Escrever título (escolher no final)

Universidade Federal da Paraíba - CCEN

Gabriel de Jesus Pereira

30 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 podem ser utilizadas tanto para regressão quanto para classificação. Elas servem de base para os modelos baseados em árvores empregados neste trabalho, focando particularmente nas árvores de regressão[2]. O processo de construção de uma árvore se baseia no particionamento recursivo do espaço dos preditores, onde cada particionamento é chamado de nó e o resultado final é chamado de folha ou nó terminal. Em cada nó, é definida uma condição e, caso essa condição seja satisfeita, o resultado será uma das folhas desse nó. Caso contrário, o processo segue para o próximo nó e verifica a próxima condição, podendo gerar uma folha ou outro nó. Veja um exemplo na Figura 4.1.

O espaço dos preditores é dividido em J regiões distintas e disjuntas denotadas por $R_1, R_2, \ldots, R_J$. Essas regiões são construídas em formato de caixa de forma a minimizar a soma dos quadrados dos resíduos. Dessa forma, pode-se modelar a variável resposta como uma constante c_j em cada região R_j

[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.

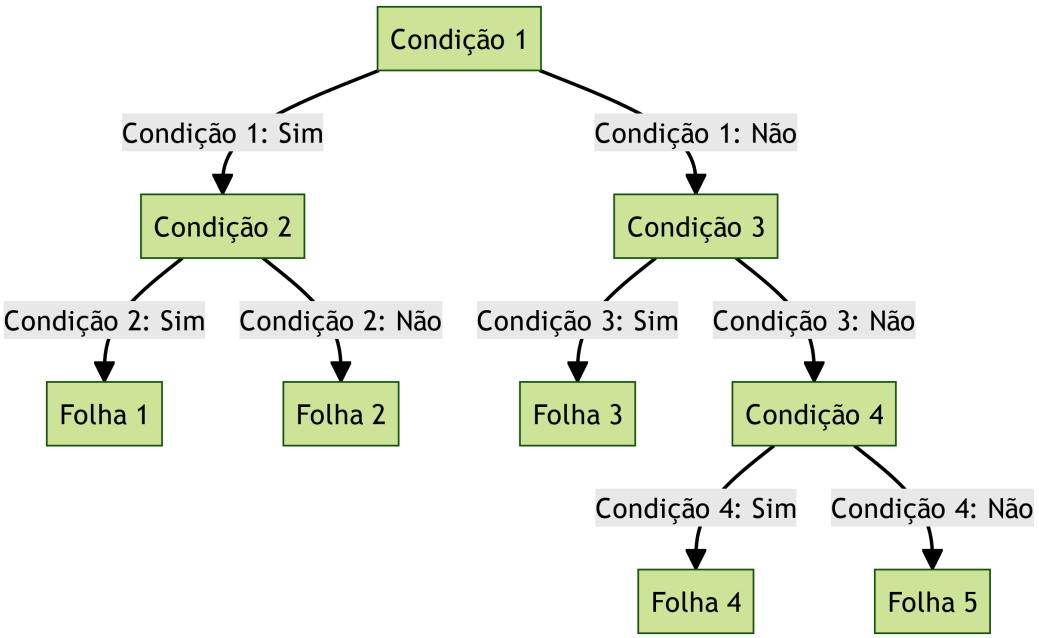


Figura 4.1: Exemplo de estrutura de árvore de regressão. A árvore tem cinco folhas e quatro nós internos.

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

O estimador para a constante c_j é encontrado pelo método de mínimos quadrados. Assim, deve-se minimizar $\sum_{x_i \in R_j} \left[y_i - f(x_i)\right]^2$. No entanto, perceba que $f(x_i)$ está sendo avaliado somente em um ponto específico x_i, o que reduzirá $f(x_i)$ para uma constante c_j. É fácil de se chegar ao resultado se for observada a definição da função indicadora $I\left(x \in R_j\right)$

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

Como as regiões são disjuntas, x_i não pode estar simultaneamente em duas regiões. Assim, para um ponto específico x_i, apenas um dos casos da função indicadora será diferente de 0. Portanto, $f(x_i) = c_j$. Agora, derivando $\sum_{x_i \in R_j} \left(yi - c_j\right)^2$ em relação a c_j

