Escrever título (escolher no final)

Universidade Federal da Paraíba - CCEN

Gabriel de Jesus Pereira

7 de agosto de 2024

Índice

Lista de algoritmos

Lista de Figuras

1 Resumo

2 Capítulo 1

– Fazer antes da conclusão –

2.1 Introdução

2.2 Objetivos

2.2.1 Objetivo Geral

2.2.2 Objetivos Específicos

2.3 Organização do Trabalho

3 Capítulo 2

— Fazer depois dos modelos baseados em árvore —

3.1 Recursos Computacionais

3.1.1 Linguagem de Programação R

3.1.2 Linguagem de Programação Python

3.1.3 Quarto

3.1.4 Linguagem de Programação Python

3.1.5 Web Scraping

4 Algoritmos de Aprendizado de Máquina

Neste capítulo, serão descritos os algoritmos de aprendizado de máquina utilizados neste trabalho. Alguns dos métodos utilizados podem fazer uso de diversos algoritmos ou modelos estatísticos. No entanto, o foco principal e o mais utilizado foram as árvores de decisão, especialmente em sua forma particular, as árvores de regressão. Assim, os algoritmos descritos são métodos baseados em árvores.

Os métodos baseados em árvore envolvem a estratificação ou segmentação do espaço dos preditores[1] em várias regiões simples. Dessa forma, todos os algoritmos utilizados neste trabalho partem dessa ideia. Portanto, o primeiro a ser explicado será o de árvores de decisão, pois fundamenta todos os outros algoritmos. Depois das árvores de decisão, serão explicados os métodos ensemble e, por fim, diferentes variações do método de gradient boosting.

4.1 Árvores de decisão

Árvores de decisão são métodos de aprendizado supervisionado não paramétrico utilizado tanto para regressão quanto para classificação. Elas servem de base para muitos dos modelos baseados em árvores empregados neste trabalho, uma vez que esses modelos geram múltiplas árvores de decisão. IZBICKI; SANTOS (2020) define o processo de construção de uma árvore como o particionamento recursivo no espaço das covariáveis, em que cada particionamento recebe o nome de nós e o resultado final recebe o nome de folha. Em cada nó é definida uma condição e, caso essa condição seja satisfeita, ter-se-á como resultado uma das folhas desse nó. Não obstante, caso o resultado seja contrário, seguirá para o próximo nó e verificará a próxima condição, podendo gerar uma folha ou a condição de outro nó. Veja um exemplo na Figura 4.1:

Descrevendo formalmente o processo de construção de uma árvore de regressão[2], a sua execução é composta por dois passos. No primeiro passo, dividimos o espaço dos preditores em J regiões distintas e disjuntas denotadas por $R_1, R_2, ..., R_J$. No segundo

[1] O espaço dos preditores é o conjunto de todos os valores possíveis para as variáveis independentes $\mathbf{x}$

[2] Uma árvore de regressão é um caso específico da árvore de decisão, mas para regressão.

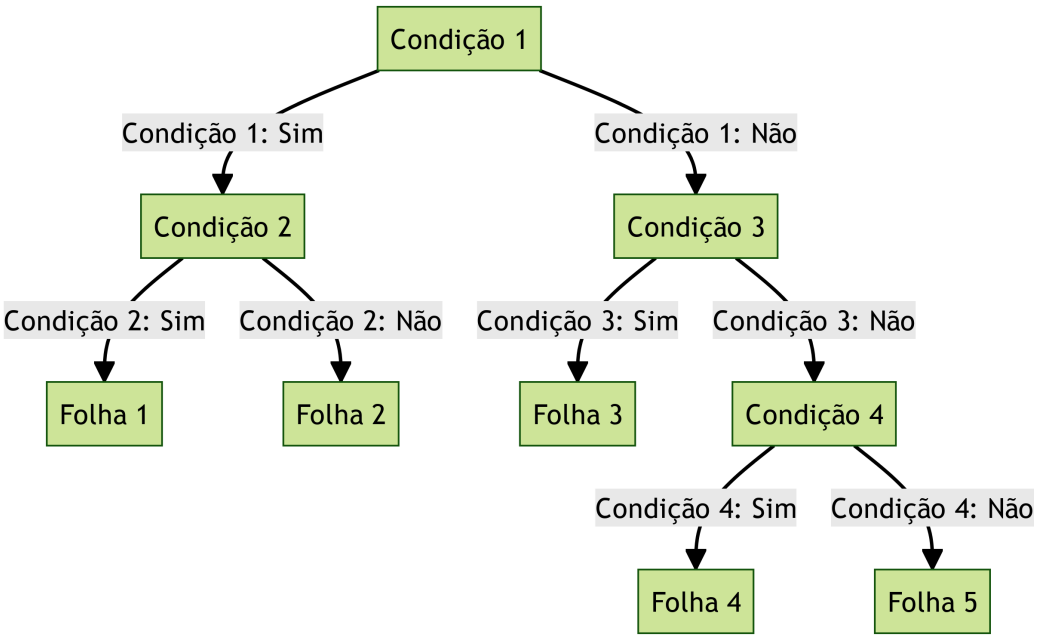

Figura 4.1: Exemplo de estrutura de árvore de regressão. A árvore tem cinco folhas e quatro nós internos.

passo, para cada observação que pertence a região R_j, a previsão será a mesma. Essa previsão será simplesmente a média dos valores da variável dependente das observações de treinamento que estão dentro da região R_j (JAMES *et al.*, 2013). Dessa forma, dado que se tenha duas regiões R_1 e R_2, e a média dos valores da variável resposta tenha sido 10 e 20, respectivamente. Então, Para uma observação $X = x, x \in R_1$, o valor previsto será 10 e, caso contrário, se $x \in R_2$, o valor previsto será 20.

