COC361 - Inteligência Computacional

Relatório Técnico - Grupo 1 (PCQM4Mv2)

Universidade Federal
do Rio de Janeiro

- Henrique Chaves - DRE 119025571
  - henriquechaves@poli.ufrj.br
- Pedro Boechat - DRE 119065050
  - pedroboechat@poli.ufrj.br

Rio de Janeiro - RJ
13 de Março de 2022 (2021.2)

Conteúdo

1 Introdução

A Teoria do Funcional da Densidade (do inglês, *Density Functional Theory*, DFT) é uma teoria para calcular a estrutura eletrônica de átomos e moléculas. Seu objetivo é analisar quantitativamente propriedades moleculares a partir das leis fundamentais da mecânica quântica [1]. Uma dessas propriedades é o *HOMO-LUMO gap*, que corresponde a diferença energética medida entre o orbital molecular ocupado mais alto (HOMO) e o orbital molecular não ocupado mais baixo (LUMO) e pode ser usado para prever, por exemplo, a estabilidade de uma molécula. Portanto, para problemas que envolvem descoberta de novos materiais ou drogas, essa propriedade se mostra fundamental para julgar a viabilidade de síntese de uma molécula, a partir da estimação da reatividade da mesma.

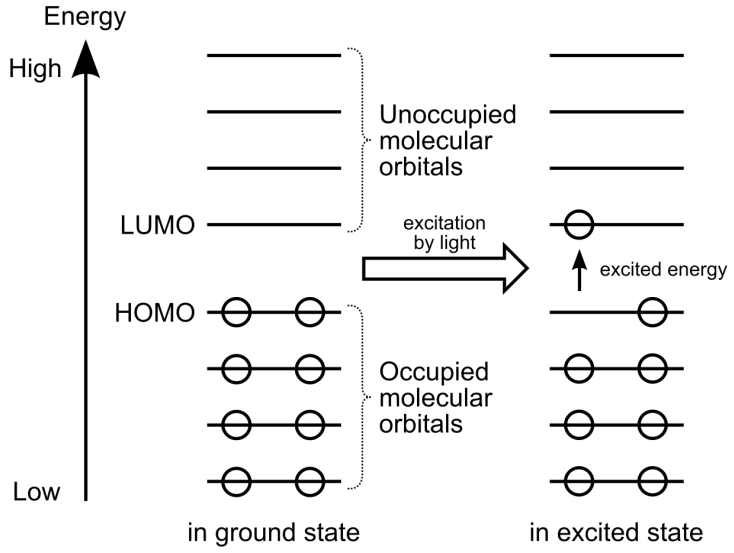

Figura 1: Diagrama do HOMO e do LUMO de uma molécula [2].

O cálculo do *HOMO-LUMO gap* pode ser computado por meio de simulações utilizando *softwares* de quimioinformática, como o `Quantum Espresso` ou o `VASP`. No entanto, essas simulações são demoradas e exigem recursos computacionais elevados, mesmo para moléculas pequenas. Uma das alternativas viáveis para as simulações é a elaboração de um modelo de aprendizado de máquina que permita estimar um valor relativamente próximo para esse cálculo, sem necessidade de exatidão. Com esse valor estimado é possível fazer a triagem de moléculas com características buscadas e seguir para uma análise mais profunda delas. Nesse sentido, ao longo desse projeto, vamos treinar e testar a eficiência de diferentes modelos de aprendizado de máquina para a estimativa do valor do *HOMO-LUMO gap*.

2 Dataset e Tecnologia

2.1 Descrição dos dados

Para esse trabalho, escolhemos o dataset `PCQM4Mv2` [3]. Este dataset possui 3746620 linhas e duas colunas:

- `smiles`: Contém códigos SMILES (*Simplified Molecular Input Line Entry Specification*) de moléculas químicas. Os SMILES são uma forma de representar estruturas químicas usando caracteres ASCII [4].
- `homolumogap`: Contém o valor do *HOMO-LUMO gap* para a molécula, calculado por meio da DFT.

Como a variável alvo (`homolumogap`) é um número, o objetivo para esse dataset é de treinar modelos de regressão utilizando aprendizado de máquina.

2.2 Apresentação da tecnologia

Definimos a utilização da linguagem de programação Python para a realização do trabalho. Por conta de suas bibliotecas apropriadas para ciência de dados, como por exemplo `scikit-learn` para pré-processamento de dados, treinamento de modelos de aprendizado de máquina e cálculo de métricas, e `matplotlib` para visualização de dados, essa linguagem se mostrou fundamental.

Para o treinamento da maioria dos modelos, fizemos o uso do mencionado `scikit-learn`, com exceção das Redes Neurais Artificiais, onde a biblioteca `scikeras` foi utilizada. Com essa biblioteca, foi possível criar topologias de Redes Neurais Artificiais em `tensorflow`, que é uma biblioteca para aprendizado de máquina e principalmente para aprendizado profundo, porém se aproveitando de funcionalidades especiais do `scikit-learn`, como por exemplo, treinamento automático de modelos a partir da definição de uma grade de parâmetros a serem testados.

A plataforma `Jupyter Notebook` foi utilizada extensivamente para a realização do trabalho, pois com ela é possível realizar experimentos e visualizações de forma interativa além do que a linguagem de programação oferece.

3 Metodologia

3.1 Expansão do dataset utilizando o RDKit

Utilizamos recursos da biblioteca de quimioinformática `RDKit` para expandir o dataset com mais propriedades das moléculas a partir da coluna `smiles`. Foram utilizados métodos e descritores (padrões e personalizados) para se obter as seguintes variáveis:

- `number_of_atoms`: Número total de átomos na molécula.
- `number_of_atoms_except_H`: Número de átomos na molécula excluindo os átomos de Hidrogênio (H).
- `number_of_bonds`: Número total de ligações químicas na molécula.
- `number_of_heavy_bonds`: Número de ligações químicas "pesadas" na molécula.
- `number_of_conformations`: Número total de possíveis conformações da molécula.
- `number_of_heavy_atoms`: Número de átomos "pesados" na molécula.
- `exact_mol_weight`: Valor exato do peso molecular.
- `average_mol_weight`: Valor médio do peso molecular.
- `heavy_mol_weight`: Valor médio do peso molecular excluindo os átomos de Hidrogênio (H).
- `number_of_radical_electrons`: Número de elétrons desemparelhados na molécula.
- `number_of_valence_electrons`: Número de elétrons emparelhados na molécula.
- `fp_morgan_density_1`: Valor da densidade da molécula de acordo com o Fingerprint 1 de Morgan.
- `fp_morgan_density_2`: Valor da densidade da molécula de acordo com o Fingerprint 2 de Morgan.
- `fp_morgan_density_3`: Valor da densidade da molécula de acordo com o Fingerprint 3 de Morgan.
- `max_absolute_partial_charge`: Valor máximo absoluto da carga parcial da molécula.
- `min_absolute_partial_charge`: Valor mínimo absoluto da carga parcial da molécula.
- `max_partial_charge`: Valor máximo da carga parcial da molécula.
- `min_partial_charge`: Valor mínimo da carga parcial da molécula.
- `number_of_B_atoms`: Número de átomos de Boro (B) na molécula.
- `number_of_C_atoms`: Número de átomos de Carbono (C) na molécula.
- `number_of_N_atoms`: Número de átomos de Nitrogênio (N) na molécula.

- `number_of_O_atoms`: Número de átomos de Oxigênio (O) na molécula.
- `number_of_F_atoms`: Número de átomos de Flúor (F) na molécula.
- `number_of_Si_atoms`: Número de átomos de Silício (Si) na molécula.
- `number_of_P_atoms`: Número de átomos de Fósforo (P) na molécula.
- `number_of_S_atoms`: Número de átomos de Enxofre (S) na molécula.
- `number_of_Cl_atoms`: Número de átomos de Cloro (Cl) na molécula.
- `number_of_Br_atoms`: Número de átomos de Bromo (Br) na molécula.

Com isso, obteve-se inicialmente 28 variáveis independentes (features) a partir da estrutura de cada molécula, que poderiam ser usadas como preditores do valor de *HOMO-LUMO gap*.

3.2 Pré-processamento de dados

Com os dados já expandidos, continuamos o pré-processamento deles com o fim de otimizá-los para o uso nos modelos de aprendizado de máquina. Primeiramente, foram removidas entradas onde existem dados nulos, totalizando 2618 remoções. Esses dados nulos foram encontrados exatamente em 4 variáveis juntas: `max_absolute_partial_charge`, `min_absolute_partial_charge`, `max_partial_charge`, `min_partial_charge`.

Em seguida, foram encontradas duas variáveis com valores idênticos em quase todas as entradas (`number_of_atoms_except_H` e `number_heavy_atoms`), com exceção de 7 registros, e portanto a primeira foi removida para evitar redundância de dados.

Com os dados limpos, a próxima etapa é normalizar a escala para que as variáveis independentes possuam uma mesma amplitude. Esse processo é chamado de normalização e é importante para que variáveis que tenham amplitudes maiores não tenham influência maior no modelo do que variáveis com amplitudes menores só por conta de sua escala. Uma normalização comum de ser aplicada é a de transformação em *Z-Score*, onde para cada registro de cada variável se subtrai a média da variável e divide pelo desvio padrão da mesma. Essa normalização foi aplicada aos dados por conta de sua simplicidade e por não comprimi-los.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

Na equação (1), temos que:

- x é um *data point* de uma variável independente (*feature*).
- μ é a média da *feature* em questão.
- σ é o desvio padrão da *feature* em questão.

3.3 Descrição dos modelos utilizados

3.3.1 Regressão Linear

O modelo de regressão linear busca aproximar uma reta, visando minimizar o erro dado por uma métrica, por exemplo o erro quadrático médio (do inglês, *Mean Squared Error*, MSE), em uma função dada por:

$$\hat{y} = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \tag{2}$$

Nessa equação (2), temos que:

- $\hat{y}$ é o valor predizido da variável alvo.
- n é o número de variáveis independentes (*features*).
- x_i é o valor da $i_{ésima}$ variável independente.

- θ_i é o $i_{ésimo}$ coeficiente do modelo.

Note que na equação (2) há um termo θ_0 que não é multiplicado por nenhuma variável, que é justamente o ponto em que a reta intercepta o eixo y, também conhecido como coeficiente linear.

O MSE é a métrica de otimização mais escolhida para problemas de regressão, pois ele acaba penalizando diferenças maiores, aumentando o erro e fazendo com que o modelo busque reduzir essas diferenças. Portanto, foi a métrica escolhida para ser utilizada por todos os modelos, a fim de padronizar a comparação de resultados.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3}$$

Para a equação (3), que define a fórmula do MSE, temos que:

- m é o número de registros (linhas) no conjunto de dados.
- y_i é o valor real da variável alvo no $i_{ésimo}$ registro.
- $\hat{y}_i$ é o valor predizido da variável alvo no $i_{ésimo}$ registro.

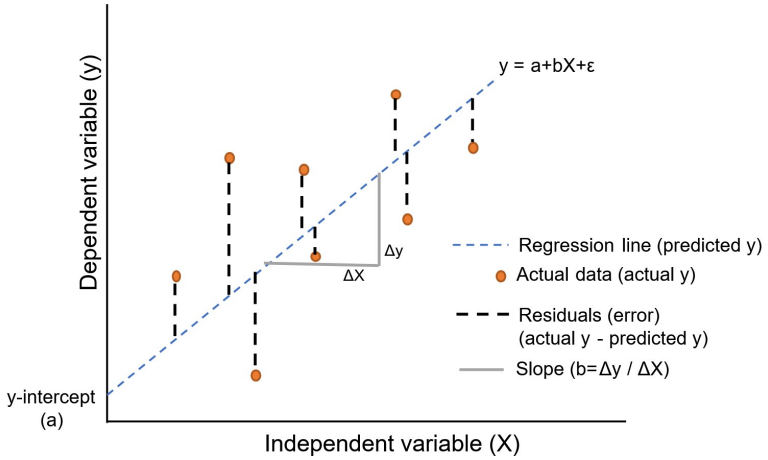

Figura 2: Exemplo de regressão linear [5].

Na figura 2, há um exemplo de gráfico de uma regressão linear em que só há uma variável independente, e com 8 *data points*. A ideia é encontrar os coeficientes que minimizem a função de custo, o MSE.

Há diversas formas de implementar o algoritmo de Regressão Linear e encontrar os melhores coeficientes. A biblioteca *scikit-learn*, por sua vez, calcula a aproximação desses melhores coeficientes baseado na equação (4).

$$\hat{\theta} = X^{+}y \tag{4}$$

Nessa equação (4), temos que:

- X^{+} é a inversa de *Moore-Penrose* da matriz X, que é uma matriz $m \times n$ na qual contém os dados preditores, onde n é o número de variáveis independentes (features) e m é o número de registros (linhas) no conjunto de dados.
- y é o vetor coluna com a variável alvo real de cada registro.

A inversa de *Moore-Penrose* é uma generalização de matriz inversa, podendo ser obtida de várias formas, e muito útil para resolução de problemas em que se precisa encontrar o melhor ajuste para um sistema de equação lineares em que não há solução, como na maioria dos casos de problemas que envolvem Regressão Linear.

3.3.2 Árvores de decisão

O modelo de árvores de decisão funciona por meio de uma estrutura de dados na qual uma árvore tem:

- Nó decisão: Representa um teste de algum atributo (ex. $x_1 < 20$);
- Arestas: Representam o resultado do teste de um nó decisão (ex. sim ou não);
- Nó folha: Representa um valor numérico que será usado como $\hat{y}$.

Após escolhida uma medida de impureza, como o MSE (3), o algoritmo irá escolher como raiz de uma subárvore um atributo que maximize o ganho de informação, dado por:

$$G = I(T) - \sum_{i=1}^{n} \frac{N_i}{N} I(T_i) \tag{5}$$

Nessa equação (5), temos que:

- G é o ganho de informação a ser calculado;
- $I(T)$ é o índice de impureza do conjunto T completo;
- $\sum_{i=1}^{n} \frac{N_i}{N} I(T_i)$ é a média ponderada do índice de impureza para cada subconjunto particionado.

