2
3 #70% treino, 30% teste
4 X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪  train_size=0.70, random_state=42)
```

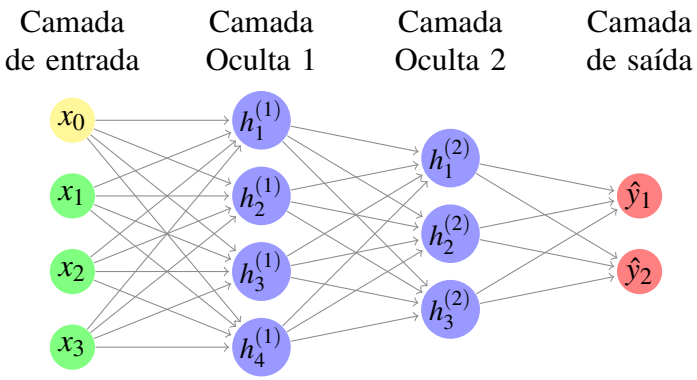

Figura 1.12. Exemplo de rede neural multicamadas.

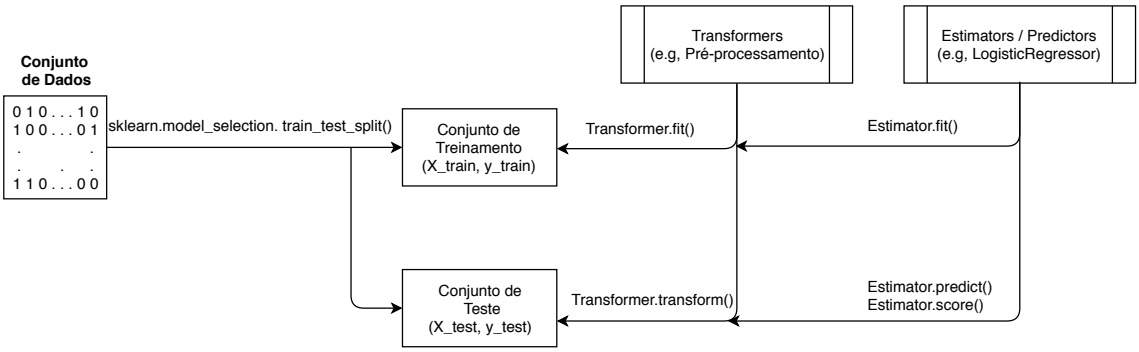

Figura 1.13. Componentes básicos da biblioteca Scikit-Learn e seu uso na construção de modelos de Machine Learning.

A linha ❹ executa a divisão do conjunto de dados em duas partes (treino, teste), de modo que o teste tenha 30% dos dados. Para garantir a reprodutibilidade dos resultados, a semente de aleatoriamente (random_state) precisa ser definida com algum valor (no caso o número que nos dá o significado da vida do universo e tudo mais, 42).

- **Transformer**: De modo a contemplar as necessidades de modificação e/ou filtro de dados antes de introduzi-los em um algoritmo de ML, a biblioteca Scikit-Learn disponibiliza uma interface denominada Transformer. Atividades de pré-processamento, seleção de características, extração de características e algoritmos de redução da dimensão são exemplos de transformadores disponibilizados pela biblioteca. Por exemplo, para a necessidade de padronizar os dados (média = 0 e desvio-padrão = 1), pode-se fazer uso do Transformer `StandardScaler`, por meio do método `fit`:

```
1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler()
4 scaler.fit(X_train)
5 X_train = scaler.transform(X_train)
```

Dessa forma, o objeto `scaler` armazena a média e o desvio-padrão do conjunto de treinamento, obtidos com a invocação do método `fit()` (❹), para que quando da invocação do método `transform` (❺) estes valores sejam utilizados. Esta característica é muito importante para a padronização do conjunto de teste, o qual deve ser padronizado fazendo-se uso da média e desvio-padrão que foram aprendidos a partir do conjunto de treinamento. A transformação do conjunto de teste se daria pelo seguinte código:

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

- **Estimator**: trata-se de uma interface para a implementação de algoritmos de ML. Todo algoritmo de ML disponibilizado na biblioteca Scikit-Learn implementa esta interface. Não apenas os algoritmos de ML são implementações da interface Estimator, mas também outras funcionalidades (como seleção de características, imputação de dados, etc.).

  A interface Estimator disponibiliza o método `fit` para que um modelo seja ajustado a partir de um conjunto de dados. Na inicialização de um algoritmo disponibilizado pela biblioteca é possível definir o conjunto de hiperparâmetros que será utilizado no momento da execução do método `fit`.

  A biblioteca também conta com um conjunto de valores padronizados para cada possível hiperparâmetro. Ao longo da execução do método `fit`, o algoritmo, fazendo-se uso dos dados de treinamento, busca determinar os parâmetros específicos do modelo que está sendo ajustado. Por exemplo, ao longo de uma regressão linear, o algoritmo irá determinar os valores dos coeficientes da regressão a partir dos dados fornecidos ao método `fit`. Por convenção, todo parâmetro aprendido pelo modelo recebe um subtraço como sufixo. No caso da regressão, os coeficientes determinados pelo modelo são armazenados na variável `coef_`.

  A seguir é apresentado um trecho de código-fonte para a execução de uma regressão logística. Nesse caso, o estimator é `LogisticRegression`, cujo hiperparâmetro `penalty` recebe o valor `l1`, reproduzindo assim uma regularização l_1. Outros hiperparâmetros não especificados (como o parâmetro de regularização C) irão receber o valor padrão. Uma vez que o método `fit` é invocado, o modelo será construído a partir dos dados de treinamento fornecidos.

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty="l1")
clf.fit(X_train, y_train)
```

Para conhecer todos os hiperparâmetros disponíveis para os ajustes de um modelo preditivo, pode-se fazer uso do método `get_params()`, conforme a seguir:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty="l1")
#obtendo hiperparametros disponiveis:
clf.get_params()
>>{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'max_iter': 100,
 'multi_class': 'warn',
 'n_jobs': None,
 'penalty': 'l1',
 'random_state': None,
 'solver': 'warn',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

- **Predictor**: A interface Predictor estende a interface Estimator por meio da adição de um método `predict`. A partir dos exemplos anteriores, é possível invocar o método `predict` para a regressão logística, obtendo assim as classes preditas pelo modelo. Normalmente faz-se uso do conjunto de teste (neste caso, `X_test`) como os dados para os quais queremos fazer a predição.

```
y_pred = clf.predict(X_test}
```

Alguns algoritmos implementados na biblioteca Scikit-Learn disponibilizam o método `predict_proba`, o qual retorna a probabilidade de pertinência de uma instância para cada classe candidata. A interface Predictor também disponibiliza um método denominado `score` voltado para a avaliação do desempenho do modelo. Em modelos de aprendizado supervisionado, este método normalmente recebe como parâmetros o conjunto de dados teste e os rótulos de teste (`X_test` e `y_test`, respectivamente).