As regiões $R_1, R_2, ..., R_J$ são construídas em formato de caixa de forma a minimizar a soma dos quadrados. Dessa forma, podemos modelar a variável resposta como uma constante c_m em cada região R_j. Assim, definimos a resposta:

$$f(x) = \sum_{j=1}^{J} c_j I\left(x \in R_j\right)$$

Agora, utilizando o critério de minimizar a soma dos quadrados, deve-se minimizar $\sum_{x_i \in R_j} \left[y_i - f\left(xi\right)\right]^2$ para encontrar um estimador para o parâmetro c_j. No entanto, perceba que $f\left(x_i\right)$ está sendo avaliado somente em um ponto específico x_i, o que reduzirá $f\left(x_i\right)$ para uma constante c_j. É fácil de se chegar ao resultado se observarmos a definição da função indicadora $I\left(x \in R_j\right)$:

$$I_{R_j}(x_i) = \begin{cases} 1, & \text{se } x_i \in R_j \\ 0, & \text{se } x_i \notin R_j \end{cases}$$

e, como um ponto x_i não pode estar ao mesmo tempo em duas regiões R_j, pois as regiões são disjuntas, temos que apenas um dos casos a função indicadora será diferente de 0. Portanto, $f(x_i) = c_j$. Assim, derivando $\sum_{x_i \in R_j} (y_i - c_j)^2$ em relação a c_j

$$\frac{\partial \sum_{x_i \in R_j} (y_i - c_j)^2}{\partial c_j} = -2 \sum_{x_i \in R_j} (y_i - c_j) \tag{4.1}$$

e igualando Equação 4.1 a 0, temos a seguinte equação

$$\sum_{x_i \in R_j} (y_i - \hat{c}_j) = 0$$

que se abrirmos o somatório e dividirmos pelo número total de pontos N_j na região R_j, teremos que o estimador de c_j será somente a média dos y_i na região R_j:

$$\sum_{x_i \in R_j} y_i - c_j \sum_{x_i \in R_j} 1 = 0 \Longrightarrow \hat{c}_j = \frac{1}{N_j} \sum_{x_i \in R_j} y_i \tag{4.2}$$

No entanto, JAMES *et al.* (2013) caracteriza como inviável considerar todas as possíveis partições do espaço das variáveis em J caixas devido ao alto custo computacional. Dessa forma, a abordagem a ser adotada é uma divisão binária recursiva. O processo começa no topo da árvore de regressão, o ponto em que contém todas as observações, e continua sucessivamente dividindo o espaço dos preditores. As divisões são indicadas como dois novos ramos na árvore, como pode ser visto na Figura 4.1.

Para executar a divisão binária recursiva, deve-se primeiramente selecionar a variável independente X_j e o ponto de corte s tal que a divisão do espaço dos preditores conduza a maior redução possível na soma dos quadrados dos resíduos. Dessa forma, definimos dois semi-planos

$$R_1(j, s) = \{X | X_j \leq s\} \text{ e } R_2(j, s) = \{X | X_j > s\}$$

e procuramos a divisão da variável j e o ponto de corte s que resolve a equação

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

em que c_1 e c_2 é a média da variável dependente para as observações de treinamento nas regiões $R_1(j,s)$ e $R_2(j,s)$, respectivamente. Assim, encontrando a melhor divisão, os dados são particionados nas duas regiões resultantes e o processo de divisão é repetido em todas as outras regiões.

O tamanho da árvore pode ser considerado como um hiperparâmetro para regular a complexidade do modelo pois, uma árvore muito grande pode causar um ajuste excessivo aos dados de treinamento, capturando não apenas os padrões relevantes, mas também os ruído. Dessa forma, o modelo apresenta um bom desempenho nos dados de treinamento, mas não consegue desempenhar bem em dados que não foram observados devido a sua incapacidade de generalizar. Não obstante, uma árvore pequena pode não capturar os padrões, relações e estruturas importantes contidas nos dados. Dessa forma, adotamos como estratégia para selecionar o tamanho da árvore é crescer uma grande árvore T_0 e interromper o processo de divisão apenas quando atingir um tamanho mínimo de nós. No fim, a grande árvore T_0 é podada usando o critério custo de complexidade, que será definida a seguir.

Para o processo de poda da árvore, definimos uma árvore qualquer T que pode ser obtida através do processo de poda de T_0 e portanto $T \subset T_0$. Dessa forma, sendo N_j a quantidade de pontos na região R_j, seja

$$\hat{c}_j = \frac{1}{N_j} \sum_{xi \in R_j} y_i$$

$$Q_j(T) = \frac{1}{N_j} \sum_{x_i \in R_j} \left(y_i - \hat{c}_j\right)^2$$

$\hat{c}_j$ a média das observações da variável dependente na R_j do nó interno j e $Q_j(T)$ uma estatística medidas de impureza do nó pelo erro quadrático. Assim, definimos o critério custo de complexidade:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_j Q_j(T) + \alpha|T|$$

em que $|T|$ denota a quantidade total de folhas e $\alpha \geq 0$ é o parâmetro de tunagem que equilibra o tamanho da árvore e a adequação aos dados de forma que para cada α, a árvore $T_\alpha \subseteq T_0$ minimiza $C_\alpha(T)$. Valores grandes para α resulta em árvores menores, valores menores resulta em árvores maiores e $\alpha = 0$ resulta na própria árvore T_0. A procura por T_α consiste em sucessivamente colapsar o nó interno que produz o menor aumento em $\sum_j N_j Q_j(T)$ e o processo continua até produzir uma árvore de um único nó. Esse processo gera uma sequência de subárvores que contém uma única menor subárvore que, para cada α, minimiza $C_\alpha(T)$. Além disso, a estimação de α é feita por validação

cruzada com cinco ou dez folds, e a estimativa $\hat{\alpha}$ é escolhida para minimizar a soma dos quadrados dos resíduos durante a validação cruzada. Assim, a árvore final será $T_{\hat{\alpha}}$. O Algoritmo 4.1 exemplifica o processo de crescimento de uma árvore de regressão:

Algoritmo 4.1 Algoritmo para crescer uma árvore de regressão

1. Use a divisão binária recursiva para crescer uma árvore grande T_0 nos dados de treinamento, parando apenas quando cada folha tiver menos do que um número mínimo de observações.