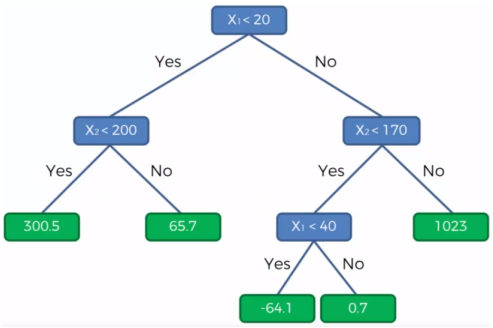

Figura 3: Exemplo de árvore de decisão [6].

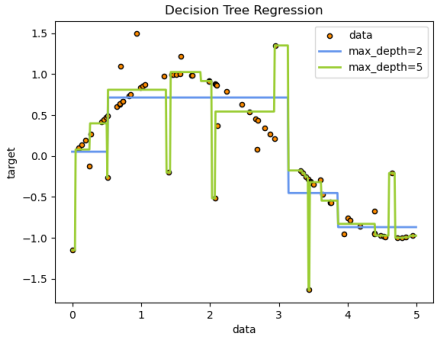

Figura 4: Exemplo de funções geradas utilizando o algoritmo de árvores de decisão [7].

Na figura 3, temos um exemplo de árvore de decisão regressora, como a que será treinada para o nosso dataset. Na figura 4 temos um exemplo de duas funções resultantes de uma árvore de decisão junto do conjunto com o qual foram treinadas, ressaltando o problema de overfitting que pode ser causado por uma má escolha de profundidade da árvore.

3.3.3 Random Forest

O modelo random forest resulta da utilização de *bagging* (ou *bootstrap aggregating*) com o modelo de árvores de decisão. O *bagging* consiste em subamostrar um conjunto de dados em vários subconjuntos, treinar de forma paralela um modelo para cada um dos subconjuntos e combinar o resultado deles. Assim, no caso da random forest, serão treinados vários modelos de árvore de decisão, como explicado na seção 3.3.2, e a média dos resultados dessas árvores será o valor estimado pelo modelo.

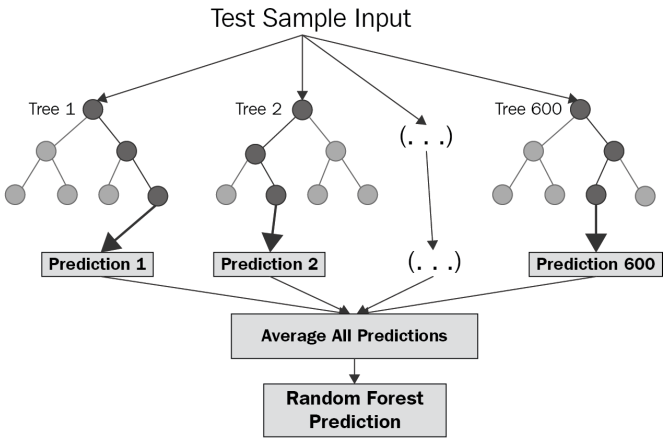

Figura 5: Exemplo de random forest [8].

Na figura 5, temos um exemplo de random forest regressora, como a que será treinada para o nosso dataset.

3.3.4 Gradient Boosting

O modelo gradient boosting resulta da utilização de *boosting* com o modelo de árvores de decisão. O *boosting* consiste em um modelo aditivo e sequencial no qual a cada iteração será adicionado um fator de correção, para o ajuste de parâmetros do modelo, com base no erro dele para os registros do conjunto de dados. No caso do gradient boosting, o ajuste de parâmetros é realizado em etapas, calculado por:

$$f^*(x) = \underset{\alpha,\Theta_M}{\operatorname{argmin}} \sum_{t=1}^{N} L(y(t), f_{M-1}(x) + \alpha_M h_M(x, \Theta_M)) \tag{6}$$

Nessa equação (6) temos, resumidamente, a solução do problema de otimização de uma função de avaliação $L(y(t), f_{M-1}(x) + \alpha_M h_M(x, \Theta_M))$ para cada iteração em M, ajustando o conjunto de parâmetros relacionado ao incremento de uma iteração. Em uma iteração, será calculado o resíduo $g_i(t)$ que será o gradiente negativo da função de avaliação, construindo um novo modelo $h(i)$ aproximado à função do resíduo, e atualizando-o como a soma do modelo da iteração anterior com uma combinação do novo modelo.

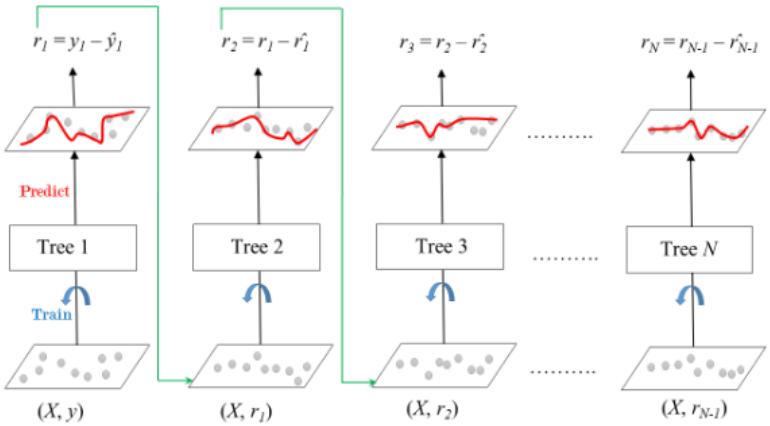

Figura 6: Exemplo de gradient boosting [9].

Na figura 6, temos um exemplo de modelo de gradient boosting regressor, como o que será treinado para o nosso dataset.

3.3.5 SVM

O modelo SVM regressor, também chamado de SVR (*Support Vector Regressor*), utiliza um kernel para calcular um hiperplano que minimize o erro fora da margem, ou seja:

$$\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \tag{7}$$

Para regressões não-lineares é necessário escolher um kernel para o SVR, como o RBF:

$$K(\vec{x}, \vec{l^i}) = e^{-\frac{\|\vec{x}-\vec{l^i}\|^2}{2\sigma^2}} \tag{8}$$

A função do kernel, definida por (8), será utilizada para projetar os pontos do conjunto de dados em uma dimensão maior, de forma que seja possível encontrar um hiperplano que corte o kernel minimizando a função (7).

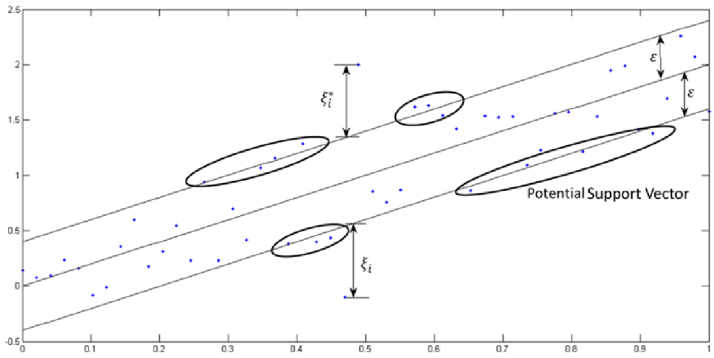

Figura 7: Exemplo de regressão SVM [10].

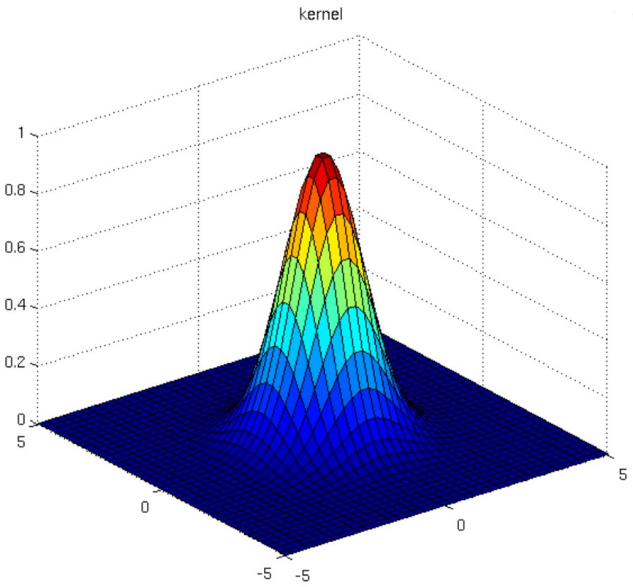

Figura 8: Visualização 3D do kernel RBF [11].

Na figura 7, temos um exemplo de modelo SVR linear. Já na figura 8 temos uma visualização 3D do kernel RBF, que será utilizado no nosso modelo SVR não-linear.

3.3.6 Redes Neurais

A rede neural ou rede neural artificial é um sistema computacional composto de elementos simples, chamados de neurônios, porém altamente conectados que processam informações de forma dinâmica. Elas são inspiradas nos neurônios do nosso cérebro, a fim de copiar o mecanismo de transmissão de sinais de excitação e respostas a esses sinais.

O perceptron é a forma mais básica de uma rede neural, e possui uma camada única com um neurônio. Existem 4 componentes básicos pra formar uma rede simples, que são basicamente os *inputs* (dados de entrada), os pesos, um *bias*, e os *outputs* (dados de saída). Além disso, pode haver uma função de ativação em cada neurônio que vai trazer uma não-linearidade ao modelo.

Por muito tempo as redes neurais não foram utilizadas pois suas aplicações não eram muito úteis, como por exemplo resolver um simples problema de OU-Exclusivo (XOR). Com o avanço computacional, as redes foram ficando mais complexas e consequentemente mais poderosas, e hoje em dia as redes neurais são o backbone do *Deep Learning*, tendo aplicações em diversas áreas como descoberta de novas drogas, previsões em séries temporais, criação artísticas, visão computacional, entre outros.

As arquiteturas mais utilizadas de redes neurais são as de camadas de neurônios, onde o neurônio de cada camada é conectado por meio de pesos, com todos os neurônios da camada anterior e posterior. Normalmente é considerado *Deep Learning* quando a rede neural tem mais de 3 camadas ocultas, mas é possível treinar redes com dezenas ou até centenas de camadas.

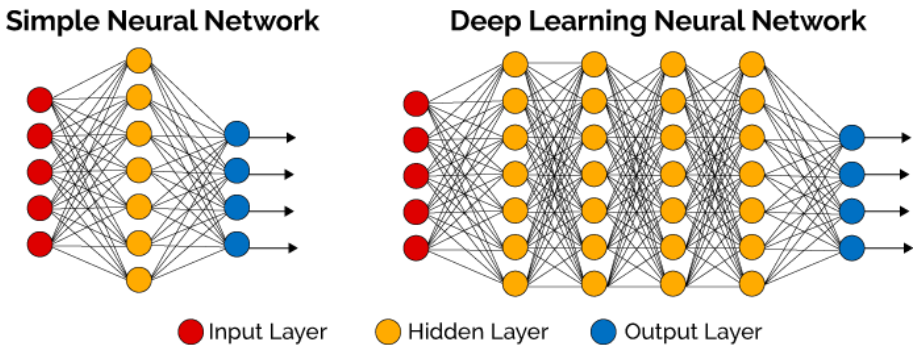

Figura 9: Rede Neural Simples vs Rede Neural Profunda [12].

Na figura 9, é possível ver arquiteturas de redes neurais simples e profundas. A camada de entrada, tem a quantidade de neurônios de acordo com a quantidade de variáveis de entrada dos dados, assim como a camada de saída tem a quantidade de neurônios baseado nas variáveis de saída (alvo). As camadas ocultas, podem ter a quantidade de neurônios variáveis, não dependendo da quantidade de *features* do conjunto de dados.

O modelo mais comum de redes neurais em camadas é o *feed-forward*, onde os dados fluem em somente uma direção para poder gerar a inferência. Esse tipo de rede é muito utilizada em problemas de classificação e regressão e podem substituir a maioria dos algoritmos tradicionais de aprendizado de máquina, pois seu treinamento é rapido e traz resultados significantes.

Começamos inicializando os pesos das camadas com uma função de inicialização como, por exemplo:

- Inicialização normal:

$$w \approx N(0, \sigma) \tag{9}$$

- Inicialização Xavier Glorot:

$$w \approx \mathcal{U}\left[-\sqrt{\frac{6}{\text{in} + \text{out}}}, \sqrt{\frac{6}{\text{in} + \text{out}}}\right] \tag{10}$$

- Inicialização He:

$$w \approx \mathcal{U}\left[-\sqrt{\frac{6}{\text{in}}}, \sqrt{\frac{6}{\text{in}}}\right] \tag{11}$$

O processo continua com o *foward-propagation*, onde os valores de x são multiplicados pelos pesos w. Um conceito importante para o processamento dos dados é o de função de ativação. A função de

ativação serve basicamente para ativar ou desativar um neurônio, definindo se ele vai ser importante ou não no processo de predição de acordo com as entradas no neurônio. Ela é extremamente útil para gerar não-linearidade ao modelo.

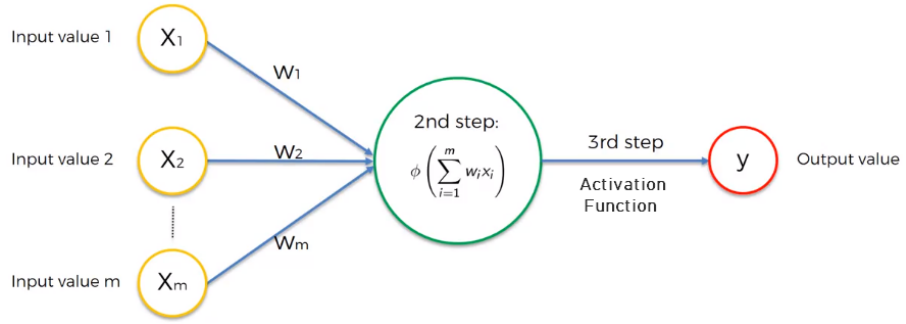

Figura 10: Esquema de processamento por um neurônio [13].