1.5.1. Conhecendo o conjunto de dados

As doenças cardiovasculares são uma das principais causas de morte nos Estados Unidos, sendo contabilizadas em mais de 30% das ocorrências anuais. Quando detectadas precocemente é possível reduzir os infartos em 80%, assim como a mortalidade em 45% (LADERAS et al., 2018).

Na busca por fomentar o acesso a base de dados clínicos que possam auxiliar no desenvolvimento de modelos preditivos para essas doenças, Laderas et al. (2018) cria-

ram um conjunto de dados sintéticos de modo a possibilitar o desenvolvimento de modelos preditivos para doenças cardiovasculares. Para a elaboração do conjunto de dados, buscou-se incluir dependências realísticas entre as variáveis (por exemplo, pacientes com alto índice de massa corpórea (IMC) têm maior prevalência de diabetes tipo 2), bem como buscou-se apresentar baixa prevalência da doença para determinadas coortes (pessoas com menos de 40 anos, por exemplo). O conjunto de dados sintéticos possui 425.195 observações e as variáveis disponíveis estão apresentadas na Tabela 1.2.

Todos os códigos exibidos neste capítulo foram executados na plataforma Google Colaboratory e estão disponíveis no endereço eletrônico: <https://tinyurl.com/SBCAS2019-ML>.

Para fins de simplificação, alguns comandos e importações de bibliotecas serão omitidos, mas o conteúdo completo dos comandos encontra-se disponível no arquivo disponibilizado.

Os comandos que seguem são executados sob a variável `dataset`, que representa o arquivo de dados lido a partir de seu repositório, conforme segue:

```
1 dataset = pd.read_csv('https://git.io/fjm0H')
```

Tabela 1.2. Variáveis disponíveis no conjunto de dados cvdRiskData.

Variável	Definição	Tipo	Possíveis valores
patientID	Identificador único do paciente	Alfanumérica	Diversos
age	Faixa etária do paciente	Categórica	0-20, 20-40, 40-55, 55-70, 70-90
htn	Identifica se o paciente possui hipertensão	Categórica	Y, N
treat	Identifica se o paciente recebe tratamento para a hipertensão	Categórica	Y, N
smoking	Identifica se o paciente é fumante	Categórica	Y, N
race	Identifica a raça informada pelo paciente	Categórica	AmInd (america indian), Asian/PI (asian/pacif islander), Black/AfAm (black/african american), White
gender	Identifica o sexo do paciente	Categórica	Male, Female NA significa que o paciente não quis registrar essa informação
t2d	Identifica se o paciente possui Diabetes tipo 2	Categórica	Y, N
numAge	Idade do paciente, em anos	Contínua	0-90
bmi	Índice de massa corpórea (IMC)	Contínua	15-36
tchol	Colesterol total	Contínua	155-245
sbp	Pressão arterial sistólica	Contínua	74-226
cvd	Identifica se o paciente possui doença cardiovascular	Categórica	Y, N

1.5.2. Preparação dos dados

Para a predição de risco cardiovascular, optou-se por filtrar do conjunto de dados os registros de indivíduos que possuam idade superior a 55 anos. Além disso, as variáveis preditoras `patientID`, `age` e `treat` serão removidas uma vez que `patientID` é o

ID do paciente, que não tem relacionamento algum com a variável de interesse, `age` já se encontra representada pela variável `numAge` e `treat` só foi respondida por pacientes com hipertensão.

O código abaixo realiza o filtro da idade e remove as variáveis elencadas. Tal funcionalidade é provida na biblioteca `dfply`, a qual tenta reproduzir na linguagem Python as funcionalidades da biblioteca R `dplyr`. Na linha ❷ há a aplicação do filtro de idades acima de 55 anos, enquanto que na linha ❸ ocorre a remoção das variáveis elencadas. Na biblioteca `dfply`, a variável `X` é uma referência ao conjunto de dados que antecede o símbolo », neste caso `dataset`. Assim, `X.numAge` representa a coluna `numAge` do conjunto de dados `dataset`.

```
cvd_data = (dataset >>
                mask(X.numAge > 55) >>
                drop(X.patientID, X.age, X.treat)
               )
```

Uma vez realizado o fitro e exclusão das variáveis, passa-se para o tratamento das variáveis categóricas, as quais devem ser decompostas em um conjunto de variáveis indicadoras (*dummies*), conforme segue:

```
cvd_data = pd.get_dummies(cvd_data, columns=['htn', 'smoking', 't2d',
  ↪  'gender', 'race'])
```

A linha ❶ faz uso da função `get_dummies`, da biblioteca Pandas, que irá converter uma variável categórica em n colunas, onde n é o número de valores distintos da variável.

As variáveis contínuas (`numAge`, `bmi`, `tchol`, `sbp`) serão padronizadas. Para isso, o código a seguir é utilizado.

```
from sklearn.compose import ColumnTransformer

### variáveis contínuas que serão padronizadas
continuous_cols = ['numAge', 'bmi', 'tchol', 'sbp']
def setScaler():
  ct = ColumnTransformer([
        ('scaler', StandardScaler(), continuous_cols)
    ], remainder='passthrough'
  )
  return ct

scaler = setScaler()
```

Como será feita a padronização apenas das variáveis contínuas, será utilizada a funcionalidade `ColumnTransformer` do Scikit-Learn, que permite que um transformador (no caso, `StandardScaler()` ❼) seja aplicado apenas nas colunas definidas em `continuous_cols`.

A padronização das variáveis deve ser feita apenas com base no conjunto de treinamento, sendo posteriormente aplicada ao conjunto de teste. Assim, é necessário dividir o conjunto de dados em conjunto de treinamento e conjunto de teste, como segue:

```
1 variaveis_preditoras = cvd_data.iloc[:, cvd_data.columns != 'cvd']
2 classe = cvd_data.iloc[:, cvd_data.columns == 'cvd']
3 X_train, X_test, y_train, y_test =
    ↪ train_test_split(variaveis_preditoras, classe, train_size = 0.70,
    ↪ random_state = 42)
```

A linha ❶ seleciona as variáveis preditoras (todas aquelas diferentes de `cvd`). A linha ❷ seleciona a variável de interesse (`cvd`) e a armazena em `classe`. A linha `3`, por sua vez, faz a divisão do conjunto de dados em 70% para treinamento (`train_size`) e 30% para teste.

Uma vez particionados, pode-se aplicar a padronização das variáveis contínuas.

```
1 scaler.fit(X_train)
2 X_train = scaler.transform(X_train)
3 X_test = scaler.transform(X_test)
```

Baseando-se no fluxo de aplicação de transformadores exibido na Figura 1.13, na linha ❶ o transformador é ajustado para os dados de treinamento, enquanto que na linha ❷ o conjunto de treinamento (`X_train`) tem suas variáveis contínuas padronizadas, assim como o conjunto de teste na linha seguinte.