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

e igualando Equação 4.1 a 0, tem-se a seguinte igualdade

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

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 simplesmente a média dos y_i na região R_j:

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

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 um hiperparâmetro para regular a complexidade do modelo, pois uma árvore muito grande pode causar sobreajuste aos dados de treinamento, capturando não apenas os padrões relevantes, mas também o ruído. Como resultado, o modelo pode apresentar bom desempenho nos dados de treinamento, mas falhar ao lidar com novos dados devido à sua incapacidade de generalização. Por outro lado, uma árvore muito pequena pode não captar padrões,

relações e estruturas importantes presentes nos dados. Dessa forma, a estratégia adotada para selecionar o tamanho da árvore consiste em crescer uma grande árvore T_0, interrompendo o processo de divisão apenas ao atingir um tamanho mínimo de nós. Posteriormente, a árvore T_0 é podada utilizando o critério de custo complexidade, que será definido a seguir.

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

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

uma medida de impureza do nó pelo erro quadrático médio. Assim, define-se o critério de custo complexidade

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

onde $|T|$ denota a quantidade total de folhas, e $\alpha \geq 0$ é um hiperparâmetro que equilibra o tamanho da árvore e a adequação aos dados. A ideia é encontrar, para cada α, a árvore $T_\alpha \subset T_0$ que minimiza $C_\alpha(T)$. Valores grandes de α resultam em árvores menores, enquanto valores menores resultam em árvores maiores, e $\alpha = 0$ resulta na própria árvore T_0. A busca por T_α envolve colapsar sucessivamente o nó interno que provoca o menor aumento em $\sum_j N_j Q_j(T)$, continuando o processo até produzir uma árvore com um único nó. Esse processo gera uma sequência de subárvores, na qual existe uma única subárvore menor que, para cada α, minimiza $C_\alpha(T)$.

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

No caso de uma árvore de decisão para classificação, a principal diferença está no critério de divisão dos nós e na poda da árvore. Para a classificação, a previsão em um nó j, correspondente a uma região R_j com N_j observações, será simplesmente a classe majoritária. Assim, tem-se

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{x_i \in R_j} I(y_i = k)$$

como a proporção de observações da classe k no nó j. Dessa forma, as observações no nó j são classificadas na classe $k(j) = \arg\max_k \hat{p}_{jk}$, que é a moda no nó j.

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).

Para a divisão dos nós no caso da regressão, foi utilizado o erro quadrático médio como medida de impureza. Para a classificação, algumas medidas comuns para $Q_j(T)$ são o erro de classificação, o índice de Gini ou a entropia cruzada.

4.2 Métodos Ensemble

As árvores de decisão são conhecidas por sua alta interpretabilidade, mas geralmente apresentam um desempenho 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 estimadores base, um processo conhecido como aprendizado em conjunto (Ensemble Learning). De acordo com HASTIE *et al.* (2009), o aprendizado em conjunto 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 aprendizado em conjunto utilizados neste trabalho.

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 criadas por meio de amostras bootstrap do conjunto de treinamento, utilizadas como novos conjuntos de treinamento. O Bagging pode ser empregado para melhorar a estabilidade e a precisão de modelos ou algoritmos de aprendizado de máquina, além de reduzir a variância e evitar o sobreajuste. Por exemplo, o Bagging pode ser utilizado para melhorar o desempenho da árvore de regressão descrita anteriormente.

BREIMAN (1996) define formalmente o algoritmo de Bagging, que utiliza um conjunto de treinamento $\mathcal{L}$. A partir desse conjunto, são geradas amostras bootstrap $\mathcal{L}^{(B)}$ com B réplicas, formando uma coleção de modelos $\{f(x, \mathcal{L}^{(B)})\}$, onde f representa um modelo estatístico ou algoritmo treinado nas amostras bootstrap para prever ou classificar uma variável dependente y com base em variáveis independentes $\mathbf{x}$. Se a variável dependente y for numérica, a predição é obtida pela média das previsões dos modelos:

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

onde f_B representa a predição agregada. No caso em que y prediz uma classe, utiliza-se a votação majoritária. Ou seja, se estivermos classificando em classes $j \in 1, \ldots, J$, então $N_j = \#\{B; f(x, \mathcal{L}^{(b)}) = j\}$ representa o número de vezes que a classe j foi predita pelos estimadores. Assim,

$$f_B(x) = \arg\max_j N_j$$

isto é, o j para o qual N_j é máximo

Embora a técnica de Bagging possa melhorar o desempenho de uma árvore de regressão ou de classificação, isso geralmente vem ao custo de menor interpretabilidade. Quando o Bagging é aplicado a uma árvore de regressão, construímos B árvores de regressão usando B réplicas de amostras bootstrap e tomamos a média das predições resultantes (JAMES *et al.*, 2013). Nesse processo, as árvores de regressão crescem até seu máximo, sem passar pelo processo de poda, resultando em cada árvore individual com alta variância e baixo viés. No entanto, ao agregar as predições das B árvores, a variância é reduzida.