Na figura 10, é possível observar o fluxo de processamento de um neurônio. Após o neurônio pegar os *inputs* e multiplicar pelos respectivos pesos, somar essas multiplicações e também o *bias*, esse resultado é mapeado em um outro valor a partir da função de ativação, que servirá de *input* para os neurônios da próxima camada.

Das principais funções de ativação não-lineares, tem-se: logística (*sigmoid*), tangente hiperbólica (*tanh*) e ReLU. O formato delas podem ser visualizados na figura 11.

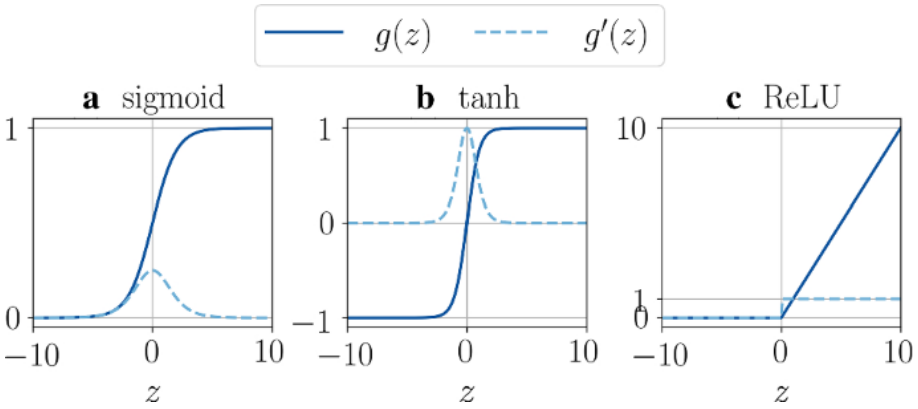

Figura 11: Funções de ativação e suas respectivas derivadas [14].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

A função logística serve para mapear qualquer valor no intervalo entre 0 e 1. É muito utilizada por modelos onde se quer ter uma probabilidade como resultado. A fórmula dessa função pode ser observada na equação (12).

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{13}$$

A função tangente hiperbólica é parecida com a logística por conta do seu formato, mas a sua amplitude varia entre -1 a 1. Ela é excelente para mapear problemas de intensidade, pois o 0 pode ser associado como o neutro/médio e os extremos como -1 e 1. A fórmula dessa função pode ser observada na equação (13).

$$f(x) = \max(0, x) \tag{14}$$

A função ReLU é parecida com a função linear ($y = x$), mas na verdade ela transforma os valores negativos em nulo. Ela é a principal função de ativação utilizada pois é computacionalmente mais eficiente que as outras duas mencionadas, pois sua derivada é mais simples, facilitando o cálculo do algoritmo

de *backpropagation*. Por outro lado, ela pode gerar derivadas nulas que vão acabar dando origem a "neurônios mortos" que acabam nunca sendo ativados. A ReLU também é utilizada como função de ativação dos neurônios da camada de saída quando se tem problemas em que não se podem ter valores negativos, por exemplo: preço, contagem, entre outros. A fórmula dessa função pode ser observada na equação (14).

Por fim, a rede atualiza seus pesos de forma a minimizar a função de perda através do algoritmo de *backpropagation* e os novos valores dos pesos são calculados por:

$$w^*_{i,j} = w_{i,j} - a\frac{\partial E(D,w)}{\partial w_{i,j}} \tag{15}$$

4 Resultados

4.1 Visualização dos dados

Com os dados limpos e normalizados, foram geradas diversas figuras de visualização para entender melhor o comportamento dos dados.

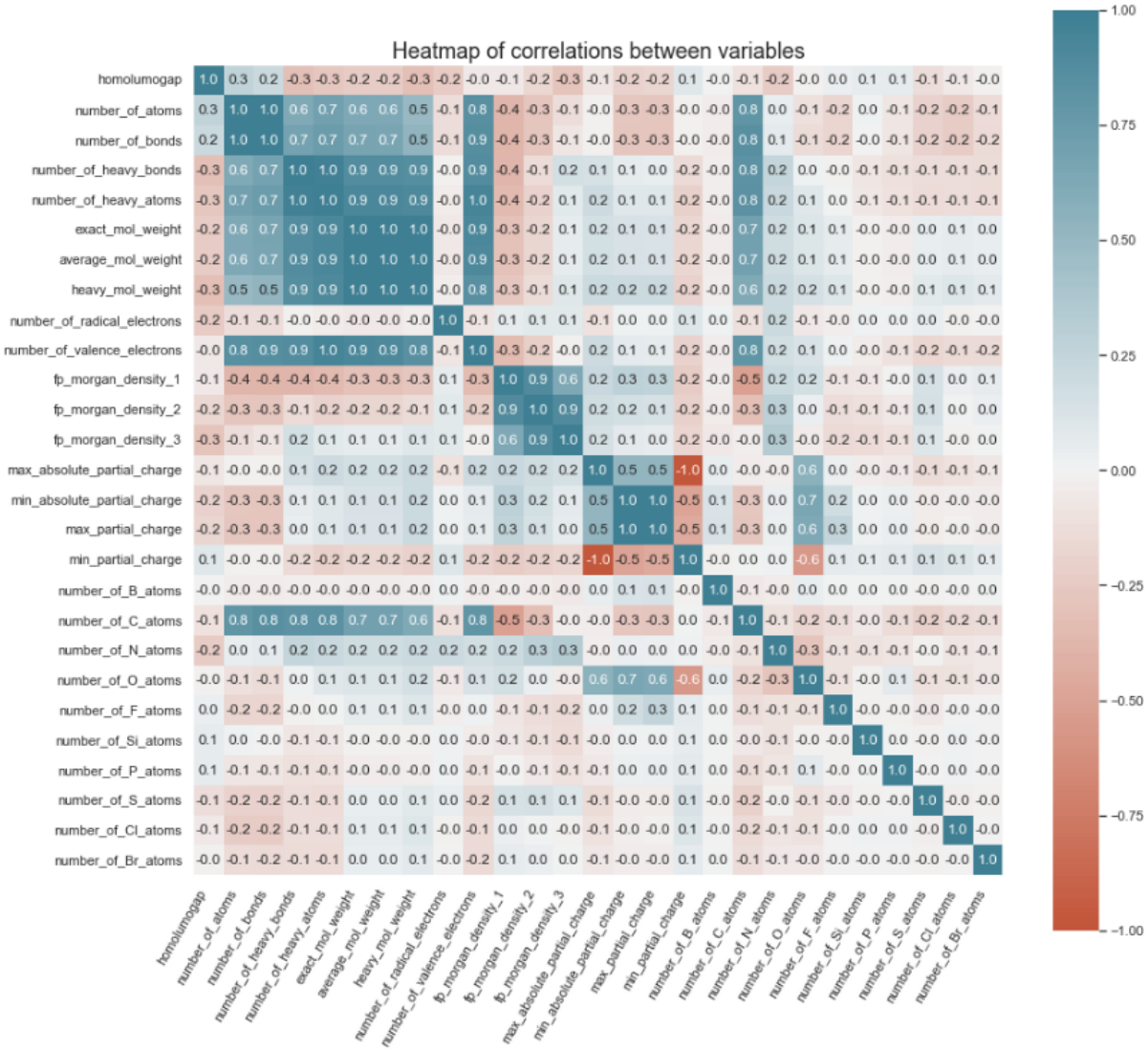

Figura 12: Mapa de calor de correlação entre as variáveis (independentes e alvo).

Através de um mapa de calor, é possível ver a correlação entre as variáveis, podendo assim identificar se alguma variável independente há grande capacidade de predição do alvo, a partir de uma alta correlação.

No caso dos dados apresentados, nenhuma variável independente teve alta correlação com a variável alvo, mas algumas variáveis independentes tiveram alta correlação com outras variáveis independentes. Essa situação pode ser um problema para certos modelos de aprendizado de máquina, por conta da redundância dos dados, porém nenhuma variável foi removida pois há outros modelos que não sofrem com essa alta correlação, como é o caso das Redes Neurais Artificiais.

Além do mapa de calor, também foram geradas visualizações sobre a dispersão dos dados, como os boxplots das variáveis e também a distribuição dos dados, através de histogramas.

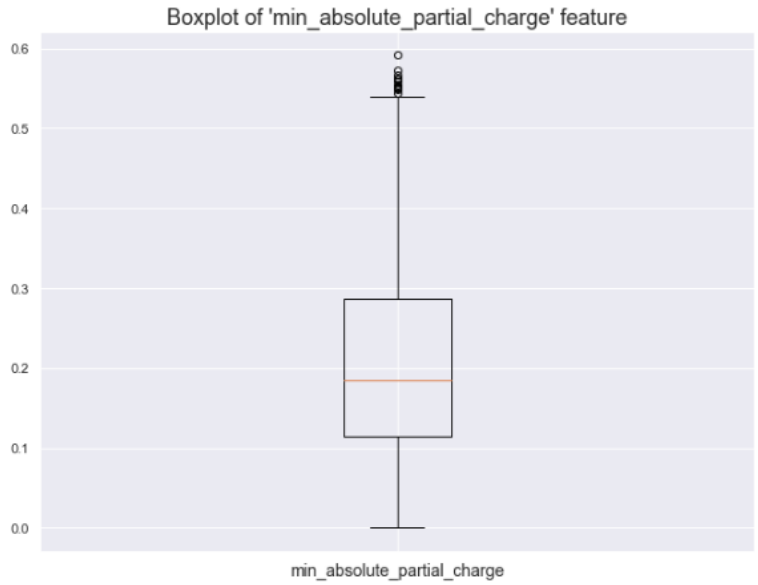

Figura 13: Exemplo de boxplot com dispersão simétrica da variável `min_absolute_partial_charge`.

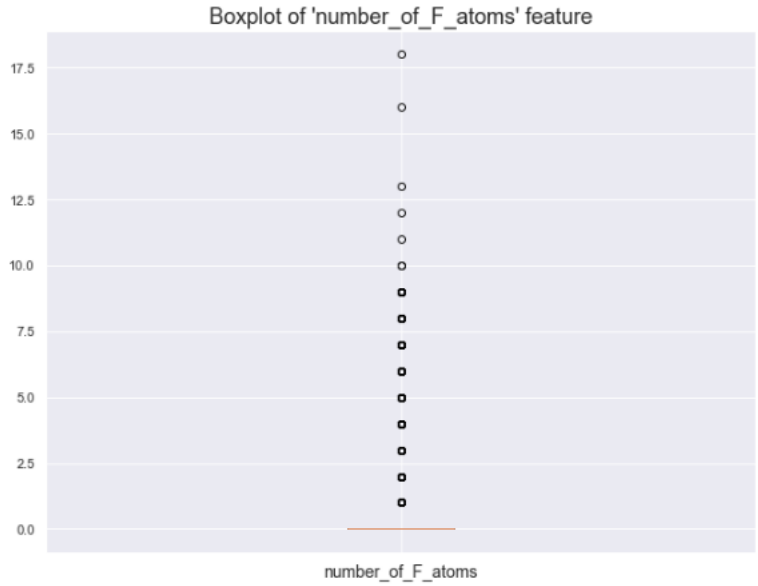

Figura 14: Exemplo de boxplot com dispersão assimétrica da variável `number_of_F_atoms`.

A partir das visualizações, percebeu-se que a distribuição dos dados é bastante heterogênea, tendo algumas variáveis com distribuições simétricas e outras com distribuições assimétricas.

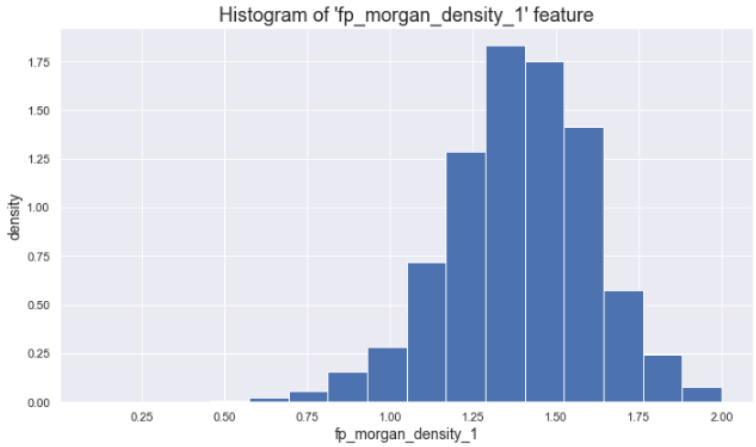

Figura 15: Exemplo de histograma da variável `fp_morgan_density_1`.

Todas as outras figuras gráficas geradas para visualização de dados podem ser consultadas no apêndice A.

4.2 Validação cruzada

Por se tratar de um conjunto de dados bastante grande (mais de 3 milhões de registros), os dados foram separados aleatoriamente em dois conjuntos: 90% em `train`, ficando com exatamente 3104579 registros e 10% em `test`, ficando com exatamente 344954 registros. Com essa divisão, é possível treinar os modelos somente no conjunto de treino, e por último avaliar os modelos treinados usando o conjunto de teste.

Conforme mencionado na metodologia, a equação (1) foi utilizada para normalizar os dados. A média e desvio padrão de cada variável utilizada foram colhidas somente do conjunto de treinamento, mas a normalização ocorreu tanto no conjunto de treinamento quanto no conjusto de teste. Para facilitar a transformação, foi utilizada a classe `StandardScaler` da biblioteca *scikit-learn*.