Ao término do pré-processamento o conjunto de dados de treinamento se apresenta conforme exibido na Figura 1.14. Variáveis categóricas foram transformadas em *dummies*. A variável `smoking`, por exemplo, apresentava dois possíveis valores (`Y`, `N`), tendo sido, portanto, transformada em duas colunas: `smoking_Y`, e `smoking_N`. As variáveis contínuas, como `bmi`, foram padronizadas.

1.5.3. Implementação dos modelos

A seguir apresenta-se a execução dos algoritmos de Regressão Logística, K-vizinhos mais próximos, Árvore de Decisão, Random Forest, Gradient Boosted Trees e Redes Neurais Artificiais.

Tais algoritmos irão ajustar modelos de classificação para a variável de interesse `cvd`, predizendo assim a ocorrência de doença cardiovascular a partir do conjunto de variáveis preditoras. Após ajuste dos modelos ao conjunto de treinamento, serão obtidas as seguintes métricas sob o conjunto de teste: (i) precisão, (ii) sensibilidade (recall), (iii)

numAge	bmi	tchol	sbp	htn_N	htn_Y	smoking_N	smoking_Y	t2d_N	t2d_Y	gender_F	gender_M	race_AmInd	race_Asian/PI	race_Black/AfAm	race_White
1.273356	-1.347215	-0.859665	1.195618	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
-0.659311	0.247619	-0.453873	0.531070	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
-1.196162	-0.435881	0.695870	0.531070	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
0.736504	-0.663715	-0.487689	-0.734736	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
-0.122459	-0.208048	1.338374	-1.367639	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
-1.088792	2.981619	1.879430	1.512070	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
-0.015088	-1.119381	-0.994929	1.353844	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
0.521763	-0.208048	0.087183	1.227263	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
0.092282	0.019786	0.154814	-0.798026	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
-0.551940	-1.119381	0.154814	0.784231	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0

Figura 1.14. Amostra do conjunto de dados de treinamento após a etapa de pré-processamento.

F-score e (iv) AUC. Além disso, os modelos terão suas curvas de calibração analisadas e mensuradas pela métrica *brier score*.

Para facilitar a execução dos modelos de classificação, foi construída a função *runModel()* que irá executar os modelos preditivos, definida como segue:

```
def runModel(model, X_train, y_train, X_test, y_test,
  ↪  confusion_matrix=True, normalizeCM=False, roc=True,
  ↪  plot_calibration=True, random_state=42, title="", pos_label=1):

    clf = model
    name = title
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    if hasattr(clf, "predict_proba"):
        prob_pos = clf.predict_proba(X_test)
    else:
        prob_pos = clf.decision_function(X_test)
        prob_pos = \
            (prob_pos - prob_pos.min()) / (prob_pos.max() -
              ↪  prob_pos.min())

    prob_pos = prob_pos[:,1]
    clf_score = brier_score_loss(y_test, prob_pos,
      ↪  pos_label=pos_label)
    print("%s:" % name)
    print("\tBrier: %1.3f" % (clf_score))
    print("\tPrecision: %1.3f" % precision_score(y_test, y_pred))
    print("\tRecall: %1.3f" % recall_score(y_test, y_pred))
    print("\tF1: %1.3f\n" % f1_score(y_test, y_pred))

    if confusion_matrix:
        skplt.metrics.plot_confusion_matrix(y_test, y_pred,
          ↪  normalize=normalizeCM, title=name)
    if roc:
```

```
25          skplt.metrics.plot_roc(y_test, prob_pos, plot_micro=False,
              ↪ plot_macro=False, classes_to_plot=[1],
              ↪ title=name,figsize=(10,10))
26
27      if plot_calibration:
28          fraction_of_positives, mean_predicted_value = \
29                  calibration_curve(y_test, prob_pos, n_bins=10)
30          fig = plt.figure(3, figsize=(10, 10))
31          ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
32          ax2 = plt.subplot2grid((3, 1), (2, 0))
33          ax1.plot([0, 1], [0, 1], "k:", label="Perfeitamente
              ↪ calibrado")
34          ax1.plot(mean_predicted_value, fraction_of_positives, "s-",
35                       label="%s (%1.3f)" % (name, clf_score))
36
37          ax2.hist(prob_pos, range=(0, 1), bins=10, label=name,
38                       histtype="step", lw=2)
39          ax1.set_ylabel("Fração de positivos")
40          ax1.set_ylim([-0.05, 1.05])
41          ax1.legend(loc="lower right")
42          ax1.set_title('Gráfico de Calibração  (reliability curve)')
43          ax2.set_xlabel("Valor médio predito")
44          ax2.set_ylabel("Quantidade")
45          ax2.legend(loc="upper center", ncol=2)
46          plt.tight_layout()
47          plt.show()
```

A função *runModel* recebe como parâmetro um modelo a ser ajustado, os conjuntos de treinamento e de teste, além de parâmetros sobre quais saídas serão produzidas. Na linha 5 o modelo é ajustado ao conjunto de treinamento, produzido a predição ($\hat{y}$ = y_pred) na linha 6. Como alguns modelos implementados na biblioteca não possuem o método `predict_proba` (a qual transforma a saída da predição na probabilidade da ocorrência da classe de interesse), as linhas 7 - 12 validam a sua existência, caso contrário, fazem uso do método `decision_function`.

As linhas 16 - 20 exibem os valores do *brier score*, precisão, sensibilidade (recall) e o F-score (F1). A partir da linha 22 os parâmetros da função são validados, de modo a gerar: (i) matriz de confusão 22, curva ROC 24 e curva de calibração 27.

1.5.4. Avaliação do Desempenho

A seguir são apresentados os resultados obtidos com o ajuste dos modelos preditivos escolhidos. Os modelos tiveram seus hiperparâmetros otimizados fazendo-se uso da técnica de Grid Search disponibilizada pela biblioteca Scikit-Learn, que busca exaustivamente todas as combinações no espaço de hiperparâmetros e seus possíveis valores.

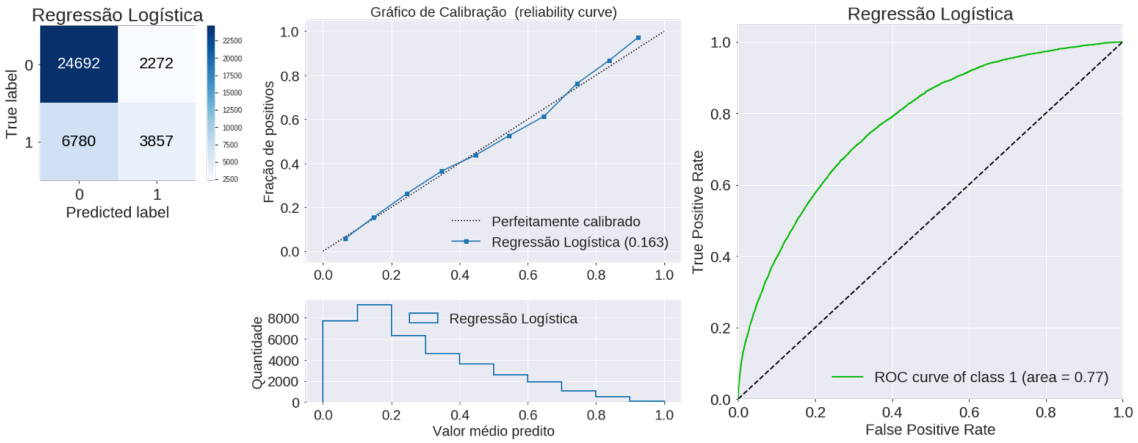

Figura 1.15. Desempenho do modelo de Regressão Logística.