2. Aplique o critério custo de complexidade à árvore grande T_0 para obter uma sequência de melhores subárvores T_α, em função de α.

3. Use validação cruzada K-fold para escolher α. Isto é, divida as observações de treinamento em K folds. Para cada $k = 1, \ldots, K$:

(a) Repita os Passos 1 e 2 em todos os folds, exceto no k-ésimo fold dos dados de treinamento.

(b) Avalie o erro quadrático médio de previsão nos dados no k-ésimo fold deixado de fora, em função de α. Faça a média dos resultados para cada valor de α e escolha α que minimize o erro médio.

4. Retorne a subárvore $T_{\hat{\alpha}}$ do Passo 2 que corresponde ao valor estimado de α.

Algoritmo 4.1: Fonte: JAMES *et al.* (2013, p. 337).

4.2 Métodos Ensemble

As árvores de decisão são conhecidas por sua alta interpretabilidade, mas geralmente apresentam um desempenho preditivo preditivo inferior em comparação com outros modelos e algoritmos. No entanto, é possível superar essa limitação construindo um modelo preditivo que combina a força de uma coleção de modelos base mais simples, um processo conhecido como aprendizado Ensemble. De acordo com HASTIE *et al.* (2009), o aprendizado Ensemble pode ser dividido em duas etapas principais: a primeira etapa consiste em desenvolver uma população de algoritmos de aprendizado base a partir dos dados de treinamento, e a segunda etapa envolve a combinação desses algoritmos para formar um estimador agregado. Portanto, nesta seção, serão definidos os métodos de aprendizagem Ensemble utilizados neste trabalho. O foco inicial será no Random Forest, seguido pela descrição e explicação teórica do Bagging, Boosting, Stacking e outros algoritmos de aprendizado de máquina.

4.2.1 Bagging

O algoritmo de Bootstrap Aggregation, ou Bagging, foi introduzido por BREIMAN (1996). Sua ideia principal é gerar um estimador agregado a partir de múltiplas versões de um preditor, que são formadas fazendo réplicas bootstrap do conjunto de treinamento e utilizando-as como novos conjuntos de treinamento. O Bagging pode ser utilizado como uma forma de melhorar a estabilidade, precisão de modelos ou algoritmos de aprendizado de máquina, além de diminuir a variância e evitar sobreajuste. Por exemplo, o Bagging poderia ser utilizada para melhorar a árvore de regressão que foi descrita anteriormente, mas também poderia ser aplicado a outros métodos.

BREIMAN (1996) define formalmente o algoritmo de Bagging, no qual temos um conjunto de treinamento $\mathcal{L}$. Tomamos amostras bootstrap $\mathcal{L}^{(B)}$ com B réplicas de $\mathcal{L}$ para formar $\{\varphi\left(x, \mathcal{L}^{(B)}\right)\}$, onde φ denota um modelo ou algoritmo treinado nas amostras bootstrap $\{\mathcal{L}^{(B)}\}$ com variáveis independentes x para previsão ou classificação de uma variável dependente y. Caso a variável dependente y seja numérica, a predição é feita tomando a média de $\varphi\left(x, \mathcal{L}^{(B)}\right)$. Assim, temos que a predição é feita da seguinte forma:

$$\varphi_B\left(x\right) = \frac{1}{B}\sum_{b=1}^{B} \varphi\left(x, \mathcal{L}^{(B)}\right)$$

onde φ_B denota a agregação. Não obstante, se y prediz uma classe, utilizamos a votação majoritária. Ou seja, se estivermos classificando classes $j \in 1, \ldots, J$, então podemos tomar $N_j = \#\{B; \varphi\left(x, \mathcal{L}^{(B)}\right) = j\}$ que representa a quantidade de vezes que a classe j foi predita pelos estimadores. Assim, tomamos

$$\varphi_B\left(x\right) = \arg\max_j N_j$$

isto é, o j para o qual N_j é máxima.

Embora a técnica de Bagging tenha o poder de melhorar o desempenho de uma árvore de regressão ou classificação, isso é alcançado ao custo de menor interpretabilidade. No caso da aplicação de Bagging para uma árvore de regressão, construímos B árvores de regressão usando B réplicas de amostra bootstrap e tomamos a média das predições resultantes (JAMES *et al.*, 2013). Nesse caso, as árvores de regressão crescem ao máximo, sem passar pelo processo de poda, resultando em cada árvore individual com alta variância, mas baixo viés. No entanto, ao agregarmos φ_B das B árvores, reduzimos a variância. Para mitigar a falta de interpretabilidade do método Bagging aplicado a uma árvore de regressão, podemos utilizar a soma do quadrado dos resíduos como uma estatística da importância das variáveis independentes. Um valor elevado da redução total média da soma do quadrado dos resíduos devido às divisões

em um determinado preditor, calculada sobre todas as árvores B, indica um preditor importante.

As árvores construídas pelo algoritmo de árvore de decisão são beneficiadas pela proposta de agregação do Bagging, mas esse benefício é limitado devido à correlação positiva existente entre as árvores. Se as árvores forem variáveis aleatórias independentes e identicamente distribuídas, cada uma com variância σ^2, a variância da média das variáveis aleatórias das B árvores será $\frac{1}{B}\sigma^2$. Não obstante, se forem apenas identicamente distribuídas, mas não necessariamente independente, com correlação pareada positiva ρ, a esperança da média das B árvores é a mesma que a esperança de uma árvore individual. Portanto, o viés do agregado das árvores é o mesmo das árvores individuais, e a única melhora possível é através da redução de sua variância. Assim, a variância da média será

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{4.3}$$

Isso significa que, à medida que a quantidade de árvores B aumenta, o segundo termo da soma desaparece. Portanto, os benefícios da agregação ocasionada pelo algoritmo de Bagging são limitados pela correlação das árvores (HASTIE *et al.*, 2009). Ou seja, mesmo ao aumentar o número de árvores no Bagging, a correlação entre elas faz com que as previsões individuais não sejam completamente independentes, então a variância da média das previsões não diminui tão rapidamente quanto seria esperado se as árvores fossem completamente independentes. Uma forma de melhorar o algoritmo de Bagging foi através da Random Foreste, que será descrita adiante.