Para o treinamento de cada modelo, foram realizadas validações cruzadas de 10 ciclos, onde se utiliza nove décimos do conjunto de dados pra treinar e um décimo do conjunto para validar o resultado do modelo, variando sequencialmente qual parte do conjunto está sendo utilizada pra treinar e qual parte está sendo utilizada para validar o resultado. Essa técnica é importante para verificar a eficiência do modelo, principalmente no que diz respeito a *overfitting* ou *underfitting* dos dados, comportamentos que precisam ser evitados ao se ter um modelo pronto. Para automatizar a validação cruzada, foi utilizada a classe `GridSearchCV` da biblioteca *scikit-learn*, onde foi possível além de executar a validação cruzada, treinar os modelos com parâmetros variados. Os resultados de cada modelo podem ser encontrados nas subseções a seguir.

4.3 Modelos

4.3.1 Regressão Linear

Para a regressão linear, foi utilizada a classe `LinearRegression` da biblioteca *scikit-learn*. Por ser um modelo simples, não foi necessário buscar por nenhum parâmetro adicional além da instanciação por padrão.

No total, 10 fits foram realizados, já que a validação cruzada ocorreu em 10 ciclos.

O resultado do **MSE** do melhor modelo entre os fits, predizendo o conjunto de treinamento inteiro foi de **0.385**. Outros resultados, comparando com diferentes modelos podem ser encontrados na seção 4.4.

4.3.2 Árvore de decisão

Para a árvore de decisão, foi utilizada a classe `DecisionTreeRegressor` da biblioteca *scikit-learn*. Parâmetros diferentes foram passados para serem treinados, além dos padrões pela biblioteca. São eles:

- splitter: best, random

- max_depth: 2, 6, 12

O parâmetro `splitter` serve para dizer se deve considerar todas as *features* ao dividir o nó da árvore em dois (`best`) ou não (`random`). Já o parâmetro `max_depth` serve para controlar a profundidade máxima da árvore, podendo controlar o comportamento de *overfitting* e *underfitting* de acordo com a profundidade escolhida.

No total, 60 fits foram realizados, 10 ciclos em cada conjunto de parâmetros.

O modelo com melhor resultado foi o que teve o conjunto de parâmetros como `splitter = best` e `max_depth = 12`. O resultado do **MSE** desse modelo, predizendo o conjunto de treinamento inteiro foi de **0.327**. Outros resultados, comparando com diferentes outros modelos podem ser encontrados na seção 4.4.

4.3.3 Random Forest

Para a random forest, foi utilizada a classe `RandomForestRegressor` da biblioteca *scikit-learn*. Parâmetros diferentes foram passados para serem treinados, além dos padrões pela biblioteca, são eles:

- bootstrap: True, False
- max_depth: 2, 6, 12
- n_estimators: 5, 50

O parâmetro `bootstrap` serve para dizer se é necessário treinar todas as árvores com o conjunto de dados completo (`False`) ou se somente com um subconjunto (`True`). O parâmetro `max_depth` é igual ao mencionado na seção 4.3.2. Já o parâmetro `n_estimators` serve para definir a quantidade de árvores a serem treinadas (tamanho da floresta). Dependendo da quantidade definida, é necessário uma grande quantidade de memória disponível.

No total, 120 fits foram realizados, 10 ciclos em cada conjunto de parâmetros.

O modelo com melhor resultado foi o que teve o conjunto de parâmetros como `bootstrap = True`, `max_depth = 12` e `n_estimators = 50`. O resultado do **MSE** desse modelo, predizendo o conjunto de treinamento inteiro foi de **0.306**. Outros resultados, comparando com diferentes outros modelos podem ser encontrados na seção 4.4.

4.3.4 Gradient Boosting

Para o gradient boosting, foi utilizada a classe `GradientBoostingRegressor` da biblioteca *scikit-learn*. Parâmetros diferentes foram passados para serem treinados, além dos padrões pela biblioteca, são eles:

- learning_rate: 0.05, 0.1
- n_estimators: 5, 50
- subsample: 0.8, 1

O parâmetro `learning_rate` serve para dizer o quanto de contribuição cada árvore vai ter no treinamento, sendo quanto maior esse parâmetro, maior a contribuição. O parâmetro `n_estimators` é igual ao mencionado na seção 4.3.3, porém não há uma média entre as árvores e sim um sequenciamento a fim de minimizar o erro. Já o parâmetro `subsample` serve para estipular qual fração do conjunto de dados deve ser treinada em cada árvore independentemente.

No total, 80 fits foram realizados, 10 ciclos em cada conjunto de parâmetros.

O modelo com melhor resultado foi o que teve o conjunto de parâmetros como `learning_rate = 0.1`, `n_estimators = 50` e `subsample = 1`. O resultado do **MSE** desse modelo, predizendo o conjunto de treinamento inteiro foi de **0.440**. Outros resultados, comparando com diferentes outros modelos podem ser encontrados na seção 4.4.

4.3.5 SVM

Para o SVM, foi utilizada a classe `SVR` da biblioteca *scikit-learn*. Foi utilizado o kernel RBF nesse modelo, além de que parâmetros diferentes foram passados para serem treinados. São eles:

- C: 1, 100
- gamma: 0.8, 1

O parâmetro `C` é um parâmetro de regularização e serve para ajustar a margem de erro. Já o parâmetro `gamma` é o coeficiente do kernel RBF e serve para ajustar a curvatura do hiperplano.

No total, 40 fits foram realizados, 10 ciclos em cada conjunto de parâmetros. Infelizmente, o treinamento utilizando SVM é bastante pesado, e por conta disso foi necessário reduzir massivamente o tamanho do conjunto de dados de treinamento para apenas 10000 registros. Por conta disso, era esperado que os modelos treinados de SVM não conseguissem generalizar bem para o conjunto de treinamento completo, e principalmente no conjunto de teste. Além disso, seus resultados não podem ser comparados com os demais diretamente, pois os dados utilizados para treinar foram diferentes por conta da amostra reduzida.

O modelo com melhor resultado foi o que teve o conjunto de parâmetros como `C = 1`, `gamma = 0.8`. O resultado do **MSE** desse modelo, predizendo o conjunto de treinamento inteiro foi de **0.570**, como era de se esperar, um resultado pior do que os outros modelos até o momento. Outros resultados, comparando com diferentes modelos, apesar de não ser recomendado, podem ser encontrados na seção 4.4.

4.3.6 Redes Neurais

Para a criação das topologias de Redes Neurais Artificiais, foi utilizada a API `Keras` da biblioteca `Tensorflow`. Além disso, também foi utilizada a biblioteca `scikeras`, para utilizar o *wrapper* `KerasRegressor`, podendo assim conseguir fazer a busca de parâmetros utilizando o `GridSearchCV`.

Basicamente, as topologias seguiam a seguinte estrutura:

- Camada de entrada com função de ativação `relu`
- 1ª Camada Oculta com função de ativação `relu`, com 32 ou 64 neurônios, com taxa de `dropout` de 30% ou sem `dropout`
- 2ª Camada Oculta com função de ativação `relu`, com 32 ou 64 neurônios, com taxa de `dropout` de 30% ou sem `dropout`
- Camada de saída com função de ativação `linear`
- Otimizador: Adam
- Função de perda: MSE

O parâmetro `dropout` serve para dizer quantos neurônios serão ignorados aleatoriamente em cada época do treinamento. Esse parâmetro é fundamental para agir como regularizador da rede, já que consegue reduzir o *overfitting*. Uma rede que tenha muitas camadas e com muitos neurônios acaba facilmente aprendendo o padrão dos dados, e os regularizadores acabam tornando o modelo mais agnóstico. O otimizador serve para ajudar o modelo a ter uma maior performance de aprendizagem durante o treinamento, o Adam é normalmente o otimizador escolhido quando não se tem um conhecimento profundo sobre os dados, já que é facilmente configurável e consegue atingir bons resultados.

No total, 40 fits foram realizados, 10 ciclos em cada conjunto de parâmetros. Cada treinamento foi feito em 40 épocas, usando lotes (*batchs*) de 512 registros para cada atualização dos pesos da rede.

A topologia de Rede Neural com melhor resultado foi a que teve 64 neurônios nas camadas ocultas, e sem o regularizador `dropout`. O resultado do **MSE** desse modelo, predizendo o conjunto de treinamento inteiro foi de **0.197**. Outros resultados, comparando com diferentes outros modelos podem ser encontrados na seção 4.4.

4.4 Comparação dos resultados

Por se ter um conjunto de dados muito grande, era de se esperar que as Redes Neurais Artificiais conseguissem obter os melhores resultados. As 4 topologias de Redes Neurais foram capazes de obter resultados melhores que qualquer outro modelo de aprendizado de máquina. Na tabela 1 é possível observar, em ordem descrescente, a média e desvio padrão do MSE na validação cruzada para todos os modelos treinados. É possível notar que um modelo tão simples, como o de Regressão Linear conseguiu ter um resultado muito melhor do que diversos outros modelos mais sofisticados.

Conjunto de parâmetros	Média do MSE	Desvio Padrão do MSE	Tipo de Modelo
dropout_rate: 0, n_neurons: 64	0.209054	0.047653	Redes Neurais
dropout_rate: 0, n_neurons: 32	0.229017	0.048251	Redes Neurais
dropout_rate: 0.3, n_neurons: 64	0.261277	0.048857	Redes Neurais
dropout_rate: 0.3, n_neurons: 32	0.286353	0.051733	Redes Neurais
bootstrap: True, max_depth: 12, n_estimators: 50	0.317390	0.074361	Random Forest
bootstrap: True, max_depth: 12, n_estimators: 5	0.321892	0.076026	Random Forest
bootstrap: False, max_depth: 12, n_estimators: 50	0.344578	0.084087	Random Forest
bootstrap: False, max_depth: 12, n_estimators: 5	0.344621	0.084135	Random Forest
max_depth: 12, splitter: best	0.344641	0.084111	Árvore de Decisão
—	0.388943	0.065095	Regressão Linear
max_depth: 12, splitter: random	0.420922	0.089476	Árvore de Decisão
learning_rate: 0.1, n_estimators: 50, subsample: 1	0.450435	0.083229	Gradient Boosting
learning_rate: 0.1, n_estimators: 50, subsample: 0.8	0.450961	0.083613	Gradient Boosting
bootstrap: True, max_depth: 6, n_estimators: 50	0.537048	0.119244	Random Forest
bootstrap: True, max_depth: 6, n_estimators: 5	0.537797	0.118632	Random Forest
learning_rate: 0.05, n_estimators: 50, subsample: 1	0.552082	0.102831	Gradient Boosting
bootstrap: False, max_depth: 6, n_estimators: 5	0.552123	0.122632	Random Forest
max_depth: 6, splitter: best	0.552123	0.122632	Árvore de Decisão
bootstrap: False, max_depth: 6, n_estimators: 50	0.552123	0.122632	Random Forest
learning_rate: 0.05, n_estimators: 50, subsample: 0.8	0.552885	0.103418	Gradient Boosting
C: 1, gamma: 0.8	0.562424	0.025804	SVM
C: 100, gamma: 0.8	0.567231	0.025989	SVM
C: 100, gamma: 1	0.596168	0.025587	SVM
C: 1, gamma: 1	0.602217	0.027345	SVM
max_depth: 6, splitter: random	0.689717	0.130057	Árvore de Decisão
learning_rate: 0.1, n_estimators: 5, subsample: 0.8	0.812757	0.125674	Gradient Boosting
learning_rate: 0.1, n_estimators: 5, subsample: 1	0.813111	0.125373	Gradient Boosting
bootstrap: True, max_depth: 2, n_estimators: 50	0.836068	0.138524	Random Forest
max_depth: 2, splitter: best	0.836077	0.138518	Árvore de Decisão
bootstrap: False, max_depth: 2, n_estimators: 5	0.836077	0.138518	Random Forest
bootstrap: False, max_depth: 2, n_estimators: 50	0.836077	0.138518	Random Forest
bootstrap: True, max_depth: 2, n_estimators: 5	0.836124	0.138404	Random Forest
learning_rate: 0.05, n_estimators: 5, subsample: 0.8	0.897862	0.126136	Gradient Boosting
learning_rate: 0.05, n_estimators: 5, subsample: 1	0.898193	0.126180	Gradient Boosting
max_depth: 2, splitter: random	0.910961	0.117054	Árvore de Decisão

Tabela 1: Resultado da validação cruzada em 10 ciclos para todos os modelos treinados.

Já a tabela 2 mostra somente os melhores modelos de cada técnica de aprendizagem de máquina na validação cruzada. Dessa forma, é possível perceber que o modelo de SVM foi o que teve o pior resultado, como era de se esperar por conta do treinamento não ideal. Além disso, o modelo de Gradient Boosting teve um resultado inferior ao de Regressão Linear, demonstrando que nem sempre uma técnica complexa é melhor que uma técnica simples.

Conjunto de parâmetros	Média do MSE	Desvio Padrão do MSE	Tipo de Modelo
dropout_rate: 0, n_neurons: 64	0.209054	0.047653	Redes Neurais
bootstrap: True, max_depth: 12, n_estimators: 50	0.317390	0.074361	Random Forest
max_depth: 12, splitter: best	0.344641	0.084111	Árvore de Decisão
—	0.388943	0.065095	Regressão Linear
learning_rate: 0.1, n_estimators: 50, subsample: 1	0.450435	0.083229	Gradient Boosting
C: 1, gamma: 0.8	0.562424	0.025804	SVM

Tabela 2: Melhor resultado da validação cruzada em 10 ciclos para cada tipo de modelo treinado.

Por último, a tabela 3 mostra o resultado dos modelos da tabela 2 com o conjunto de teste que

foi separado conforme mencionado na seção 4.2. Com isso, se faz uma avaliação final da capacidade de generalização dos modelos em dados que não foram treinados. A ordem de classificação do melhor modelo de cada técnica pelo erro foi exatamente igual para a tabela 2 evidenciando que no geral os modelos tiveram resultados parecidos no conjunto de treino e teste, não demonstrando o comportamento de *overfitting*.