Para mitigar a falta de interpretabilidade do método Bagging aplicado a árvores de regressão, pode-se usar a medida de impureza baseada no erro quadrático médio,

definida anteriormente, como uma métrica de importância das variáveis independentes. Um valor elevado na redução total média do erro quadrático médio, calculado com base nas divisões realizadas por um determinado preditor em todas as B árvores, indica que o preditor é importante.

As árvores construídas pelo algoritmo de árvore de decisão se beneficiam da 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 previsões das B árvores será $\frac{1}{B}\sigma^2$. No entanto, se as árvores forem apenas identicamente distribuídas, mas não necessariamente independentes, e apresentarem uma correlação positiva ρ, a esperança da média das B árvores será a mesma que a esperança de uma árvore individual. Portanto, o viés do agregado das árvores é o mesmo das árvores individuais, e a melhoria é alcançada apenas pela redução da variância. A variância da média das previsões será dada por:

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

Isso significa que, à medida que o número de árvores B aumenta, o segundo termo da soma se torna menos significativo. Portanto, os benefícios da agregação proporcionados pelo algoritmo de Bagging são limitados pela correlação entre as árvores (HASTIE *et al.*, 2009). Mesmo com o aumento do número de árvores no Bagging, a correlação entre elas impede que as previsões individuais sejam completamente independentes, resultando em menor diminuição da variância da média das previsões do que seria esperado se as árvores fossem totalmente independentes. Uma maneira de melhorar o algoritmo de Bagging é por meio do Random Forest, que será descrito a seguir.

4.2.2 Random Forest

O algoritmo Random Forest é uma técnica derivada do método de Bagging, mas com modificações específicas na construção das árvores. O objetivo é melhorar a redução da variância ao diminuir a correlação entre as árvores, sem aumentar significativamente a variabilidade. Isso é alcançado durante o processo de crescimento das árvores por meio da seleção aleatória de variáveis independentes.

No algoritmo Random Forest, ao construir uma árvore a partir de amostras bootstrap, antes de cada divisão, selecionam-se aleatoriamente $m \leq p$ das p variáveis independentes como candidatas para a divisão (com $m = p$ no caso do Bagging). Apenas uma dessas m variáveis é usada para realizar a divisão, com base em critérios como a minimização da impureza. Diferentemente do Bagging, que tende a gerar árvores de decisão semelhantes e, portanto, previsões altamente correlacionadas, o Random Forest

visa minimizar esse problema ao proporcionar oportunidades para que outros preditores sejam considerados. Assim, em média, $(p-m)/p$ das divisões nem sequer considerarão o preditor mais forte, permitindo que outros preditores também tenham a chance de serem usados (JAMES *et al.*, 2013). Esse processo de redução da correlação entre as árvores resulta em uma média das árvores menos variável e, consequentemente, mais confiável.

A quantidade de variáveis independentes m selecionadas aleatoriamente é um hiperparâmetro que pode ser estimado por meio de validação cruzada. Valores comuns para m são $m = \sqrt{p}$ com tamanho mínimo do nó igual a um para classificação, e $m = p/3$ com tamanho mínimo do nó igual a cinco para regressão (HASTIE *et al.*, 2009). Quando o número de variáveis é grande, mas poucas são realmente relevantes, o algoritmo Random Forest pode ter um desempenho inferior com valores pequenos de m, pois isso reduz as chances de selecionar as variáveis mais importantes. No entanto, usar um valor pequeno de m pode ser vantajoso quando há muitos preditores correlacionados. Além disso, assim como no Bagging, a Random Forest não sofre de sobreajuste com o aumento da quantidade de árvores B. Portanto, é suficiente usar um B grande o bastante para que a taxa de erro se estabilize (JAMES *et al.*, 2013).