4.2.2 Random Forest

O algoritmo da Random Forest é uma técnica derivada do método de Bagging, com algumas modificações para a construção das árvores. As árvores são construídas de forma a melhorar a redução da variância diminuindo a correlação entre as árvores, sem aumentar significantemente a variabilidade. Isso é alcançado no processo de crescimento das árvores por meio da seleção aleatória das variáveis independentes.

Ao construir uma árvore em amostras bootstrap no algoritmo da Random Forest, antes de cada divisão, selecionam-se aleatoriamente $m \leq p$ ($m = p$ é o caso do algoritmo de Bagging) variáveis independentes como candidatas para a divisão. No entanto, apenas um desses m preditores é utilizado para a divisão. Com base em algum critério, como a minimização da impureza, selecionamos o melhor preditor possível para realizar a divisão. Portanto, diferente do Bagging, que criava árvores de decisão muito semelhantes e, portanto, resultava em predições altamente correlacionadas, a Random Forest busca minimizar esse problema dando chances a outros preditores. Dessa forma, em média,

$(p-m)/p$ das divisões nem sequer considerarão o preditor mais forte, permitindo que outros preditores também tenham chance de serem utilizados (JAMES *et al.*, 2013). Esse processo de diminuir a correlação entre as árvores torna a média das árvores resultantes menos variável e, portanto, mais confiável.

A quantidade de variáveis independentes mm selecionadas aleatoriamente pode ser considerada um parâmetro a ser ajustado por meio de validação cruzada. No entanto, de acordo com HASTIE *et al.* (2009), os inventores do algoritmo recomendam os seguintes valores padrão: $m = \sqrt{p}$ e tamanho mínimo do nó igual a um para classificação, e $m = p/3$ e tamanho mínimo do nó igual a cinco para regressão. Quando o número de variáveis é grande, mas poucas variáveis são relevantes, o algoritmo Random Forest pode não ter um bom desempenho com valores pequenos de m, pois isso reduz as chances de selecionar as variáveis relevantes. No entanto, utilizar um valor pequeno para m pode ser útil quando há muitos preditores correlacionados. Além disso, assim como no algoritmo de Bagging, a Random Forest não sofre de sobreajuste se o valor de B for aumentado. Portanto, basta usar um B suficientemente grande para que a taxa de erro se estabilize (JAMES *et al.*, 2013). O processo de construção de uma Random Forest pode ser visualizado no Algoritmo 4.2 .

4.2.3 Boosting Trees

O Boosting, assim como o Bagging mostrado anteriormente, é uma metodologia que pode ser aplicada para melhorar a performance de um modelo ou algoritmo. No entanto, neste trabalho, o Boosting foi aplicado apenas utilizando árvores de decisão, portanto, sua descrição será restrita a esse caso, conhecidas como Boosting Trees.

No algoritmo de Bagging cada árvore era construída e ajustada utilizando amostras bootstrap. Por fim, era necessário tomar um agregado φ_B de todas as B árvores para criar um único estimador. O Boosting Trees funciona de forma similar, no entanto, cada árvore é construída utilizando uma versão modificada dos dados de treinamento original, sem a necessidade de amostras bootstrap, e a informação prévia de outras árvores. Ou seja, são construídas sequenciamente.

Para o caso da regressão, o Boosting, assim como o Bagging, combina um grande número de árvores de decisão $\hat{f}^1, \ldots, \hat{f}^B$. Assim, a primeira árvore é construída utilizando o conjunto de dados originais e os seus resíduos são calculados. Com a primeira árvore construída, a segunda árvore é ajustada para prever esses resíduos e é adicionada ao estimador ajustado para atualizar os seus resíduos. Portanto, para a regressão, os resíduos funcionam como uma informação para construir novas árvores e corrigir os erros das árvores anteriores. Ainda, como para construir cada árvore depende de árvores que já foram construídas, árvores pequenas são suficientes (JAMES *et al.*, 2013).

O processo de aprendizado na metodologia do Boosting é lenta, o que acaba gerando melhores resultados. Esse processo de aprendizado pode ser controlado por um hiperparâmetro λ chamado de shrinkage, permitindo que mais árvores, com formas diferentes, corrijam os erros de árvores passadas. No entanto, um valor muito pequeno para λ requer uma quantidade muito maior B de árvores e, diferente do Bagging e Random Forest, o Boosting pode sofrer de sobreajuste se a quantidade de árvores é muito grande. Além disso, a quantidade de divisões d em cada árvore, que controla a complexidade do boosting, pode ser considerado também um hiperparâmetro. Para $d = 1$ é ajustado um modelo aditivo, já que cada termo involve apenas uma variável. JAMES *et al.* (2013) define d como a profundidade de interação que controla a ondem de interação do modelo boosting, já que d divisões podem envolver no máximo d variáveis. Uma versão simplificada do algoritmo pode ser visualizado em Algoritmo 4.3 .

4.2.4 Stacked generalization

O Stacked generalization, ou Stacking, é um método ensemble que envolve treinar um modelo que combina as predições de vários outros algoritmos para melhorar a predição. Esse método pode funcionar com qualquer modelo estatístico ou algoritmo de aprendizagem de máquina. A ideia principal é incluir peso nas predições de forma a dar maior importância para aqueles que geraram melhores predições e ao mesmo tempo não dar altos pesos àqueles modelos que tem alta complexidade.