Modelo	MSE	MAE
Redes Neurais	0.198305	0.327192
Random Forest	0.311386	0.418187
Árvore de Decisão	0.333308	0.430803
Regressão Linear	0.385191	0.477560
Gradient Boosting	0.438995	0.509825
SVM	0.569116	0.551528

Tabela 3: Resultados do MSE e MAE dos melhores modelos de cada técnica nos dados de teste

Além dos valores de MSE, também foram calculados os valores do Erro Médio Absoluto (do inglês, *Mean Absolute Error*, MAE), para fins de comparação com o *leaderboard* dos autores do PCQM4Mv2. O MAE é uma medida de erro amplamente utilizada, assim como o MSE, e é definida por:

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i| \tag{16}$$

Para a equação (16), que define a fórmula do MAE, temos que:

- m é o número de registros (linhas) no conjunto de dados.
- y_i é o valor real da variável alvo no $i_{ésimo}$ registro.
- $\hat{y}_i$ é o valor predizido da variável alvo no $i_{ésimo}$ registro.

Nesse *leaderboard* eram mostrados apenas os 7 melhores resultados de MAE, mas podemos perceber que os modelos utilizados eram muito sofisticados e que precisavam de um grande poder computacional. O erro do resultado obtido pelo nosso melhor modelo (Rede Neural) ficou um pouco menor que o dobro do erro do 7º colocado nesse ranking.

5 Conclusões

Após a avaliação dos resultados obtidos, podemos concluir que o melhor tipo modelo para o conjunto de dados que utilizamos foi o de Redes Neurais, visto que os quatro conjuntos de parâmetros treinados tiveram os menores valores de média do MSE, além de um baixo desvio padrão do MSE.

Infelizmente não foi possível treinar o SVM com o mesmo *dataset* dos outros modelos, por conta da complexidade no treinamento de grande quantidades de dados nessa técnica. Isso acabou prejudicando a avaliação desse modelo e a comparação com os outros.

Além disso, comparando o resultado do nosso melhor modelo (Rede Neurais) com o *leaderboard*, tivemos um resultado bom para um modelo tão pouco sofisticado perto dos modelos apresentados no ranking. Portanto, acreditamos que esse modelo deva ser empregado no deploy da solução.

Os próximos passos são treinar modelos mais complexos e que funcionem bem para o problema abordado, como, por exemplo, modelos de *Graph Neural Networks*, esperando-se ter um resultado ainda melhor.

O repositório do trabalho pode ser encontrado no GitHub.

Referências

[1] S. Kurth, M.A.L. Marques e E.K.U. Gross. "Density-Functional Theory". Em: *Encyclopedia of Condensed Matter Physics*. Ed. por Franco Bassani, Gerald L. Liedl e Peter Wyder. Oxford: Elsevier, 2005, pp. 395–402. ISBN: 978-0-12-369401-0. DOI: https://doi.org/10.1016/B0-12-369401-9/00445-9. URL: https://www.sciencedirect.com/science/article/pii/B0123694019004459.

[2] Wikipédia. *Diagram of the HOMO and LUMO of a molecule. Each circle represents an electron in an orbital; when light of a high enough frequency is absorbed by an electron in the HOMO, it jumps to the LUMO.* 2022. URL: https://en.wikipedia.org/wiki/HOMO_and_LUMO#/media/File:Molecule_HOMO-LUMO_diagram.svg (acedido em 13/03/2022).

[3] Open Graph Benchmark. *Learn about PCQM4Mv2 and Python package*. 2022. URL: https://ogb.stanford.edu/docs/lsc/pcqm4mv2/ (acedido em 13/03/2022).

[4] Wikipédia. *SMILES*. 2022. URL: https://pt.wikipedia.org/wiki/SMILES (acedido em 13/03/2022).

[5] Renesh Bedre. *Linear regression basics and implementation in Python*. 2022. URL: https://www.reneshbedre.com/blog/linear-regression.html (acedido em 13/03/2022).

[6] Samet Girgin. *Decision Tree Regression in 6 Steps with Python*. 2019. URL: https://medium.com/@sametgirgin/decision-tree-regression-in-6-steps-with-python-c1564b153b74 (acedido em 13/03/2022).

[7] scikit-learn developers. *1.10. Decision Trees — scikit-learn 1.0.2 documentation*. 2022. URL: https://scikit-learn.org/stable/modules/tree.html (acedido em 13/03/2022).

[8] Afroz Chakure. *Random Forest Regression*. 2019. URL: https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f (acedido em 13/03/2022).

[9] @nikki2398. *ML – Gradient Boosting - Geeksforgeeks*. 2020. URL: https://www.geeksforgeeks.org/ml-gradient-boosting/ (acedido em 13/03/2022).

[10] Mariette Awad e Rahul Khanna. "Support Vector Regression". Em: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, 2015, pp. 67–80. ISBN: 978-1-4302-5990-9. DOI: 10.1007/978-1-4302-5990-9_4. URL: https://doi.org/10.1007/978-1-4302-5990-9_4.

[11] David Duvenaud. *Kernel Cookbook*. 2022. URL: https://www.cs.toronto.edu/~duvenaud/cookbook/index.html (acedido em 13/03/2022).

[12] Mateus Coelho. *Inteligência Artificial e Deep Learning - Laboratório iMobilis*. 2019. URL: http://www2.decom.ufop.br/imobilis/inteligencia-artificial-e-deep-learning/ (acedido em 13/03/2022).

[13] Andrea Perlato. *The Activation Function - Andrea Perlato*. 2022. URL: https://www.andreaperlato.com/aipost/the-activation-function/ (acedido em 13/03/2022).

[14] Qinghua Jiang et al. "Multilayer perceptron neural network activated by adaptive Gaussian radial basis function and its application to predict lid-driven cavity flow". Em: *Acta Mechanica Sinica* (jan. de 2022). DOI: 10.1007/s10409-021-01144-5.

[15] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc., 2019. ISBN: 1492032646.