4.2.3 Boosting Trees

O Boosting, assim como o Bagging, é um método destinado a melhorar o desempenho de modelos ou algoritmos. No entanto, neste trabalho, o Boosting foi aplicado apenas às árvores de regressão. Portanto, a explicação do Boosting será restrito ao caso de Boosting Trees (Algoritmo 4.3).

No algoritmo de Bagging, cada árvore é construída e ajustada utilizando amostras bootstrap, e ao final, um estimador agregado φ_B é formado a partir das B árvores. O Boosting Trees funciona de maneira semelhante, mas sem o uso de amostras bootstrap. A ideia principal é corrigir os erros das árvores anteriores, ajustando as novas árvores aos resíduos das anteriores, visando melhorar suas previsões. Assim, as árvores são construídas de forma sequencial, incorporando as informações das árvores anteriores.

No caso da regressão, o Boosting combina um grande número de árvores de decisão $\hat{f}^1, \ldots, \hat{f}^B$. A primeira árvore é construída utilizando o conjunto de dados original, e seus resíduos são calculados. Com a primeira árvore ajustada, a segunda árvore é ajustada aos da árvore anterior resíduos e, em seguida, é adicionada ao estimador para atualizar os resíduos. Dessa forma, os resíduos servem como informação crucial para construir novas árvores e corrigir os erros das árvores anteriores. Como cada nova árvore depende das árvores já construídas, árvores menores são suficientes (JAMES *et al.*, 2013).

O processo de aprendizado no método de Boosting é lenta, o que acaba gerando melhores resultados. Esse processo de aprendizado pode ser controlado por um hiperparâmetro λ chamado de shrinkage, ou taxa de aprendizado, permitindo que mais árvores, com formas diferentes, corrijam os erros das 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.

4.2.4 Stacked generalization

A Stacked Generalization, ou Stacking, é um método de ensemble que consiste em treinar um modelo gerado a partir da combinação da predição de vários outros modelos, visando melhorar a precisão das predições. Esse método pode ser aplicado a qualquer modelo estatístico ou algoritmo de aprendizado de máquina. A ideia principal é atribuir pesos às predições, de modo a dar maior importância aos modelos que produzem melhores resultados, ao mesmo tempo em que se evita atribuir altos pesos a modelos com alta complexidade.

Matematicamente, o Stacking define predições $\hat{f}_m^{-i}(x)$ em x, utilizando o modelo m, aplicado ao conjunto de treinamento com a i-ésima observação removida (HASTIE *et al.*, 2009). Assim, os peso são estimados de forma a minimizar o erro de predição combinado, dado pela seguinte expressão:

$$\hat{w}^{st} = \arg\min_{w} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} w_m f_m^{-i}(x_i) \right]^2$$

A previsão final dos modelos empilhados é $\sum_m \hat{w}_m^{st} \hat{f}_m(x)$. Assim, em vez de escolher um único modelo, o método de Stacking combina os modelos utilizando pesos estimados, o que melhora a performance preditiva, mas pode comprometer a interpretabilidade.

4.2.5 Gradient Boosting

O algoritmo de Gradient Boosting é semelhante ao de Boosting, mas com diferenças mínimas. Ele constrói modelos aditivos ajustando sequencialmente funções bases aos pseudos-resíduos, que correspondem aos gradientes da função perda do modelo atual (FRIEDMAN, 2002). Esses gradientes indicam a direção na qual a função perda

diminui. Neste trabalho, foram utilizadas diferentes implementações de Gradient Boosting. No entanto, todas empregam o Gradient Boosting com árvores de regressão, com algumas modificações para a construção das árvores ou para melhorar a eficiência do algoritmo existente. Assim, o algoritmo a ser explicado será o Gradient Tree Boosting (Algoritmo 4.4).