Matematicamente, o Stacking define predições $\hat{f}_m^{-i}(x)$ em x, utilizando o modelo estatístico ou algoritmo m, aplicado ao conjunto de treinamento com a $i - sima$ observação removida (HASTIE *et al.*, 2009). Assim, os peso são estimados da seguinte forma

$$\hat{w}^{st} = \arg\min_{w} \sum_{i=1}^{N}$$

4.2.5 Gradient Boosting

O método de Gradient Boosting constrói modelos de regressão aditivos ajustando sequencialmente uma função base aos resíduos, que são os gradientes da função de perda do modelo atual (FRIEDMAN, 2002). Estes gradientes representam a direção na qual a função de perda deve ser minimizada. Existem outras implementações de Gradiente Boosting que foram utilizadas nesse trabalho. No entanto, todas elas utilizam o Gradient Boosting com árvores de regressão, mas com algumas modificações para melhorar a eficiência do algoritmo já existente. O algoritmo do gradient boosting aplicado para árvores de regressão, que será explicado, pode ser visualizado no Algoritmo 4.4 .

O Gradient Boosting aplicado para árvores de regressão, tem que cada função base é um caso especial de uma árvore de regressão com J_m folhas. A primeira linha do algoritmo Algoritmo 4.4 inicializa com uma constante ótima, que é simplesmente uma árvore de regressão com uma única folha. No caso do gradient boosting aplicado a árvores de regressão, cada árvore de regressão tem a forma aditiva

$$h_m\left(x;\{b_j,R_j\}_1^J\right)=\sum_{j=1}^{J_m} b_{jm} I\left(x\in R_{jm}\right) \tag{4.4}$$

em que $\{R_{jm}\}_1^{J_m}$ são as regiões disjuntas que, coletivamente, cobrem o espaço de todos os valores conjuntos das variáveis preditoras x. Essas regiões são representadas pelas folhas de sua correspondente árvore. Como as regiões são disjuntas, Equação 4.4 se reduz simplesmente a $h_m(x)=b_{jm}$ para $x\in R_{jm}$. Por mínimos quadrados, b_{jm} é simplesmente a média dos pseudo-resíduos $\tilde{y}_i$,

$$\hat{b}_{jm}=\frac{1}{N_{jm}}\sum_{x_i\in R_{jm}}\tilde{y}_i$$

que dão a direção de diminuição da função perda L pela expressão do gradiente da linha 2(a). Assim, cada árvore de regressão é ajustada aos $\tilde{y}_i$ de forma a minimizar o erro das árvores anteriores. N_{jm} denota a quantidade de pontos na região R_{jm}. Por fim, a atualização da árvore de regressão é expressa da seguinte forma

$$f_m(x)=f_{m-1}(x)+\lambda\sum_{j=1}^{J}\gamma_{jm} I\left(x\in R_{jm}\right)$$

em que γ_{jm} representa a atualização da constante ótima para cada região, baseado na função de perda L, dada a aproximação $f_{m-1}(x)$. O , assim como no algoritmo de boosting, representa o hiperparâmetro shrinkage para controlar a taxa de aprendizado. Pequenos valores de λ necessitam maiores quantidades de iterações M para diminuir o risco de treinamento.

4.2.6 Diferentes implementações de Gradient Boosting

Algoritmo 4.2 Algoritmo de uma Random Forest para regressão ou classificação

1. Para b = 1 até B:

(a) Construa uma amostra bootstrap Z^* de tamanho N dos dados de treinamento.
(b) Faça crescer uma árvore de floresta aleatória T_b para os dados bootstrap, repetindo recursivamente os seguintes passos para cada folha da árvore, até que o tamanho mínimo do nó n_{min} seja atingido.

i. Selecione m variáveis aleatoriamente entre as p variáveis.
ii. Escolha a melhor variável entre as m.
iii. Divida o nó em dois subnós.

2. Por fim, o conjunto de árvores $\{T_b\}_1^B$ é construído.

No caso da regressão, para fazer uma predição em um novo ponto x, temos a seguinte função:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

Para a classificação é utilizado o voto majoritário. Assim, seja $\hat{C}_b(x)$ a previsão da classe da árvore de floresta aleatória b. Então,

$$\hat{C}_{rf}^{B}(x) = \arg\max_{c} \sum_{b=1}^{B} I\left(\hat{C}_b(x) = c\right)$$

onde c representa as classes possíveis.

Algoritmo 4.2: Fonte: HASTIE *et al.* (2009, p. 588).

Algoritmo 4.3 Método Boosting aplicado a árvores de regressão

1. Defina $\hat{f}(x) = 0$ e $r_i = y_i$ para todos os i no conjunto de treinamento

2. Para $b = 1, 2, \ldots, B$, repita:

(a) Ajuste uma árvore $\hat{f}^b$ com d divisões para os dados de treinamento (X, r).
(b) Atualize $\hat{f}$ adicionando uma versão com o hiperparâmetro λ de taxa de aprendizado:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

(c) Atualize os resíduos,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Retorne o modelo de boosting,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Algoritmo 4.3: Fonte: JAMES *et al.* (2013, p. 349).

Algoritmo 4.4 Gradient Tree Boosting

1. Inicialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$

2. Para $m = 1$ até M:

(a) Para $i = 1, 2, \ldots, N$, calcule

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

(b) Ajuste uma árvore de regressão aos pseudo-resíduos r_{im}, obtendo regiões terminais

$R_{jm},\ j = 1, 2, \ldots, J.$

(c) Para $j = 1, 2, \ldots, J_m$, calcule

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

(d) Atualize $f_m(x) = f_{m-1}(x) + \lambda \sum_{j=1}^{J} \gamma_{jm} I\left(x \in R_{jm}\right)$

3. Retorne $\hat{f}(x) = f_M(x)$

Algoritmo 4.4: Fonte: HASTIE *et al.* (2009)

5 Metodologia

5.1 Os dados e o procedimento adotado para sua obtenção

— AINDA SERÁ MODIFICADO —

O Web scraping, também conhecido como extração de dados da web, é uma técnica utilizada para o processo de coleta de dados estruturados da web de maneira automatizada. É um processo que vem sendo constantemente utilizado por instituições públicas e privadas para a construção de produtos que utilizam algoritmos de aprendizagem de máquinas, observa ofertas e discontos, faz análise de mercado ou monitoração de marcas.