1.5.4.1. Regressão Logística

O código a seguir é responsável pela execução do modelo de regressão logística.

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
runModel(lr, X_train, y_train, X_test, y_test, title="Regressão
  ↪ Logística")
```

O desempenho do algoritmo é apresentado na Figura 1.15. A precisão do algoritmo ficou em 0,629, enquanto a sua sensibilidade foi de 0,363. O valor da AUC do modelo foi de 0,770. Em relação a calibração do modelo, valor do *brier score* foi de 0,163.

1.5.5. KNN

Para o ajuste do modelo de k-vizinhos mais próximos, o seguinte grid foi elaborado:

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()
grid_params = {
        'n_neighbors': np.arange(3,301, 2),
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan']
}
gs = GridSearchCV(neigh, param_grid=grid_params,
  ↪ scoring='roc_auc',cv=3, n_jobs=-1)
```

Além do valor de *k* (hiperparâmetro n_neighbors), buscou-se verificar os seguintes hiperparâmetros:

- weights: define como se dará o peso das observações de treinamento, no momento da determinação dos vizinhos mais próximos de x. Este peso pode ser uniformemente distribuído, ou inversamente proporcional à distância (quanto menor a distância entre a instância e x, maior será o peso);
- metric: define a métrica de distância a ser utilizada. Neste caso, optou-se por testar a distância euclidiana e a distância de manhattan.

Após a execução do grid, obteu-se que a melhor configuração para o modelo é $k = 59$, weights = uniform, e metric = manhattan. O desempenho do modelo treinado, quando aplicado ao conjunto de teste, é apresentado na Figura 1.16. Em relação ao modelo de regressão logística, k-NN apresentou a mesma AUC (0,770), porém com uma maior precisão (0,633). k-NN também apresentou maior sensibilidade do que o modelo de regressão logística, embora com uma calibração inferior (maior *brier score*). Embora o desempenho seja superior, cabe ressaltar o custo computacional do k-NN em relação ao modelo logístico. Por fazer uso de todas as observações de treinamento para a classificação de uma nova instância, o uso de memória, e consequentemente, o tempo de processamento do k-NN é superior ao modelo de regressão logística.

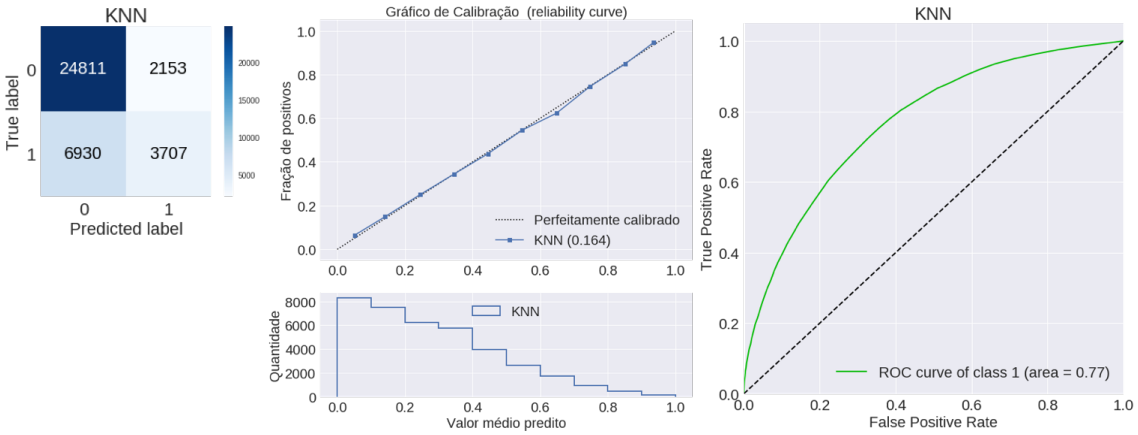

Figura 1.16. Desempenho do modelo de K-vizinhos mais próximos.

1.5.5.1. Árvore de Decisão

O algoritmo de árvore de decisão teve seus hiperparâmetros ajustados pela técnica de Grid Search, exibida no código abaixo.

```
param_grid = {"criterion": ["gini", "entropy"],
              "min_samples_split": [2,5,8,15,20],
              "max_depth": [2,4,6,8,10],
              "min_samples_leaf": [1,2,4,8,10],
              "max_leaf_nodes": [2,4, 7,9, 12, 20],
              }
dt = DecisionTreeClassifier(random_state=42)
```

```
cv_dt = GridSearchCV(dt, cv = 3,
                     param_grid=param_grid,
                     n_jobs = -1)
```

Os seguintes hiperparâmetros foram ajustados:

- criterion: função utilizada para avaliar o atributo mais adequado para subdividir a árvore de decisão. No caso, serão testadas duas opções: entropia e índice de gini;
- min_samples_split: número mínimo de observações que um nó deve possuir na árvore para que seja possível uma nova divisão. Serão testados os valores 2, 5, 8, 15 e 20;
- max_depth: limita a profundidade da árvore. Serão testados os valores 2, 4, 6, 8 e 10;
- min_samples_leaf: número mínimo de observações que deve existir em uma folha da árvore. Serão testados os valores 1, 2, 4, 8 e 10;
- max_leaf_nodes: número máximo de folhas que a árvore pode possuir. Serão testados os valores 2, 4, 7, 9, 12 e 20.