O Gradient Boosting aplicado para árvores de regressão, tem que cada função base é uma árvore de regressão com J_m folhas. Dessa forma, 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 $\mathbf{x}$. 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 r_{im},

$$\hat{b}_{jm}=\frac{1}{N_{jm}}\sum_{x_i\in R_{jm}} r_{im}$$

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 r_{im} de forma a minimizar o erro das árvores anteriores. N_{jm} denota a quantidade de pontos na região R_{jm}. Por fim, o estimador é separadamente atualizado em cada região correspondente e é expresso

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

em que γ_{jm} representa a atualização da constante ótima para cada região, baseado na função perda L, dada a aproximação $f_{m-1}(x)$. O $0<\lambda\leq 1$, 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

4.2.6.1 Light Gradient Boosting

4.2.6.2 Extreme Gradient Boosting

Algoritmo 4.2 Algoritmo de uma Random Forest para regressão ou classificação

1. Para b = 1 até B:

(a) Construa amostras bootstrap $\mathcal{L}^*$ 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 $i = 1, 2, ..., N$, calcule

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

(b) Ajuste uma árvore de regressão aos pseudo-resíduos r_{im}, obtendo regiões terminais $R_{jm},\ j = 1, 2, ..., J$.

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

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

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

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

Algoritmo 4.4: Fonte: HASTIE *et al.* (2009, p. 361).

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 Reamostragem para avaliação de performance

5.4 Métricas de avaliação

5.5 Tunagem de hiperparâmetros

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:

$$\mathcal{J} = \left(\left(J_{treino,1},\ J_{teste,1}\right), ..., \left(J_{treino,B},\ J_{test,B}\right)\right) \tag{5.1}$$

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:

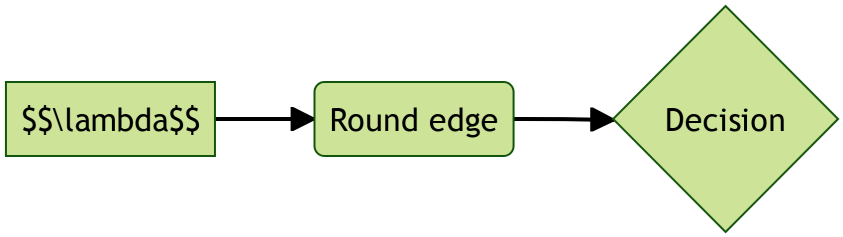


Figura 5.1: Processo de otimização de hiperparâmetros

5.5.1 Otimização Bayesiana

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

5.5.2 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:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{5.2}$$

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:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, p(y|x)\, dy \tag{5.3}$$

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:

[1] exploitation pode ser entendido como procurar próximo a boas observações e
[2] definição de função perda

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

Assim, utilizando o Teorema de Bayes e fazendo as substituições na integral da Equação 5.3:

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

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

6 Resultados

6.1 Análise exploratória de dados

A análise exploratória dos dados foi realizada após a divisão entre os conjuntos de treinamento e teste. Essa abordagem foi adotada para evitar o sobreajuste do modelo e garantir que o algoritmo não aprenda com informações indisponíveis no conjunto de teste. Assim, a descritiva dos dados foi realizada utilizando o conjunto de treinamento.

A primeira etapa da análise exploratória de dados foi identificar os dados faltantes e determinar a melhor forma de tratá-los. A Figura 6.1 mostra a porcentagem de observações ausentes em cada variável. As variáveis com a maior quantidade de dados ausentes são o valor do condomínio e o IPTU, pois essas informações são as menos preenchidas no site de onde os dados foram coletados. A terceira variável, com quase 20% de observações ausentes, é a quantidade de vagas de estacionamento. As variáveis com mais de 20% de observações ausentes foram removidas da base de dados, pois, com essa quantidade de valores faltantes, nem mesmo métodos de imputação proporcionariam um tratamento adequado. Dessa forma, apenas as variáveis de valor do condomínio e IPTU foram removidas, enquanto as demais com valores ausentes foram tratadas por meio de imputação.

Uma das dificuldades que podem surgir durante a modelagem é o desbalanceamento das classes, ou seja, a diferença na quantidade de cada tipo de imóvel. O tipo de imóvel mais predominante no conjunto de dados são os apartamentos, que representam 81,36% do total. Em seguida, vêm as casas, com 8,91%, e os flats, com 5,72%. Por fim, as casas comerciais são as menos representadas, com apenas 15 ocorrências. Esse desbalanceamento claro entre as classes pode dificultar o desempenho do modelo, especialmente na previsão de categorias menos frequentes, como as casas comerciais, onde o modelo pode ter dificuldade em obter bons resultados.