Neste projeto, para fins de estudo e análise do mercado imobiliário, os dados foram coletados por meio de extração de dados do site do Zap Imóveis. O Zap Imóveis é um site do Grupo OLX que reúne ofertas do mercado imobiliário e que funciona como uma plataforma dinâmica para facilitar a conexão entre quem deseja alugar, comprar ou vender um imóvel; podendo servir também para corretores ou outros profissionais do setor de imóveis. Este projeto foi possível graças as informações que foram coletadas do site do Zap Imóveis em dois diferentes períodos do ano de 2023. O primeiro deles, as informações foram coletadas utilizando variados pacotes para raspagem de dados e proxies rotativas da linguagem de programação R, a fim de evitar ser bloqueado pelos mecanismos de segurança do site. Na segunda etapa, os dados foram coletados empregando a linguagem de programação Python com as bibliotecas Scrapy e Playwrite, que serve para web crawling e web scraping, e o Playwrite que serve para testes em aplicativos da web, mas que neste caso foi utilizado para manejar páginas dinâmicas.

Desta forma, com a ideia de modelar o valor do imóvel e analisar o mercado imobiliário, foram coletados aqueles variáveis que estavam disponíveis no site do Zap Imóveis e que poderiam de alguma forma ser significativas ao tentar explicar o valor do imóvel durante a sua modelagem. Assim, no total foram coletatas 23 variáveis, das quais 8 são quantitativas e 15 qualitativas nominais, sendo 13 de caráter dicotômico. No entanto, nem todas essas variáveis foram coletadas diretamente do Zap Imóveis, a latitude e longitude foram obtidas pela geocodificação do endereço utilizando o pacote tidygeocoder da linguagem de programação R. Portanto, temos as seguites variáveis:

- Valor do imóvel: esta é a variável dependente, aquela que será modelada e será o principal objeto de estudo deste trabalho;
- Área: área do imóvel em m^2;
- Condomínio: valor pago pelo condomínio;
- IPTU: imposto cobrado de quem tem um imóvel urbano;
- Banheiro: quantidade de banheiros presentes na propriedade;
- Vaga de estacionamento: quantidade total de vagas de estacionamento;
- Quarto: quantidade de quartos no imóvel;
- Latitude: posição horizontal medida em frações decimais de graus;
- Longitude: posição vertical que, assim como a latitude, é medida em frações decimais de graus;
- Tipo do imóvel: foram obtidos 7 tipos de imóveis, apartamentos, casas, casas comerciais, casas de condomínio, casas de vila, coberturas, lotes comerciais e de condomínio;
- Endereço: nome do endereço do imóvel;
- Variáveis dicotômicas que indicam se o imóvel tem ou não aquela característica (representado como 1 ou 0, respectivamente): área de serviço, academia, elevador, espaço gourmet, piscina, playground, portaria 24 horas, quadra de esporte, salão de festa, sauna, spa e varanda gourmet.

No entanto, devido a observações feitas durante o estudo, nem todas essas variáveis foram utilizadas para a modelagem do valor dos imóveis, seja por conter muitos valores valores ausentes ou por não ter se mostrado significante para o que se desejava explicar. Ainda, como a coleta destes dados foram feitas em dois momentos distintos, temos dois bancos de dados, um com 29712 observações e o outro com 14956. Por fim, essas duas bases de dados foram unidas e, para não correr o risco de conter imóveis repetidos, aqueles que tinham o mesmo número de identificação foram removidos.

5.2 Descritiva dos dados

A análise exploratória de dados marca uma das primeiras etapas de qualquer estudo que utiliza a estatística como uma de suas principais ferramentas, pois permite encontrar padrões de comportamento no dados, descobrir relações entre as variáveis estudadas. Dessa forma, a primeira etapa desse estudo, após a coleta e organização dos dados obtidos do Zap Imóveis, foi fazer uma descritiva dos dados. Essa etapa permitiu encontrar padrões nos diferentes tipos de imóveis bem como o seu tipo pode influenciar na características do imóvel, o que, por consequência, pode afetar o seu valor. Assim, para identificar esses diferentes comportamentos, foram criados gráficos e tabelas a fim de caracterizar as relações das variáveis independentes com a dependente.

5.3 Aprendizado Supervisionado e não supervisionado

Na aprendizagem de máquinas, uma das estapas mais importantes é saber qual técnica será utilizada para resolver um problema que se enquadra em diferentes formas de aprendizado. Para isso, existem mais de uma forma em que um algoritmo consegue utilizar os dados e explicar o que está sendo modelado a partir deles. No entanto, a maioria dos problemas de aprendizado de máquinas recais em dois casos mais conhecidos: aprendizado supervisionado e não supervisionado.

5.3.1 Aprendizado supervisionado

Suponha uma regressão logística. Sabemos que na regressão logistíca temos um modelo com a seguinte forma $Y_i = f(X) + \epsilon$, em que Y_i assume 0 ou 1 para classificar o que está sendo modelado e representa a variável dependente, $f(X)$ representa as variáveis independentes que serão utilizadas para a modelagem e ϵ representa o erro da regressão. Dessa forma, podemos considerar o caso em que a regressão logística tenta classificar pacientes que podem ou não estar com diabetes. Para isso, utilizariamos variáveis significativas para a classificação do estado de cada paciente. Esse exemplo é conhecido como aprendizagem supervisionada. Na aprendizagem supervisionada, busca-se aprender Y_i atráves de um exemplo. Nesse caso, as variáveis dependentes podem ser interpretadas como o exemplo, as informações de relações de pacientes que podem ter ou não diabetes, e o estado do paciente pode ser interpretado como o que se deseja aprender. Este processo é entendido como *aprendizado por exemplo*, HASTIE *et al.* (2009). O aprendizado supervisionado pode aparecer em casos de regressão linear, regressão logística, ou até mesmo em métodos mais modernos, como GAM, boosting e máquina de vetores de suporte, JAMES *et al.* (2013).