Após a execução do grid, onde todas as combinações possíveis dos hiperparâmetros selecionados foram testadas, o modelo que apresentou o melhor resultado foi:

```
Model with rank: 1
Mean validation score: 0.755 (std: 0.001)
Parameters: {'criterion': 'entropy', 'max_depth': 6, 'max_leaf_nodes':
  ↪  20, 'min_samples_leaf': 10, 'min_samples_split': 20}
```

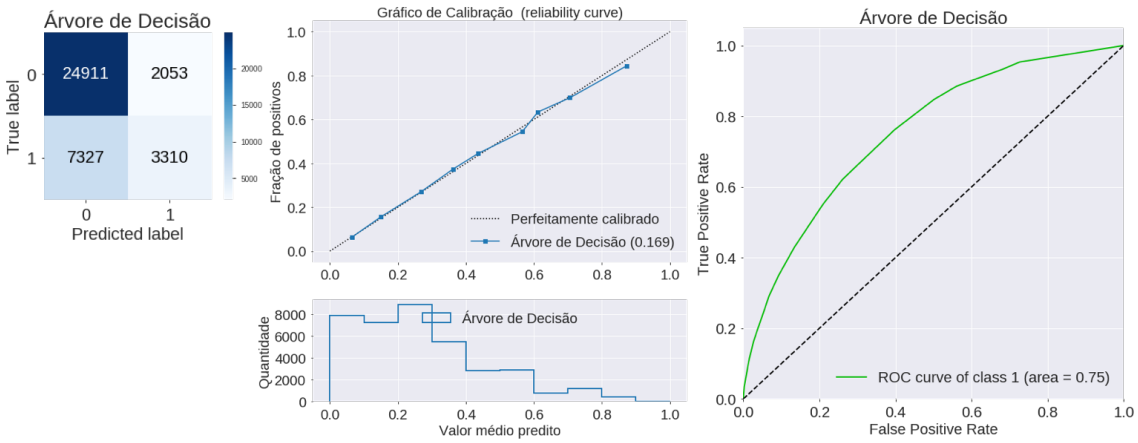

Figura 1.17. Desempenho do modelo de Árvore de Decisão.

Os resultados obtidos estão apresentados na Figura 1.17. O modelo obteve o menor desempenho em relação aos dois modelos anteriores (regressão logística e k-vizinhos mais próximos): AUC de 0,750, precisão de 0,617, sensibilidade de 0,311 e *brier score* de 0,169.

A biblioteca `dtreeviz` permite a visualização da árvore de decisão criada. A Figura 1.18 apresenta o caminho percorrido na árvore de decisão para predizer uma instância aleatoriamente obtida do conjunto de teste. É possível observar que a predição negativa para risco cardiovascular foi realizada a partir dos atributos `sbp`, `gender`, `numAge` e `race`.

1.5.5.2. Random Forest

Para o ajuste do modelo de Random Forest, os seguintes hiperparâmetros foram otimizados:

```
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000,
  ↪  num = 5)]
max_features = ['log2', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 20, num = 5)]
min_samples_split = [2, 5, 10]
min_samples_leaf = [2, 4]
bootstrap = [True, False]

param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

cv_rf = GridSearchCV(fit_rf, cv=3, param_grid=param_grid,
                     n_jobs = -1)
```

- n_estimator: número de árvores a serem construídas no comitê;
- max_features: número de características a serem utilizadas a cada split da árvore;
- max_depth: profundidade da árvore;
- min_samples_split: número mínimo de observações que um nó deve possuir na árvore. para que uma nova divisão seja possível;
- min_samples_leaf: número mínimo de observações que deve existir em uma folha da árvore;

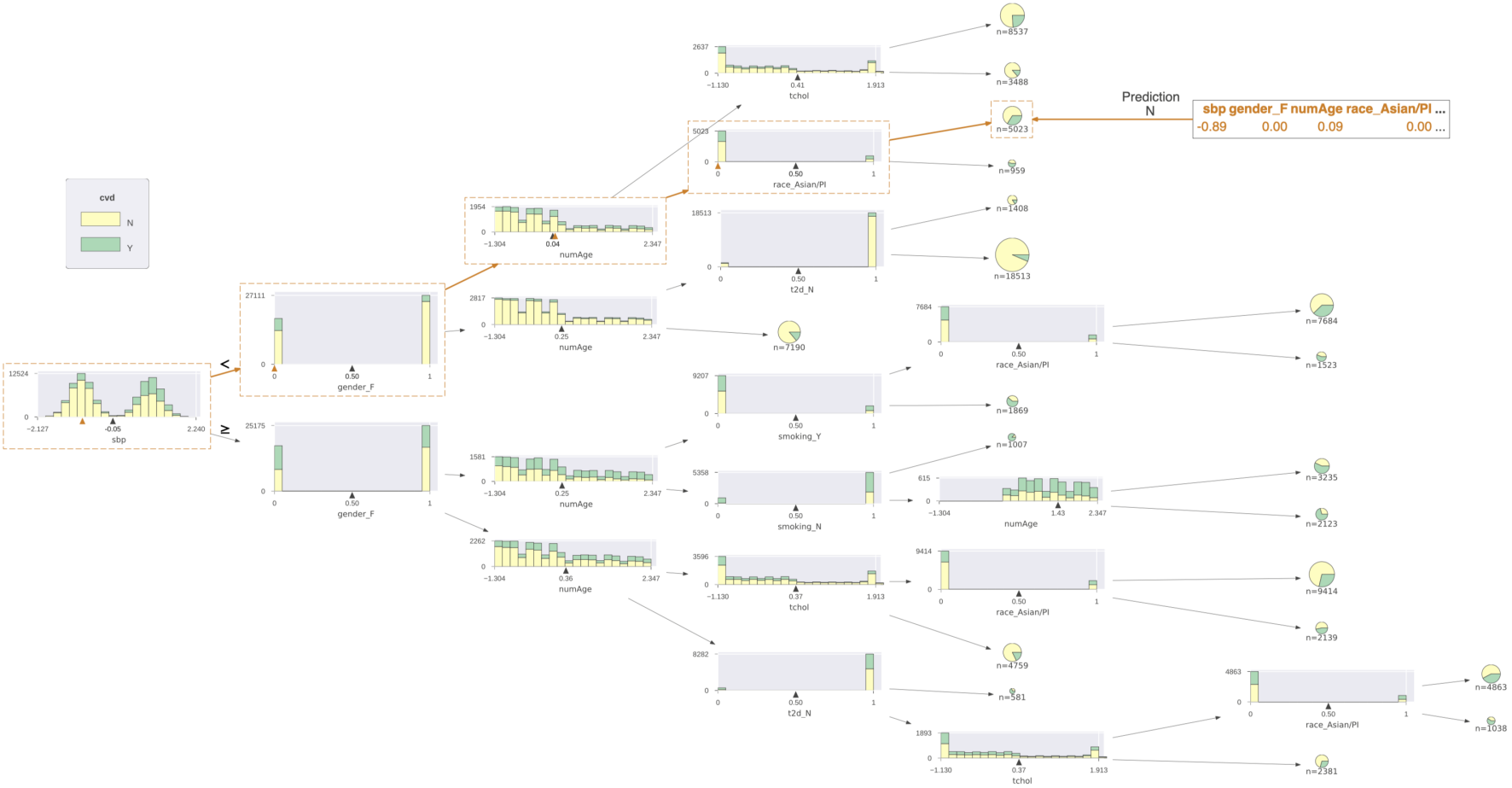

Figura 1.18. Árvore de decisão para a predição de risco cardiovascular. Caminho percorrido para predição de uma instância aleatória obtida do conjunto de teste.