A distribuição das variáveis foram análisadas em termos do tipo do imóvel a partir de um gráfico de violino. Pela Figura 6.2, é fácil perceber que a maioria das distribuições possuem assimetria negativa. Os apartamentos possuem caudas longas à direita, indicando presença de valores extremamente altos e que indicam que talvez seja necessário a aplicação de alguma transformação para a estabilização da variância.

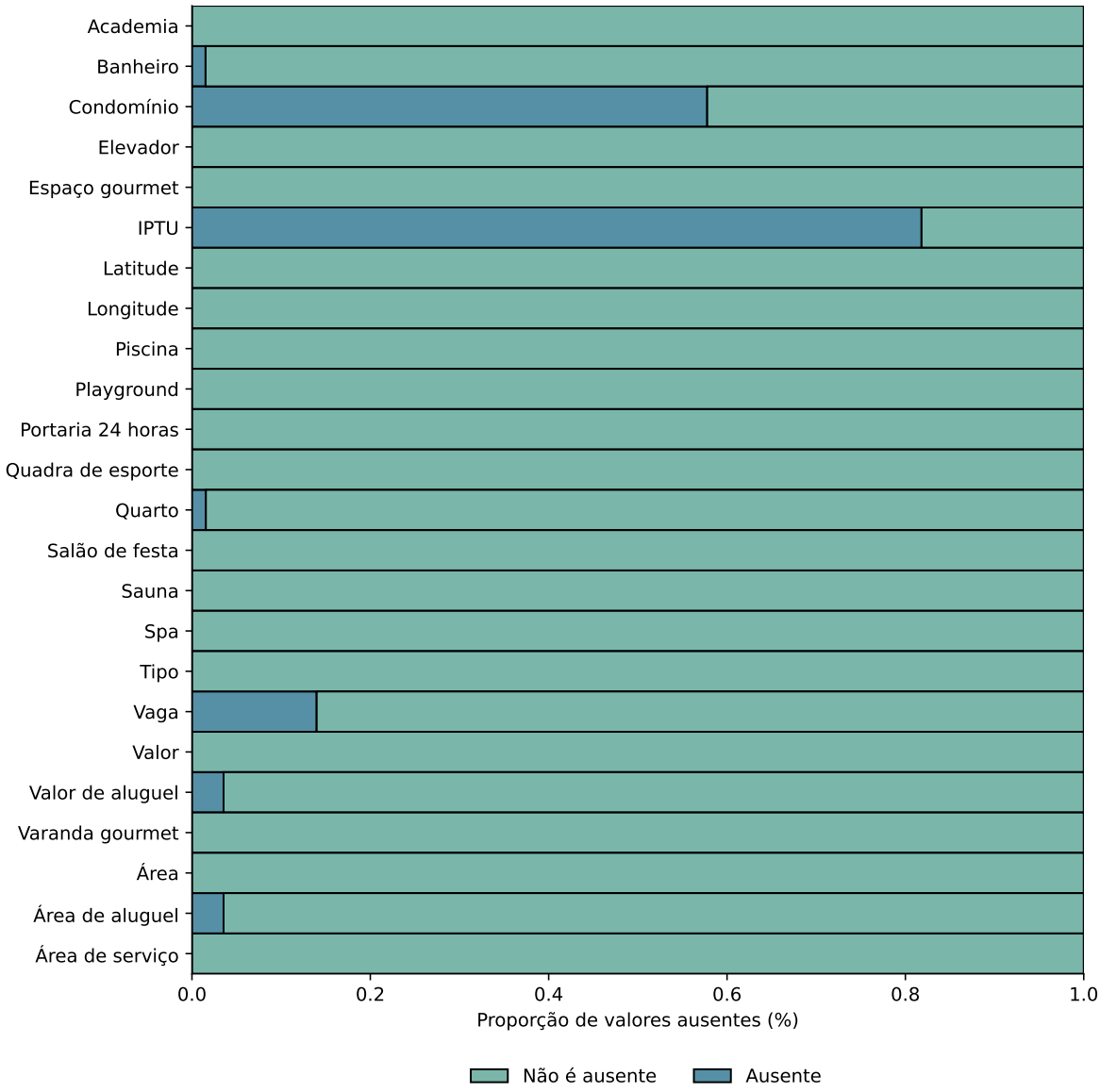