Lista de Tabelas

Lista de Figuras

A Figuras Gráficas

A.1 Boxplots

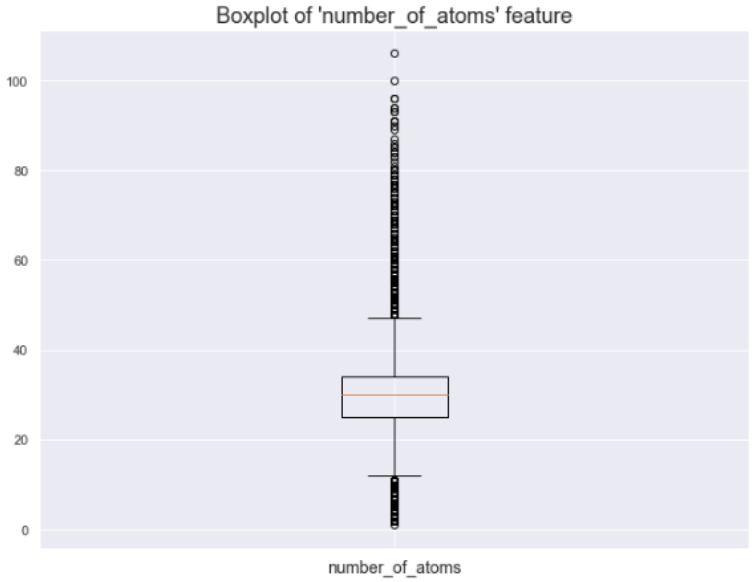

Figura A.1.1: Boxplot da variável `number_of_atoms`

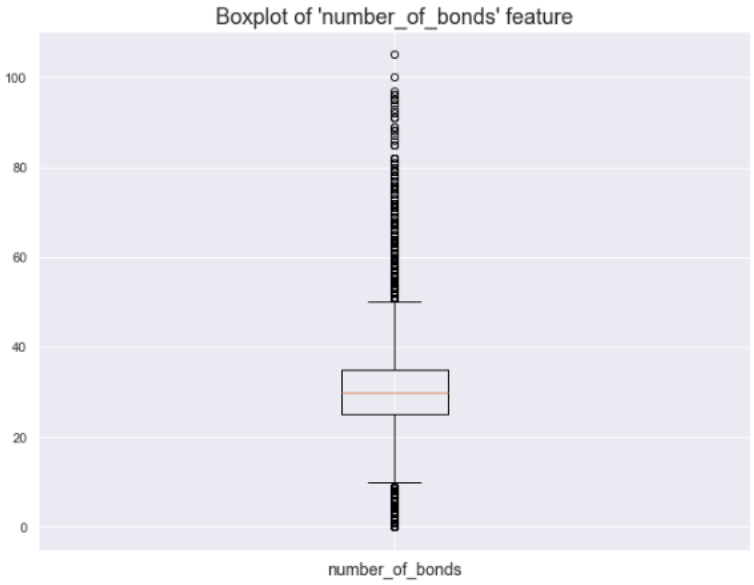

Figura A.1.2: Boxplot da variável `number_of_bonds`

A.2 Histogramas

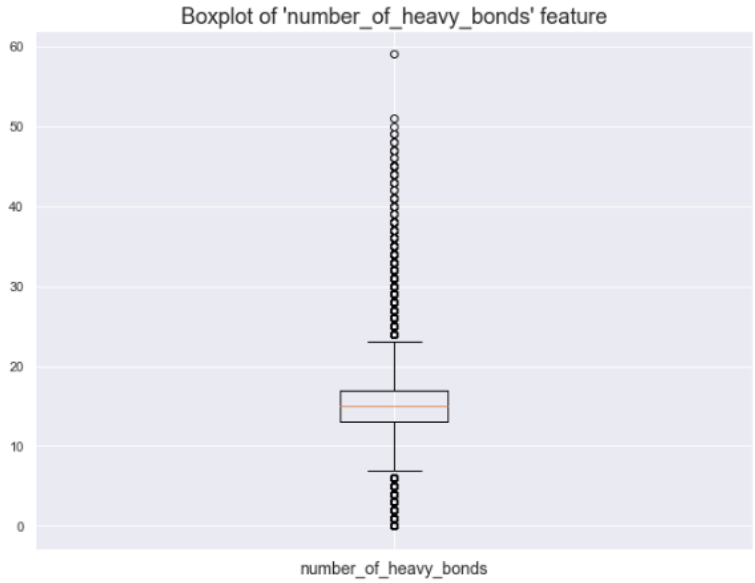

Figura A.1.3: Boxplot da variável `number_of_heavy_bonds`

Figura A.1.4: Boxplot da variável `number_of_heavy_atoms`

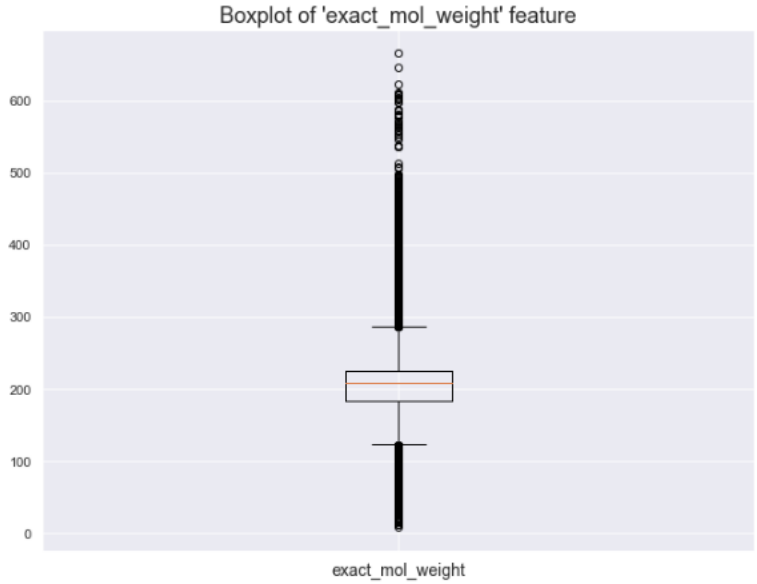

Figura A.1.5: Boxplot da variável `exact_mol_weight`

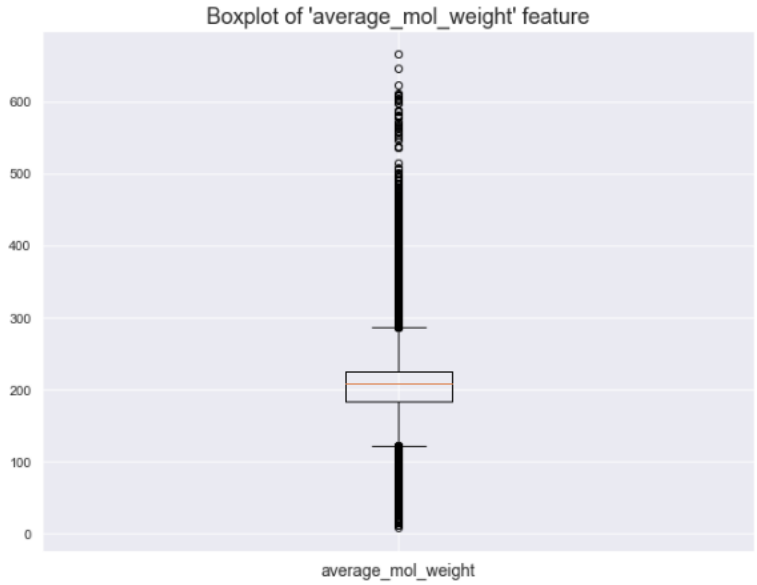

Figura A.1.6: Boxplot da variável `average_mol_weight`

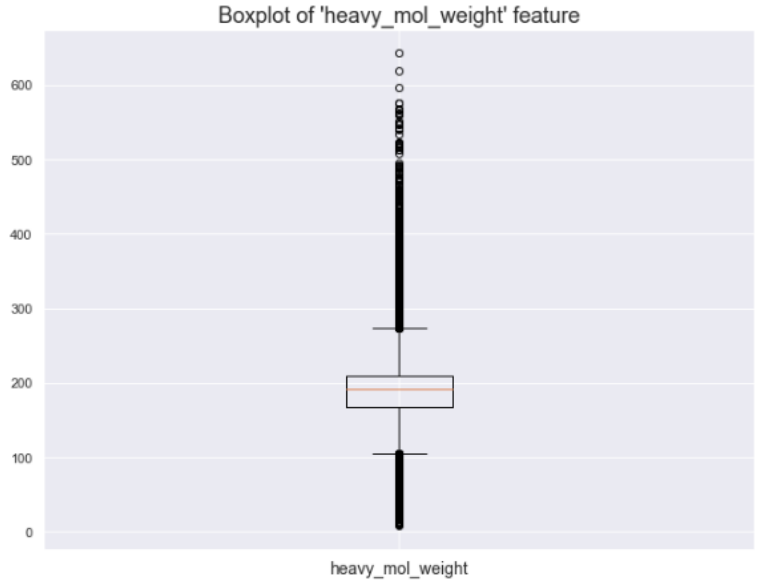

Figura A.1.7: Boxplot da variável `heavy_mol_weight`

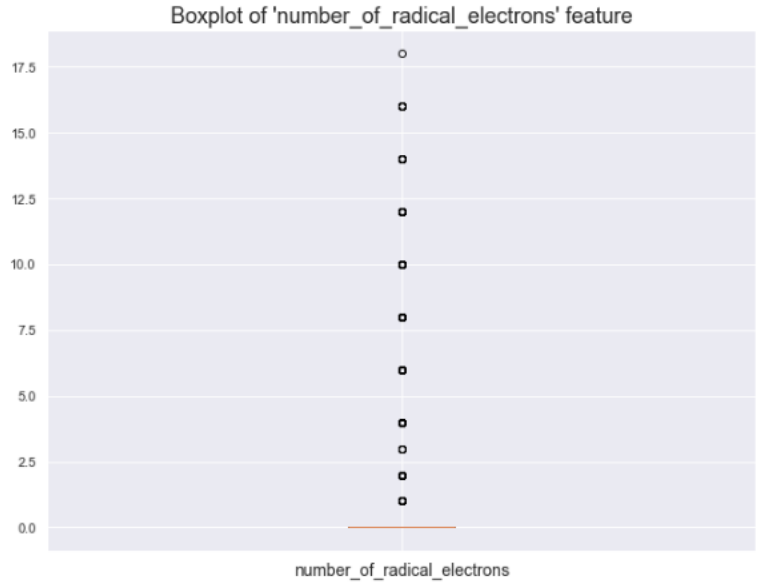

Figura A.1.8: Boxplot da variável `number_of_radical_electrons`

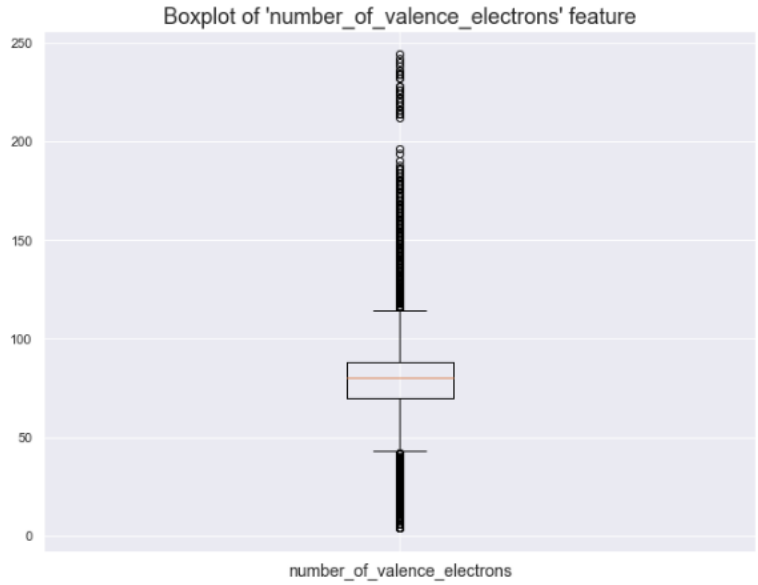

Figura A.1.9: Boxplot da variável `number_of_valence_electrons`

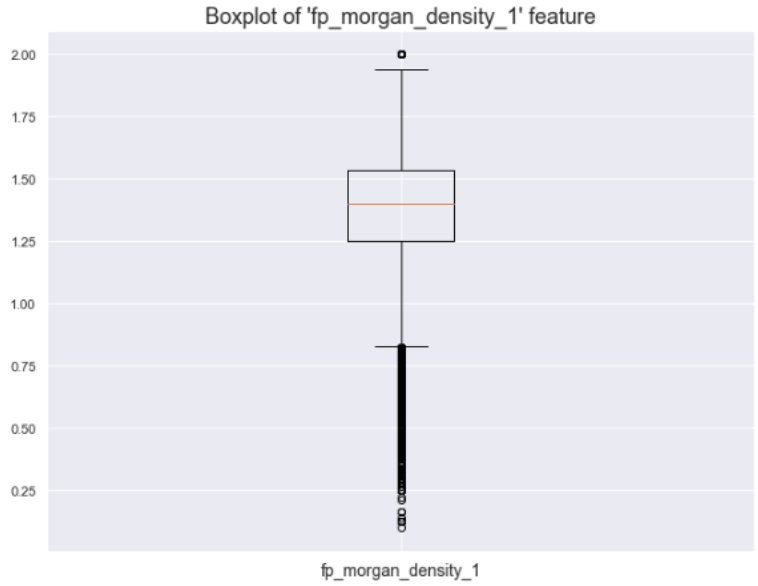

Figura A.1.10: Boxplot da variável `fp_morgan_density_1`

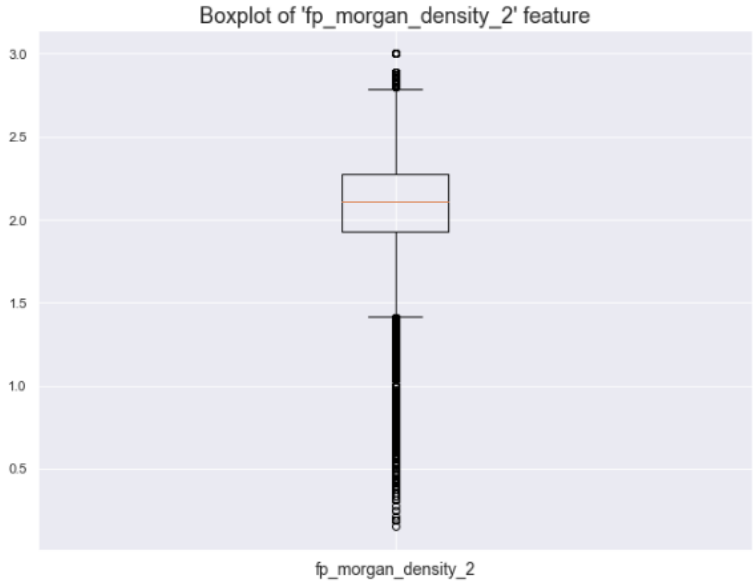

Figura A.1.11: Boxplot da variável `fp_morgan_density_2`

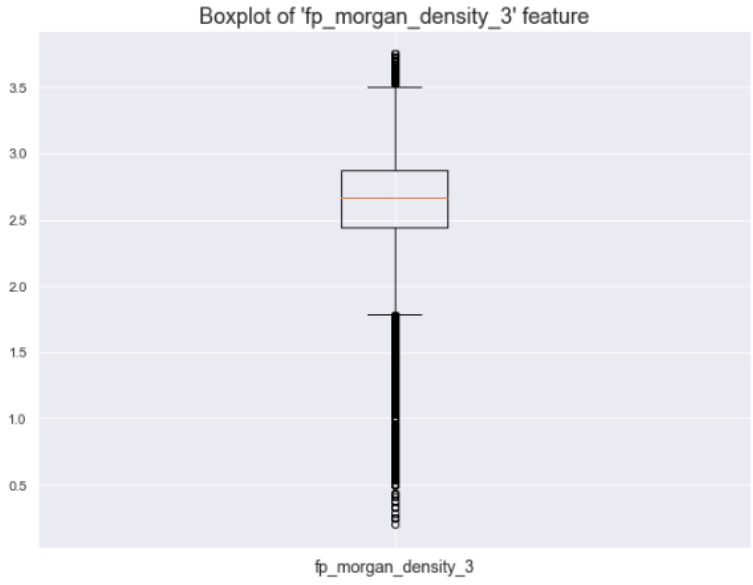

Figura A.1.12: Boxplot da variável `fp_morgan_density_3`