- bootstrap: indica se a técnica de bootstrap será utilizada.

Após a execução do grid, o valores dos hiperparâmetros foram os seguintes:

```
Model with rank: 1 Mean validation score: 0.763 (std: 0.001)
  ↪  Parameters: {'bootstrap': True, 'max_features': 'sqrt',
  ↪  'n_estimators': 775, 'max_depth': 10, 'min_samples_split': 2,
  ↪  'min_samples_leaf': 2}
```

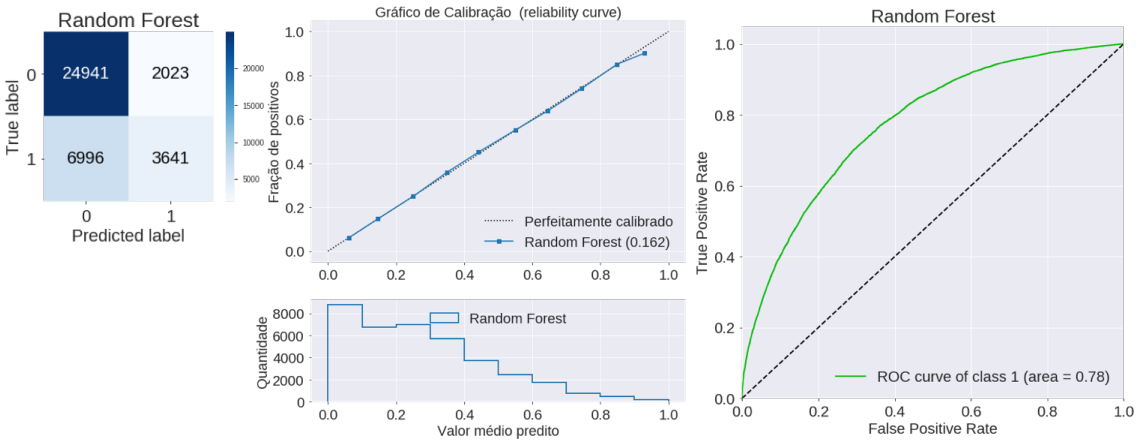

Figura 1.19. Desempenho do modelo de Random Forest.

A Figura 1.19 apresenta o resultado da execução do algoritmo sob o conjunto de teste. O modelo supera os três modelos anteriormente elaborados, com uma AUC de 0,780, e valores superiores de precisão (0,643), embora apresentando menor sensibilidade (0,342) quando comparado à regressão logística (0,363) e K-vizinhos mais próximos (0,349), o que vai ao encontro da relação precisão-sensibilidade existente nos modelos preditivos (DAVIS; GOADRICH, 2006), onde um aumento da precisão leva a uma redução da sensibilidade, ao passo em que um aumento da sensibilidade leva a uma redução da precisão. Em relação a sua eficiência como classificador probabilístico, o modelo apresentou um baixo *brier score* (0,162), superando os modelos anteriormente desenvolvidos.

1.5.5.3. Gradient Boosted Trees

Para o ajuste dos dados ao modelo de Gradient Boosted Trees será feito uso da implementação XGBoost (Extreme Gradient Boosting). Trata-se de uma melhoria do algoritmo Gradient Boosting proposto por Friedman (2001), cujas modificações estão descritas em Chen e Guestrin (2016). XGBoost vem apresentando bons resultados em competições de machine learning, de modo que a teoria para seu funcionamento ainda se encontra em consolidação pela comunidade científica.

Os seguintes hiperparâmetros foram considerados para otimização:

```
params_xgb = {
        'learning_rate': [0.02, 0.03,0.04],
        'n_estimators': [500,700,800],
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [5,10,20,40]
        }
xgb = XGBClassifier(objective='binary:logistic',silent=True,
  ↪  nthread=-1)
gridXGB = GridSearchCV(xgb, param_grid=params_xgb,
                               scoring='AUC', n_jobs=-1, cv=3,
                                 ↪  verbose=1)
```

- learning_rate: utilizada para evitar uma rápida convergência do modelo, ponderando a força do ajuste a ser feito nas novas árvores;
- n_estimators: número máximo de árvores a serem utilizadas pelo modelo;
- min_child_weight: número mínimo de observações em uma folha;
- gamma: parâmetro de regularização do modelo, na busca por prevenir sobreajuste;
- subsample: controla o número de amostras que serão fornecidas ao modelo;
- colsample_bytree: controla o número de variáveis preditoras que serão oferecidas ao modelo;
- max_depth: controla a profundidade máxima da árvore.

Após a execução do grid, a melhor combinação de hiperparâmetros foi a seguinte:

```
XGBClassifier(gamma=0.0, learning_rate=0.1, max_depth=5,
  ↪  min_child_weight=5,
       n_estimators=60,  subsample=0.6)
```

O desempenho do modelo pode ser observado na Figura 1.20. Em relação aos modelos anteriormente elaborados, o XGBoost apresentou AUC (0,780) equivalente ao modelo de Random Forest, porém com valores superiores de precisão (0,647) e sensibilidade (0,344), e com a mesma calibração (0,162).

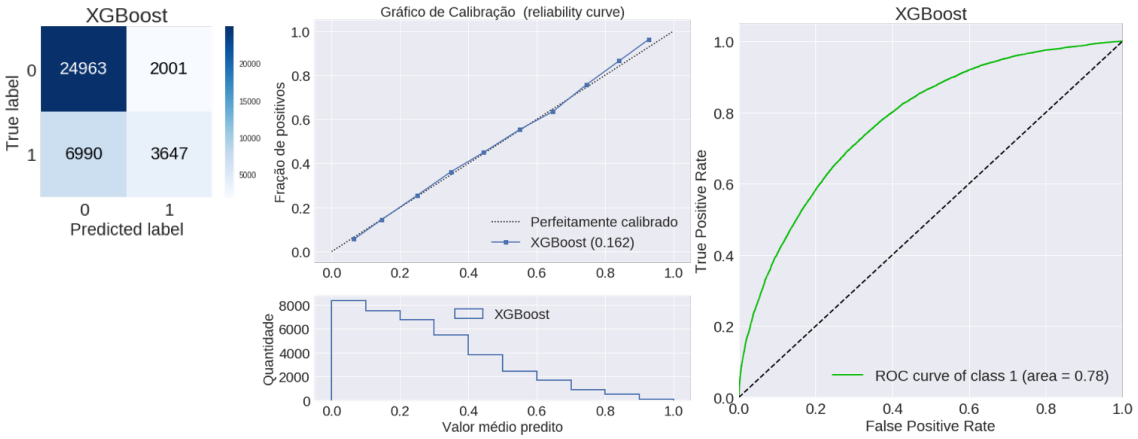

Figura 1.20. Desempenho do modelo XGBoost.