Figura 6.1: Quantidade de valores ausentes por variáveis

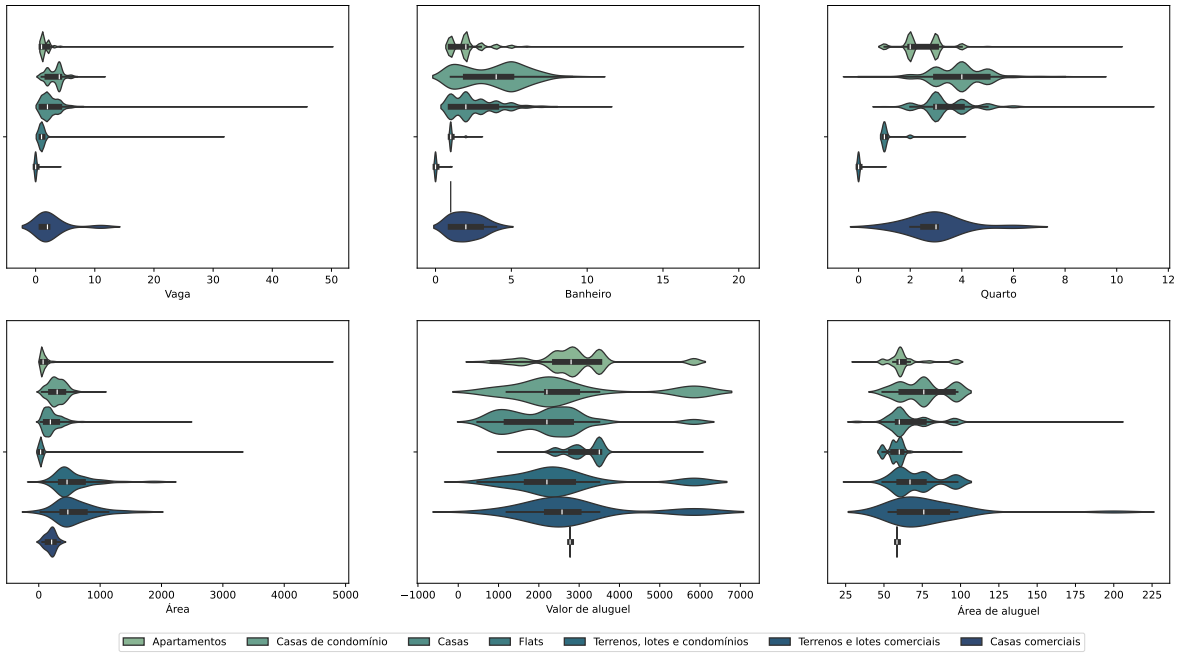
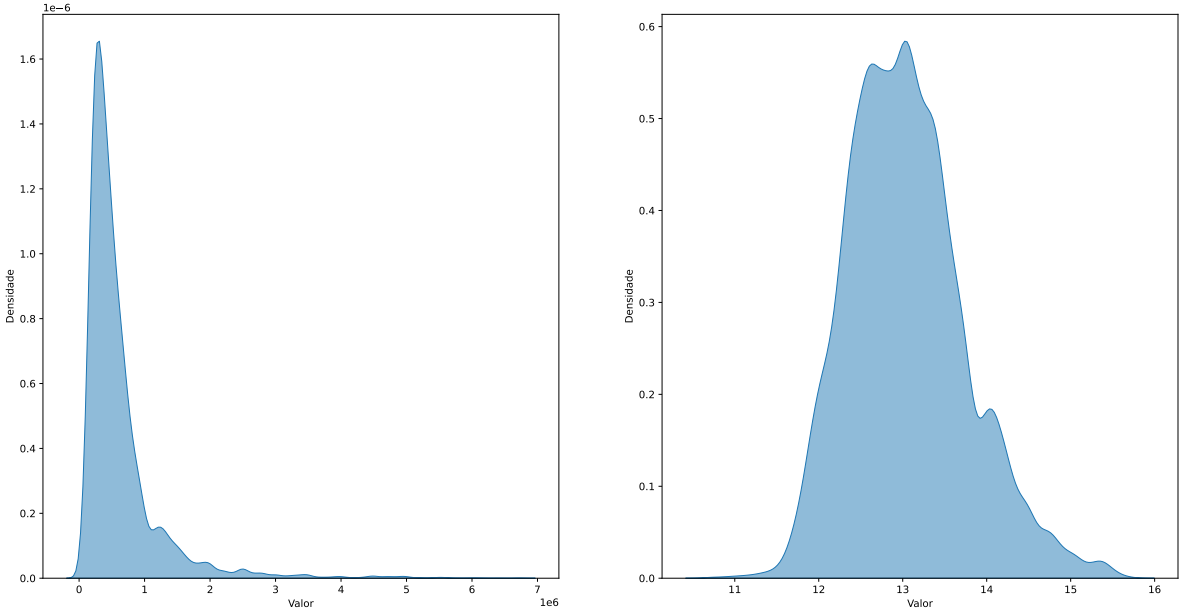
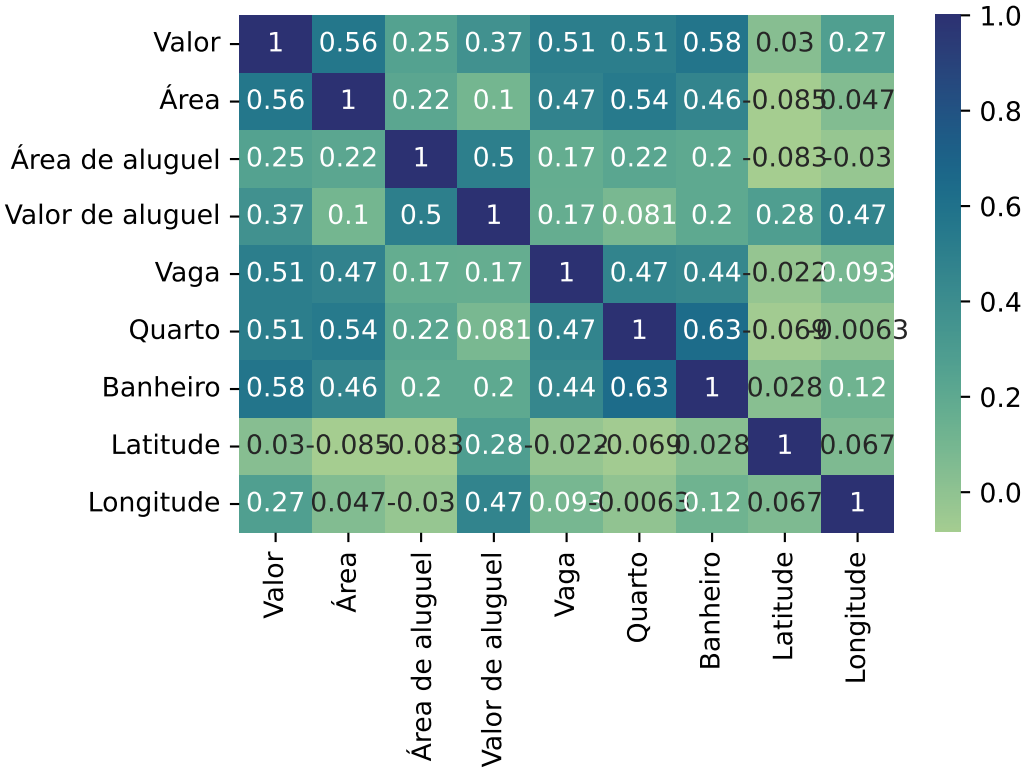


Figura 6.2: Distribuição das variáveis numéricas.




Valor
Área
Área de aluguel
Valor de aluguel
Vaga
Quarto
Banheiro
Latitude
Longitude
1 0.56 0.25 0.37 0.51 0.51 0.58 0.03 0.27
0.56 1 0.22 0.1 0.47 0.54 0.46 -0.085 0.047
0.25 0.22 1 0.5 0.17 0.22 0.2 -0.083 -0.03
0.37 0.1 0.5 1 0.17 0.081 0.2 0.28 0.47
0.51 0.47 0.17 0.17 1 0.47 0.44 -0.022 0.093
0.51 0.54 0.22 0.081 0.47 1 0.63 -0.069 -0.0063
0.58 0.46 0.2 0.2 0.44 0.63 1 0.028 0.12
0.03 -0.085 -0.083 0.28 -0.022 -0.069 0.028 1 0.067
0.27 0.047 -0.03 0.47 0.093 -0.0063 0.12 0.067 1
1.0
0.8
0.6
0.4
0.2
0.0

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.

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.