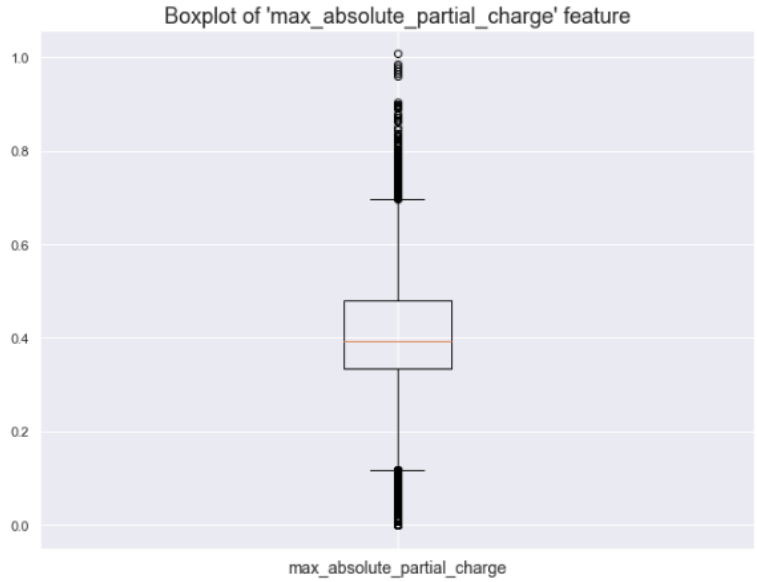

Figura A.1.13: Boxplot da variável `max_absolute_partial_charge`

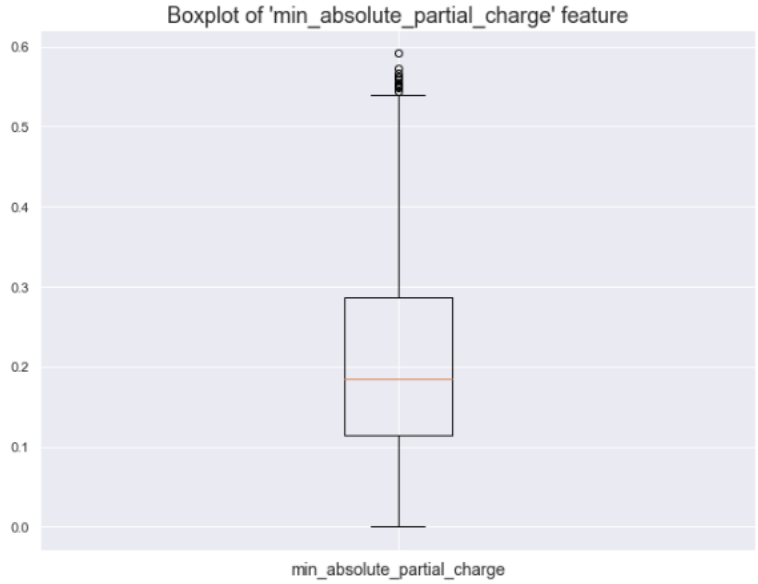

Figura A.1.14: Boxplot da variável `min_absolute_partial_charge`

Figura A.1.15: Boxplot da variável `max_partial_charge`

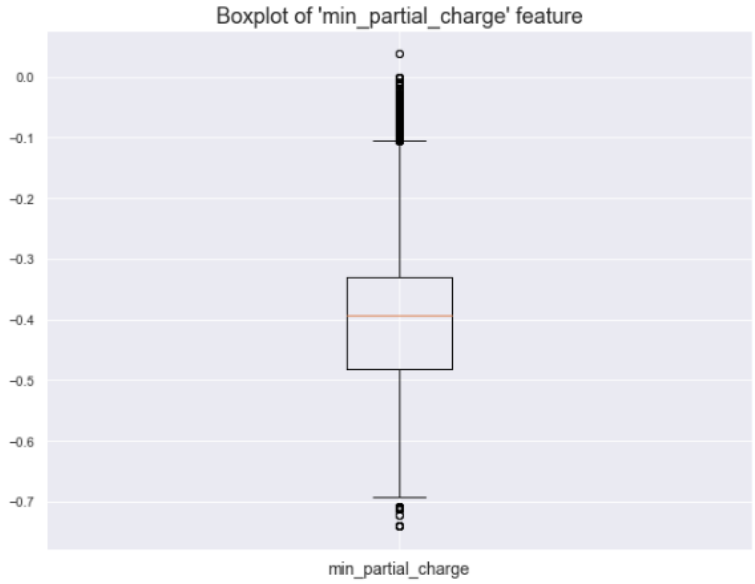

Figura A.1.16: Boxplot da variável `min_partial_charge`

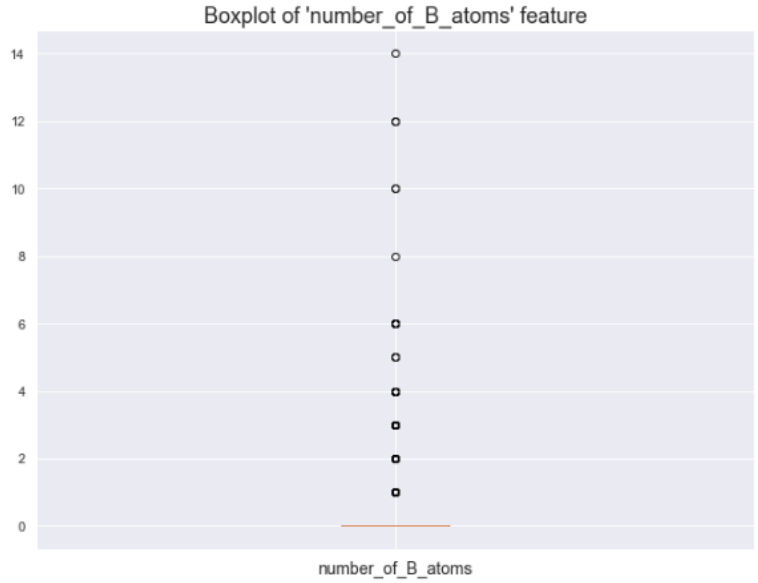

Figura A.1.17: Boxplot da variável `number_of_B_atoms`

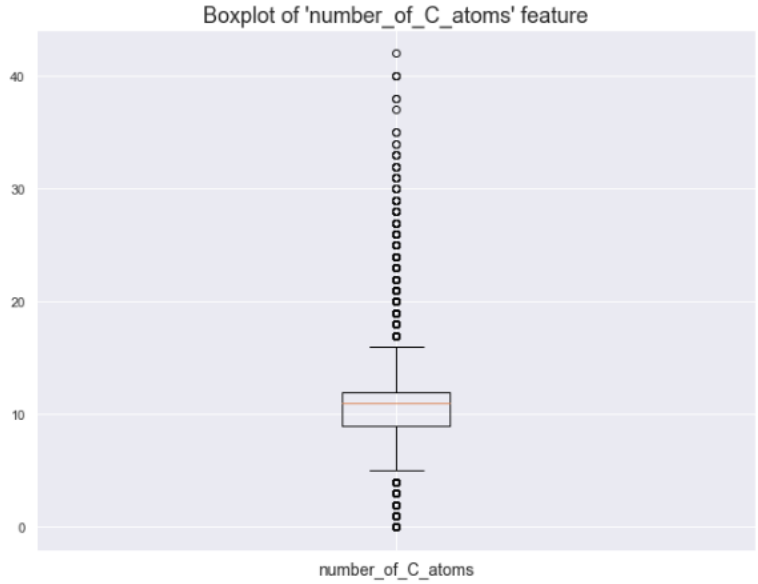

Figura A.1.18: Boxplot da variável `number_of_C_atoms`

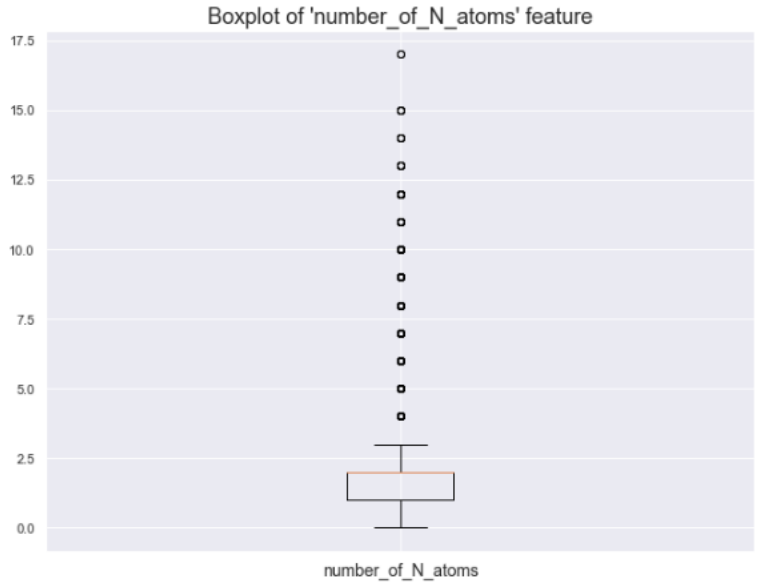

Figura A.1.19: Boxplot da variável `number_of_N_atoms`

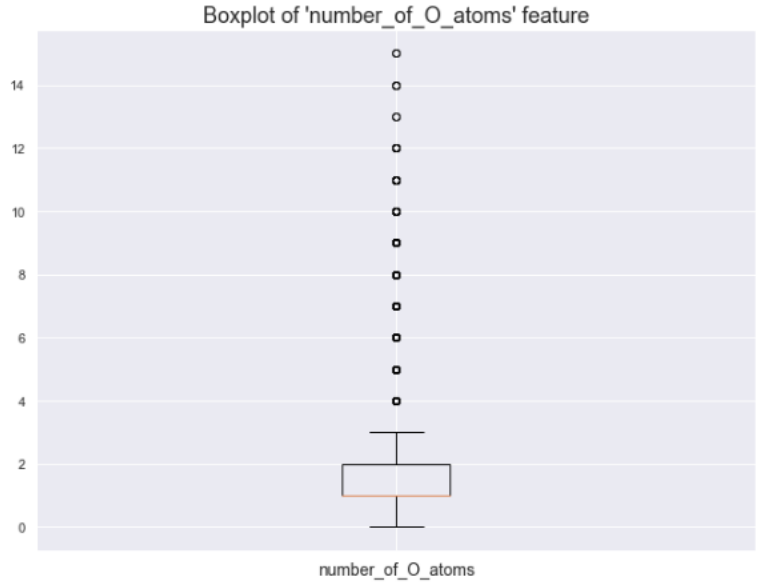

Figura A.1.20: Boxplot da variável `number_of_O_atoms`

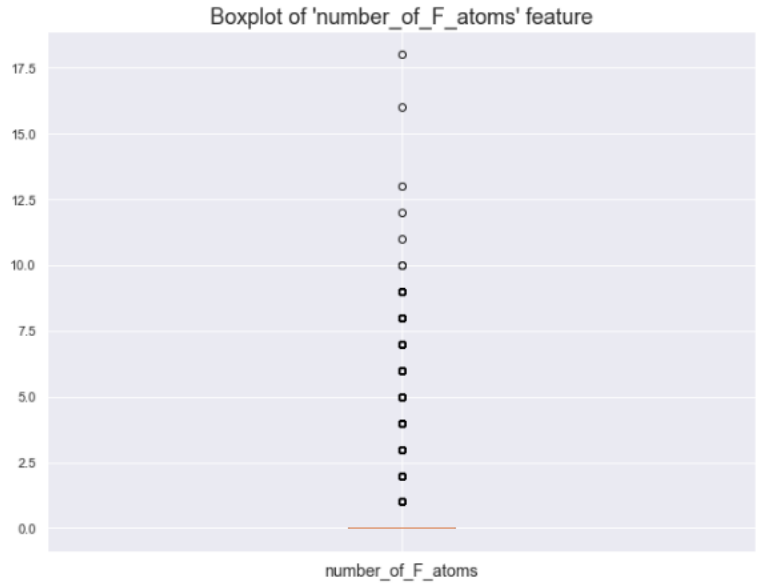

Figura A.1.21: Boxplot da variável `number_of_F_atoms`

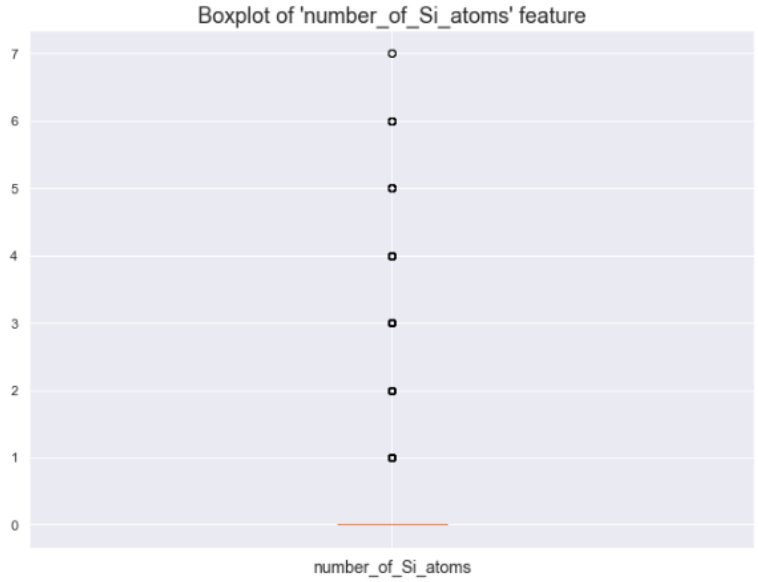

Figura A.1.22: Boxplot da variável `number_of_Si_atoms`

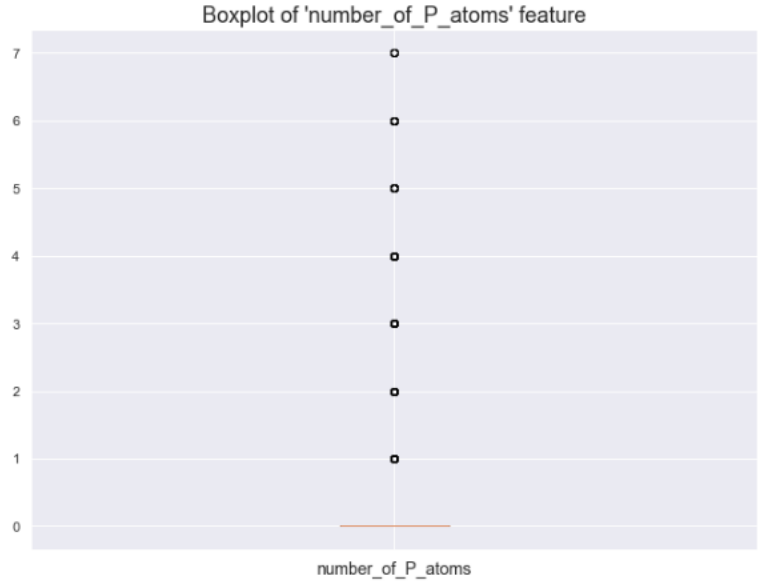

Figura A.1.23: Boxplot da variável `number_of_P_atoms`

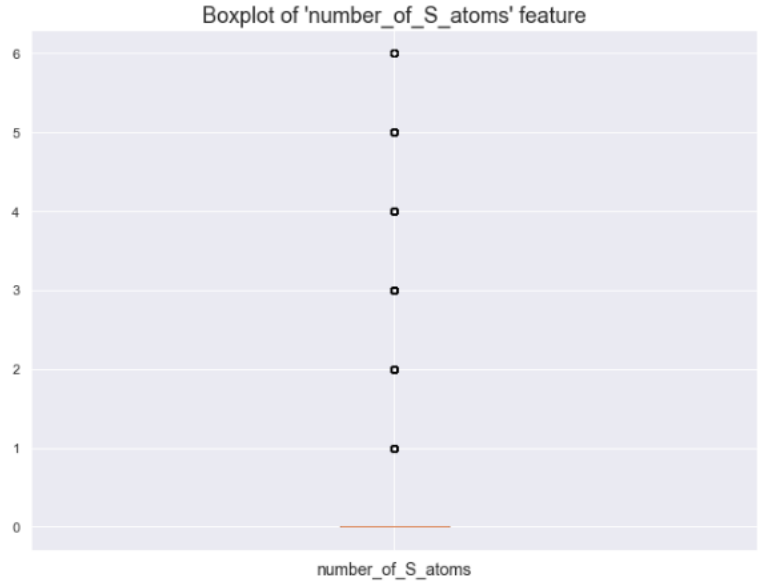

Figura A.1.24: Boxplot da variável `number_of_S_atoms`

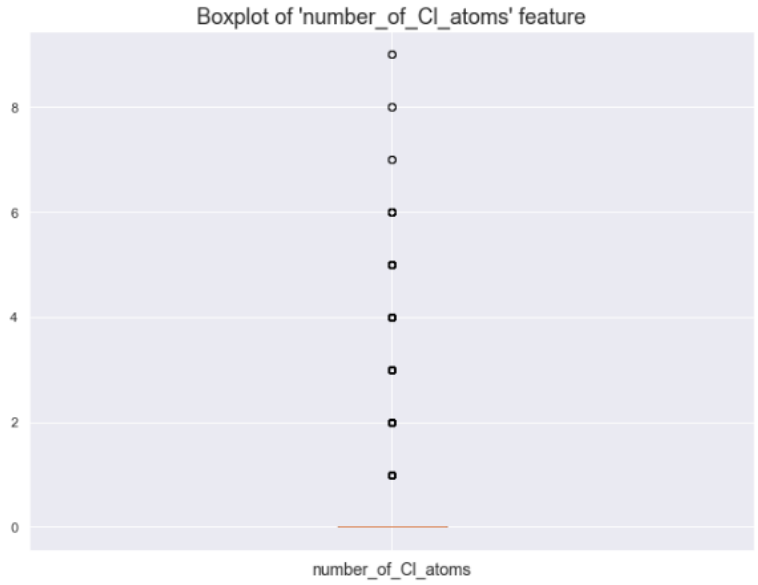

Figura A.1.25: Boxplot da variável `number_of_Cl_atoms`