1.5.5.4. Redes Neurais Artificiais

Para a construção da RNA, os seguintes hiperparâmetros foram considerados:

```
parameters_rna = {'activation':['relu', 'tanh'],
                  'solver': ['adam', 'lbfgs', 'sgd'],
                  'max_iter': [500,1000],
                  'alpha': 10.0 ** -np.arange(1, 3),
                  'hidden_layer_sizes':[2,3,5,7],
                  'batch_size':np.arange(5,15,32),
                  'learning_rate_init':[0.01, 0.03, 0.1] }

rna = neural_network.MLPClassifier(verbose=True)
```

- activation: indica qual função de ativação será utilizada pelos neurônios artificiais;
- solver: método utilizado para ajuste dos pesos;
- max_iter: número de épocas de treinamento da rede neural;
- alpha: parâmetro de regularização do modelo;
- hidden_layer_sizes: número de neurônios na camada oculta;
- batch_size: número de exemplos de treinamento usados em uma iteração. Após uma iteração, os pesos da rede são ajustados;
- learning_rate_init: valores iniciais da taxa de aprendizado.

Após a execução do grid, a seguinte combinação de hiperparâmetros obteve o melhor resultado no conjunto de treinamento:

```
Model with rank: 1
Mean validation score: 0.779 (std: 0.004)
Parameters: {'activation': 'tanh', 'alpha': 0.1, 'batch_size': 5,
  ↪ 'hidden_layer_sizes': 7, 'learning_rate_init': 0.1, 'max_iter':
  ↪ 1000, 'solver': 'lbfgs'}
```

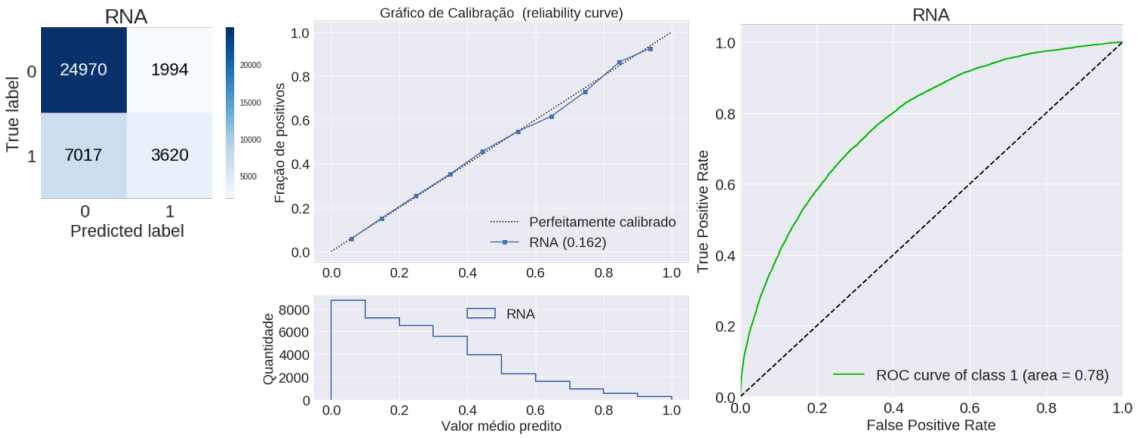

Figura 1.21. Desempenho do modelo de Redes Neurais Artificiais.

A Figura 1.21 apresenta os resultados do modelo quando aplicado ao conjunto de teste. O modelo apresentou a mesma AUC (0,780) que o modelo XGBoost, porém com precisão (0,645) e sensibilidade (0,341) inferiores. Em relação a calibração, o modelo de RNA foi que apresentou a melhor calibração (0,161) dentre todos os modelos elaborados.

A Tabela 1.3 apresenta os resultados consolidados dos modelos elaborados para a predição do risco cardiovascular em pessoas acima de 55 anos. O modelo baseado em um comitê de árvores de decisão e técnica de boosting (Gradient Boosted Trees, implementado pelo XGBoost) apresentou a maior eficiência preditiva dentre os modelos elaborados, seguido pelos modelos de Redes Neurais Artificiais e Random Forest.

Em relação a calibração dos modelos, XGBoost apresentou um bom resultado de calibração. A importância da análise da calibração dos modelos preditivos se dá pela capacidade das probabilidades preditas serem agregadas em níveis de riscos, ajudando a priorizar indivíduos em decisões relacionadas a ações de tratamento ou prevenção.

1.6. Considerações Finais

O aumento expressivo no volume de dados sendo gerados e/ou consumidos por sistemas e aplicações requer a adoção de técnicas voltadas para a análise e a extração de conhecimento, sistemas e metodologias para a gestão dos dados, buscando uma tomada de decisão com base em uma melhor compreensão dos dados.

Conforme enfatizado por Obermeyer e Emanuel (2016), a adoção de algoritmos para a análise e interpretação de conjuntos de dados irá fornecer o ferramental necessário para a compreensão e tomada de decisão. A adoção de técnicas de Machine Learning

Tabela 1.3. Ranking dos modelos preditivos para o risco cardiovascular, ordenados decrescentemente por AUC e Precisão.

Modelo	AUC	Precisão	Sensibilidade	Brier
Gradient Boosted Trees	0,780	0,647	0,344	0,162
Redes Neurais Artificiais	0,780	0,645	0,341	0,161
Random Forest	0,780	0,643	0,342	0,162
K-vizinhos mais próximos	0,770	0,633	0,349	0,164
Regressão Logística	0,770	0,629	0,363	0,163
Árvore de Decisão	0,750	0,617	0,311	0,169

busca dotar os computadores de funções que permitam extrapolar a informação implícita nos dados, não apenas restringindo-se a executar um conjunto de ações pré-determinadas pelo programador.

A área da Saúde apresenta forte potencial para a adoção de Machine Learning. Desde atividades como predição de ocorrência de uma determinada doença, até análises complexas de imagens de ressonâncias 3D, podem se beneficiar da velocidade e capacidade de realizar cálculos n-dimensionais dos algoritmos disponíveis.

Todavia, deixar que os dados falem por si sós não é uma tarefa trivial. Algoritmos podem sofrer sobreajuste para eventuais correlações espúrias presentes nos dados, assim como multicolinearidade das variáveis preditoras pode afetar a capacidade de predição e generalização dos modelos preditivos. Dessa forma, é necessária a adoção de técnicas de pré-processamento que buscam lidar com valores ausentes, *outliers*, dentre outros problemas, de modo a produzir modelos preditivos que apresentem bons resultados diante de novas instâncias a serem classificadas.

A capacidade preditiva de Machine Learning não deve ser confundida com a necessidade de inferência causal (OBERMEYER; EMANUEL, 2016). Modelos de Machine Learning podem apresentar excelentes resultados de predição e destacar quais preditores estão contribuindo significativamente para tal predição. Todavia, a importância de um preditor não necessariamente está ligada com a causa do fenômeno de interesse.