5.3.2 Aprendizado não supervisionado

Por outro lado, o aprendizado não supervisionado aparece em situações mais desafiadores, pois não há um exemplo para explicar aquilo que se pretende explicar. Este processo é conhecido como *aprendizado sem exemplo*, HASTIE *et al.* (2009). Dessa forma, no aprendizado não supervisionado, tem-se uma amostra com N observações $(x_1, ..., x_N)$ de um vetor aleatório X com densidade conjunta $f(x)$ em que o objetivo é inferir propriedades da densidade sem ajuda de exemplos para cada observação. Assim, como há uma falta de uma variável resposta y_i para supervisionar a análise, pode-se procurar entender a relação entre as variáveis ou as observações, JAMES *et al.* (2013). Por exemplo, uma das técnicas mais aplicadas em problemas que envolvem o aprendizado supervisionado é a análise de cluster, em que o objetivo é determinar, com base em $x_1, ..., x_n$, se as observações são caracterizadas em grupos distintos. Esse é um dos métodos que poderiam ser aplicados, por exemplo, na análise de crédito de clientes de um cartão de crédito, tornando possível analisar o seu perfil e classificá-lo em diferentes grupos para recomendar produtos especificos adequados ao seu perfil.

5.4 Reamostragem para avaliação de performance

5.5 Métricas de avaliação

5.6 Tunagem de hiperparâmetros

Na aprendizagem de máquina, uma das etapas fundamentais é a tunagem dos hiperparâmetros dos algoritmos de aprendizagem. Essa etapa consiste em encontrar a melhor combinação de hiperparâmetros e, consequentemente, resultando em uma configuração algoritmo que proporciona melhor performance e capacidade de generalização. No entanto, essa configuração não é trivial.

Os algoritmos de otimização de hiperparâmetros procuram pelo melhor ajuste de hiperparâmetros $\lambda \in \tilde{\Lambda}$ para um algoritmo de aprendizagem I_λ. O espaço de procura

$\tilde{\Lambda} \in \Lambda$ contém todas as possíveis configurações de hiperparâmetros consideradas para otimização. Dessa forma, temos que:

$$\tilde{\Lambda} = \tilde{\Lambda}_1 \cup \tilde{\Lambda}_2 \cup ... \cup \tilde{\Lambda}_l \tag{5.1}$$

em que l são todas as possíveis configurações de hiperparâmetros. Além disso, $\tilde{\Lambda}_i \ \ (i = 1, 2, ..., l)$ representam o subconjunto de limites dos domínios do i-ésimo hiperparâmetro Λ_i, podendo ser contínuo, discreto ou categórico (BISCHL et al., 2023).

A busca pela melhor combinação de hiperparâmetros λ é feito de forma iterativa, utilizando métodos de reamostragem para dividir o conjunto de dados entre teste e treinamento. Portanto, para validar e encontrar os hiperparâmetos, o algoritmo de aprendizagem é validado através da divisão do conjunto de dados. Assim, tem-se a seguinte estratégia de separação do conjunto de dados:

$$\mathcal{J} = \left(\left(J_{treino,1},\ J_{teste,1}\right), ..., \left(J_{treino,B},\ J_{test,B}\right)\right) \tag{5.2}$$

onde $J_{treino,i},\ J_{teste,i}$ representam os vetores de índices $(i = 1,\ 2, ...,\ B)$ para os conjuntos de treino e teste, respectivamente, e B representa o número total de divisões.

A motivação para fazer a divisão do conjunto de dados entre treino e teste é ajustar o algoritmo em dados reais e avaliar a sua performance em dados ainda não vistos, processo representado pelo banco de treino e teste, respectivamente. Para fazer a avaliação dessa performance é necessário estimar uma métrica de performance em cada divisão. Assim, para um dado algoritmo I_λ é calculado uma métrica de performance ρ para cada $J_{teste,i}$ após o ajuste do algoritmo no $J_{treino,i}$. Por fim, pode ser calculado uma média amostral da métrica de cada uma das divisões. Dessa forma, o problema de otimizar os hiperparâmetros pode ser definida da seguinte forma:

$$\lambda^* \in \underset{\lambda \in \tilde{\Lambda}}{\operatorname{argmin}}\ c\left(\lambda\right)$$

onde λ^* denota a melhor combinação possível de hiperparâmetros, $c\left(\lambda\right)$ representa a generalização do erro, podendo ser representado também por $\widehat{GE}\left(I,\ \mathcal{J},\ \rho,\ \lambda\right)$, quando $I,\ \mathcal{J},\ \rho$ são fixados e I denota um algoritmo de aprendizagem de máquina. O erro de generalização é estimado e otimizado a fim de evitar um ajuste excessivo (overfitting), podendo prejudicar quando o algoritmo fizer estimativas em dados ainda não vistos. Assim, o processo pode ser visualizado pela figura Figura 5.1:

5.6.1 Otimização Bayesiana

Existem diversas técnicas para otimização de hiperparâmetros utilizadas em aprendizagem de máquina. Uma das técnicas mais comuns é o GridSearch. BISCHL

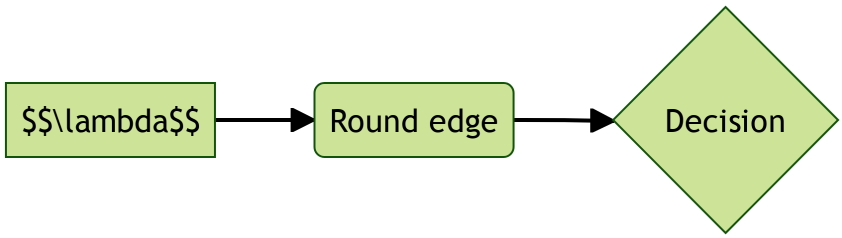

Figura 5.1: Processo de otimização de hiperparâmetros

et al. (2023) definem GridSearch como um processo que divide o intervalo contínuo de valores possíveis de cada hiperparâmetro em um conjunto de valores específicos e avalia exaustivamente o algoritmo para todas as combinações possíveis. No entanto, como todas as combinações possíveis aumentam exponencialmente com a quantidade necessária para avaliação do algoritmo, o GridSearh tem um custo computacional bastante elevado. Assim, existem algoritmos de otimização mais sofisticados que entregam melhores performances, como a otimização bayesiana, que foi utilizada neste trabalho.