A adoção de Machine Learning nas mais diversas áreas tem sido impulsionada pela disponibilidade e variedade de ferramentas computacionais para o ajuste de modelos preditivos. Este Capítulo utilizou a biblioteca Scikit-Learn (BUITINCK et al., 2013), desenvolvida na linguagem de Programação Python, que apresenta considerável popularidade entre a comunidade científica e profissional.

Foram apresentados modelos preditivos baseados em algoritmos que seguem o paradigma do aprendizado supervisionado. Tais algoritmos possibilitaram o ajuste de modelos preditivos em um conjunto de dados sintéticos sobre o risco cardiovascular, apresentando resultados próximos, embora possuindo princípios de funcionamento diferentes.

Há uma forte tendência da popularização do uso de modelos preditivos de Machine Learning na área da Saúde, assim como nas mais diversas áreas do conhecimento. Conhecer os princípios de funcionamento dos algoritmos disponíveis, as técnicas de pré-processamento que se fazem necessárias antes do ajuste dos modelos, e como interpretar

os resultados obtidos será cada vez mais uma tarefa exigida dos profissionais, independente da sua área de formação e atuação.

Referências

BORBOUDAKIS, G.; STERGIANNAKOS, T.; FRYSALI, M.; KLONTZAS, E.; TSAMARDINOS, I.; FROUDAKIS, G. E. Chemically intuited, large-scale screening of mofs by machine learning techniques. *npj Computational Materials*, Nature Publishing Group, v. 3, n. 1, p. 40, 2017.

BRAGA, A. de P.; FERREIRA, A. C. P. de L.; LUDERMIR, T. B. *Redes neurais artificiais: teoria e aplicações*. [S.l.]: LTC Editora Rio de Janeiro, Brazil:, 2007.

BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J. et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

CHAMBERS, M.; DINSMORE, T. W. *Advanced analytics methodologies: Driving business value with analytics*. [S.l.]: Pearson Education, 2014.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: ACM. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. [S.l.], 2016. p. 785–794.

CHIAVEGATTO FILHO, A. D. P. Uso de big data em saúde no brasil: perspectivas para um futuro próximo. *Epidemiologia e Serviços de Saúde*, SciELO Public Health, v. 24, p. 325–332, 2015.

DAVIS, J.; GOADRICH, M. The relationship between precision-recall and roc curves. In: ACM. *Proceedings of the 23rd international conference on Machine learning*. [S.l.], 2006. p. 233–240.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. C. *Inteligência Artificial: Uma abordagem de aprendizado de máquina*. [S.l.]: Grupo Gen-LTC, 2011.

FAVERO, L. P. L.; BELFIORE, P. P.; SILVA, F. L. d.; CHAN, B. L. *Análise de dados: modelagem multivariada para tomada de decisões*. [S.l.]: Elsevier, 2009.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, JSTOR, p. 1189–1232, 2001.

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. [S.l.]: "O'Reilly Media, Inc.", 2017.

GREEN, M. A. Use of machine learning approaches to compare the contribution of different types of data for predicting an individual's risk of ill health: an observational study. *The Lancet*, Elsevier, v. 392, p. S40, 2018.

HAYKIN, S. *Neural networks: a comprehensive foundation*. [S.l.]: Prentice Hall PTR, 1999.

JIANG, L.; CAI, Z.; WANG, D.; JIANG, S. Survey of improving k-nearest-neighbor for classification. In: IEEE. *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*. [S.l.], 2007. v. 1, p. 679–683.

KAUFMAN, S.; ROSSET, S.; PERLICH, C.; STITELMAN, O. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM, v. 6, n. 4, p. 15, 2012.

KUHN, M.; JOHNSON, K. *Applied predictive modeling*. [S.l.]: Springer, 2013. v. 26.

LADERAS, T.; VASILEVSKY, N.; PEDERSON, B.; HAENDEL, M.; MCWEENEY, S.; DORR, D. Teaching data science fundamentals through realistic synthetic clinical cardiovascular data. *bioRxiv*, Cold Spring Harbor Laboratory, 2018.

MAGALHÃES, M. N.; LIMA, A. C. P. de. *Noções de probabilidade e estatística*. [S.l.]: Editora da Universidade de São Paulo, 2002. v. 5.

MATTMANN, C. a. Computing: A vision for data science. *Nature*, v. 493, n. 7433, p. 473–475, 2013. ISSN 1476-4687. Disponível em: <http://www.nature.com/nature/journal/v493/n7433/full/493473a.html>.

MAYRINK, V. T. d. M. et al. *Avaliação do algoritmo Gradient Boosting em aplicações de previsão de carga elétrica a curto prazo*. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora (UFJF), 2016.

OBERMEYER, Z.; EMANUEL, E. J. Predicting the future—big data, machine learning, and clinical medicine. *The New England journal of medicine*, NIH Public Access, v. 375, n. 13, p. 1216, 2016.

OBERMEYER, Z.; LEE, T. H. Lost in thought—the limits of the human mind and the future of medicine. *New England Journal of Medicine*, Mass Medical Soc, v. 377, n. 13, p. 1209–1211, 2017.

OLIVERA, A. R.; ROESLER, V.; IOCHPE, C.; SCHMIDT, M. I.; VIGO, Á.; BARRETO, S. M.; DUNCAN, B. B. Comparison of machine-learning algorithms to build a predictive model for detecting undiagnosed diabetes-elsa-brasil: accuracy study. *Sao Paulo Medical Journal*, SciELO Brasil, v. 135, n. 3, p. 234–246, 2017.

OLSON, R. S.; CAVA, W. L.; MUSTAHSAN, Z.; VARIK, A.; MOORE, J. H. Data-driven advice for applying machine learning to bioinformatics problems. *arXiv preprint arXiv:1708.05070*, World Scientific, 2017.

PARR, T.; HOWARD, J. *How to explain gradient boosting*. 2018. Disponível em: <https://explained.ai/gradient-boosting/descent.html>.

RASCHKA, S.; MIRJALILI, V. *Python machine learning*. [S.l.]: Packt Publishing Ltd, 2017.

SAKR, S.; ELSHAWI, R.; AHMED, A. M.; QURESHI, W. T.; BRAWNER, C. A.; KETEYIAN, S. J.; BLAHA, M. J.; AL-MALLAH, M. H. Comparison of machine learning techniques to predict all-cause mortality using fitness data: The Henry Ford exercIse testing (FIT) project. *BMC Medical Informatics and Decision Making*, BMC Medical Informatics and Decision Making, v. 17, n. 1, p. 1–15, 2017. ISSN 14726947.

SELTZER, M. L.; ZHANG, L. The Data Deluge: Challenges and Opportunities of unlimited data in statistical signal processing. p. 3701–3704, 2009.

WENG, S. F.; REPS, J.; KAI, J.; GARIBALDI, J. M.; QURESHI, N. Can machine-learning improve cardiovascular risk prediction using routine clinical data? *PloS one*, Public Library of Science, v. 12, n. 4, p. e0174944, 2017.