A otimização bayesiana não se refere a um tipo específico de algoritmo de otimização, mas sim a uma filosofia de otimização baseada em inferência bayesiana, a qual contém uma extensa família de algoritmos de otimização (GARNETT, 2023). Não obstante, a otimização bayesiana tem obtido benchmarks melhores que outros algoritmos em inúmeros problemas complexos de otimização de hiperparâmetros (SNOEK; LAROCHELLE; ADAMS, 2012).

Diferente de outros algoritmos de otimização de hiperparâmetros, a otimização bayesiana determina as futuras tentativas de avaliação com base em resultados obtidos previamente (YANG; SHAMI, 2020). Para a definição dos pontos futuros, é utilizada uma função probabilística $P(\rho|\lambda)$ (BERGSTRA; YAMINS; COX, 2013). Assim, após o ajuste da função probabilística, tem-se como resultado para cada λ uma estimativa da performance $\hat{c}(\lambda)$ e da predição da incerteza $\hat{\sigma}(\lambda)$, além de obter também a distribuição preditiva da função probabilística. Com a distribuição obtida, uma função de aquisição determina o trade-off entre exploitation e exploration[1]. Dessa forma, os algoritmos de otimização bayesiana são definidos segundo a lei $\lambda \rightarrow c(\lambda)$ e procuram um equilíbrio entre o processo de exploitation-exploration para detectar as regiões ótimas mais prováveis e não perder melhores configurações em áreas ainda não exploradas.

[1]exploitation pode ser entendido como procurar próximo a boas observações e

5.6.2 Tree-Structured Parzen Estimator

Existem diversas funções probabilísticas para uso na otimização bayesiana, algumas delas é o Processo Gaussiano, Random Forest ou Tree-Structured Parzen Estimator (TPE). Nesse trabalho, foi utilizado o Tree-Structured Parzen Estimator, utilizando a biblioteca Optuna (AKIBA *et al.*, 2019) para sua aplicação.

O TPE define duas funções, $l(x)$ e $g(x)$, que são usadas para modelar a distribuição das variáveis do domínio (YANG; SHAMI, 2020). Utilizando as duas densidades, o TPE procura modelar a probabilidade de se observar um hiperparâmetro x dado uma métrica de performance ρ. Dessa forma, tem-se a seguinte definição:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{5.3}$$

em que $l(x)$ é definido como a densidade em que a função perda é menor que um limiar y^* e $g(x)$ representa é a densidade em que a função perda[2] tem valores acima do limiar y^* (BERGSTRA *et al.*, 2011). O limite y^* é escolhido através de um hiperparâmetro γ, onde γ representa o percentil dos valores observados de y, de modo que $p(y < y^*) = \gamma$.

Por padrão, o tree-structured parzen estimator tem como função de aquisição o Expected Improvement (EI), que pode ser otimizado para o TPE da seguinte forma:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, p(y|x)\, dy \tag{5.4}$$

Ainda, para encontrar a probabilidade marginal de x, temos a seguinte integral $p(x) = \int_{\mathbb{R}} p(x|y)\, p(y)\, dy$. Particionando o domínio de y, chega-se em:

$$p(x) = \int_{-\infty}^{y^*} p(x|y)\, p(y)\, dy + \int_{y^*}^{\infty} p(x|y)\, p(y)\, dy = \gamma l(x) + (1-\gamma)\, g(x)$$

Assim, utilizando o Teorema de Bayes e fazendo as substituições na integral da Equação 5.4:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, \frac{p(x|y)\, p(y)}{p(x)} dy = \left(\gamma y^* - \int_{-\infty}^{y^*} p(y)\, dy\right) \left(\gamma + (1-\gamma)\, \frac{g(x)}{l(x)}\right)^{-1}$$

em que a segunda expressão do produto mostra que para maximizar o Expected Improvement é necessário pontos de x com maior probabilidade em $l(x)$ e com baixa probabilidade em $g(x)$. Não obstante, no TPE, maximizar o EI é equivalente a

[2] definição de função perda

maximizar a razão entre as duas distribuições $r(x) = \frac{l(x)}{g(x)}$ (COWEN-RIVERS *et al.*, 2022).

6 Capítulo 4

6.1 Resultados

7 Conclusão

8 Referências

AKIBA, T. *et al.* Optuna: A Next-generation Hyperparameter Optimization Framework. [S.l.]: [s.n.], 2019.

BERGSTRA, J. *et al.* Algorithms for hyper-parameter optimization. **Advances in neural information processing systems**, 2011. v. 24.

_____; YAMINS, D.; COX, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. [S.l.]: PMLR, 2013. p. 115–123.

BISCHL, B. *et al.* Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, 2023. v. 13, n. 2, p. e1484.

BREIMAN, L. Bagging predictors. **Machine learning**, 1996. v. 24, p. 123–140.

COWEN-RIVERS, A. I. *et al.* Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. **Journal of Artificial Intelligence Research**, 2022. v. 74, p. 1269–1349.

FRIEDMAN, J. H. Stochastic gradient boosting. **Computational statistics & data analysis**, 2002. v. 38, n. 4, p. 367–378.

GARNETT, R. **Bayesian optimization**. [S.l.]: Cambridge University Press, 2023.

HASTIE, T. *et al.* **The elements of statistical learning: data mining, inference, and prediction**. [S.l.]: Springer, 2009. V. 2.

IZBICKI, R.; SANTOS, T. M. Dos. **Aprendizado de máquina: uma abordagem estatística**. [S.l.]: Rafael Izbicki, 2020.

JAMES, G. *et al.* **An introduction to statistical learning**. [S.l.]: Springer, 2013. V. 112.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. **Advances in neural information processing systems**, 2012. v. 25.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, 2020. v. 415, p. 295–316.