Figura A.1.26: Boxplot da variável `number_of_Br_atoms`

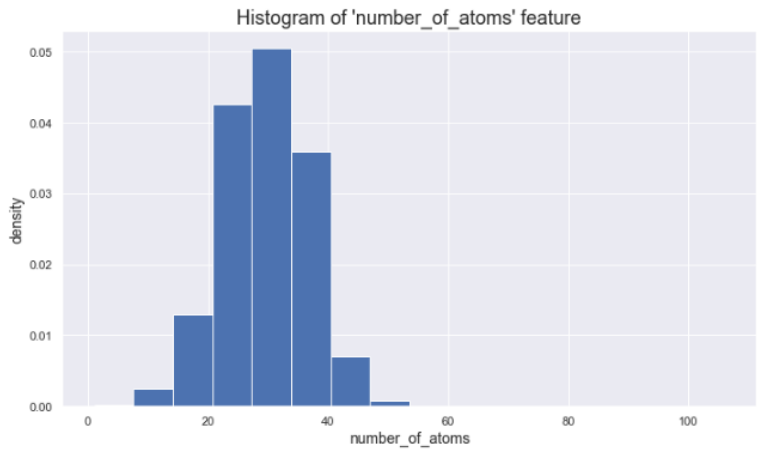

Figura A.2.1: Histograma da variável `number_of_atoms`

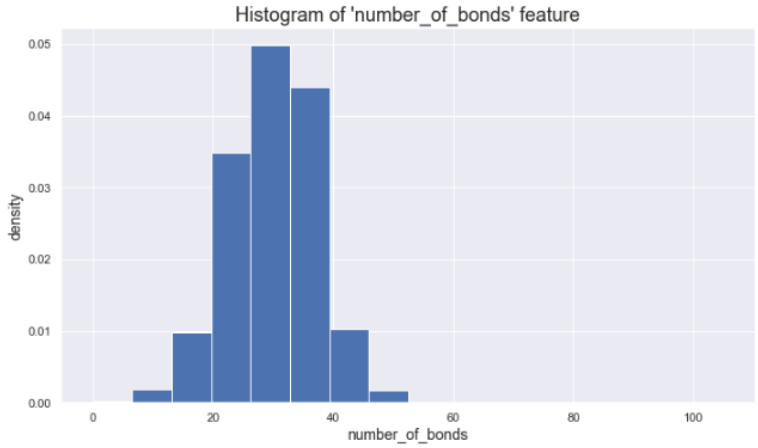

Figura A.2.2: Histograma da variável `number_of_bonds`

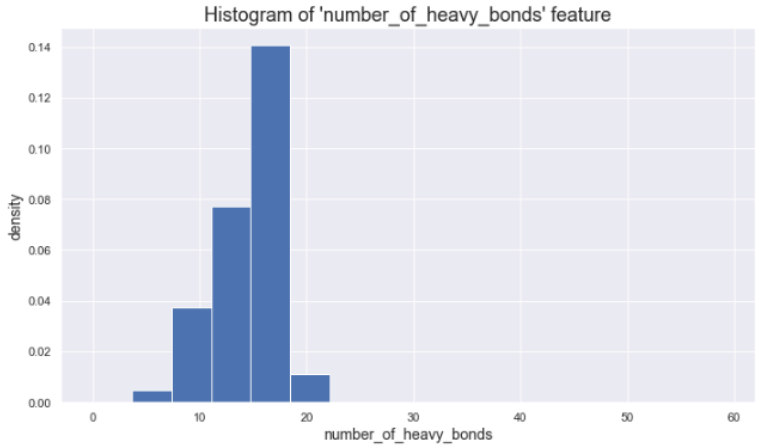

Figura A.2.3: Histograma da variável `number_of_heavy_bonds`

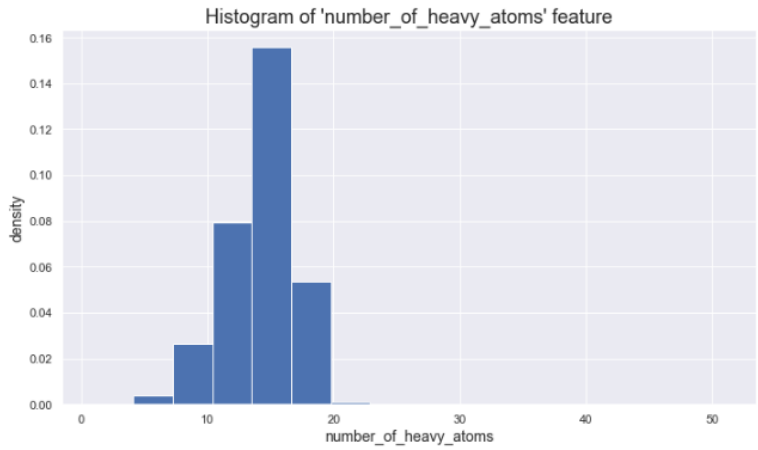

Figura A.2.4: Histograma da variável `number_of_heavy_atoms`

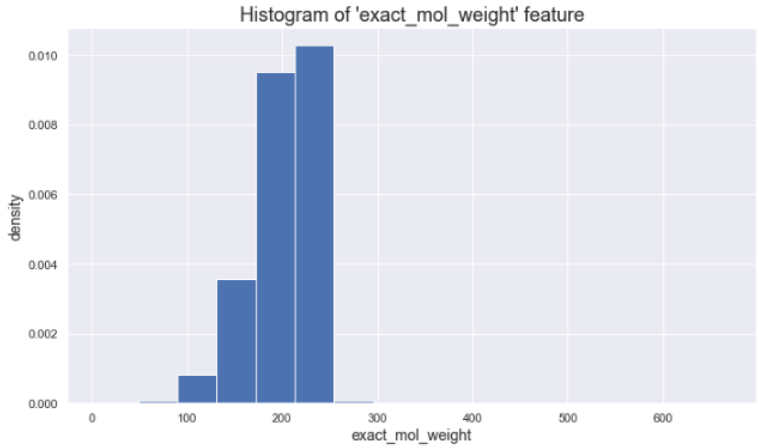

Figura A.2.5: Histograma da variável `exact_mol_weight`

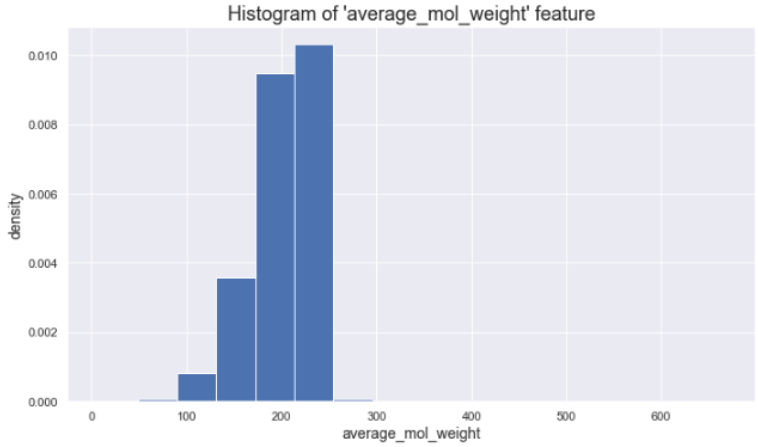

Figura A.2.6: Histograma da variável `average_mol_weight`

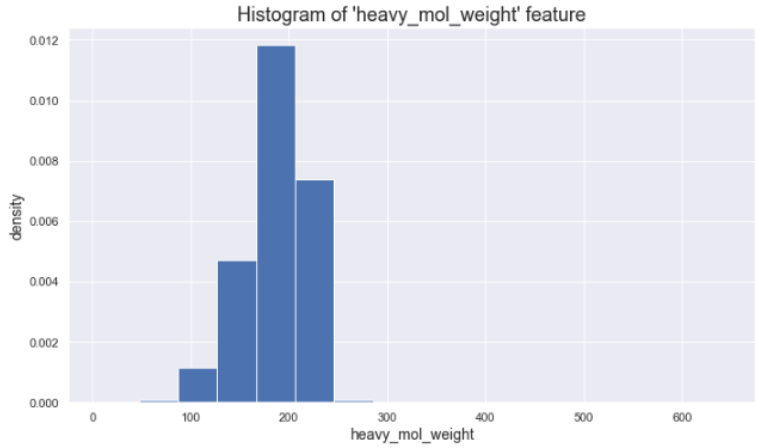

Figura A.2.7: Histograma da variável `heavy_mol_weight`

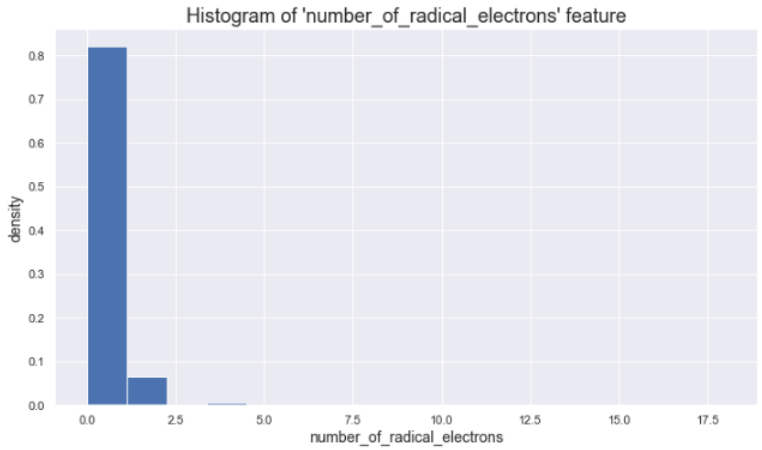

Figura A.2.8: Histograma da variável `number_of_radical_electrons`

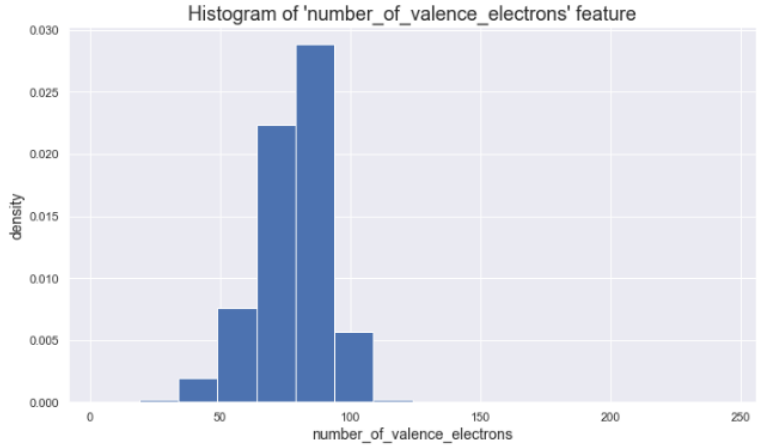

Figura A.2.9: Histograma da variável `number_of_valence_electrons`

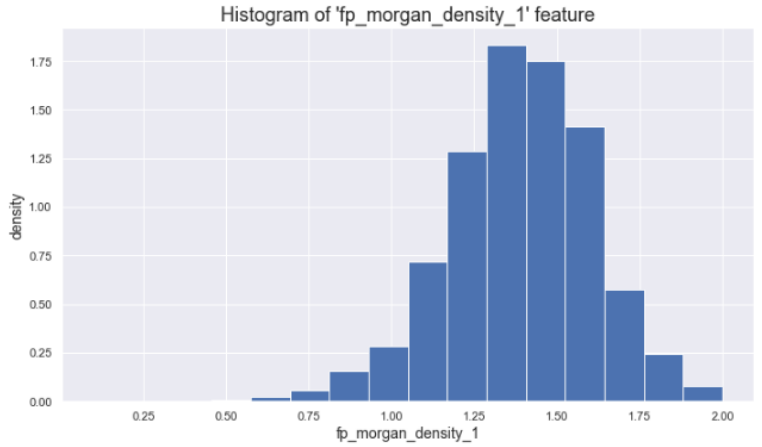

Figura A.2.10: Histograma da variável `fp_morgan_density_1`

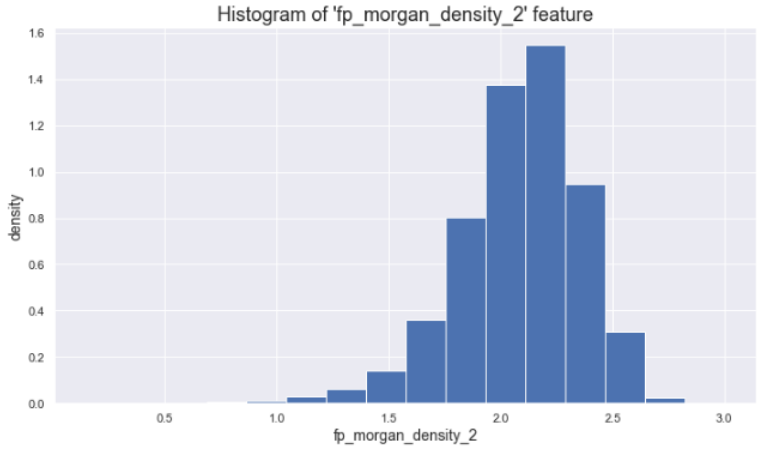

Figura A.2.11: Histograma da variável `fp_morgan_density_2`

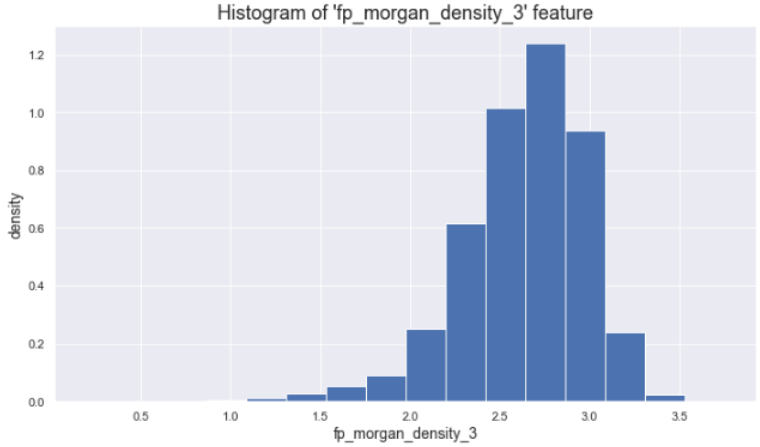

Figura A.2.12: Histograma da variável `fp_morgan_density_3`

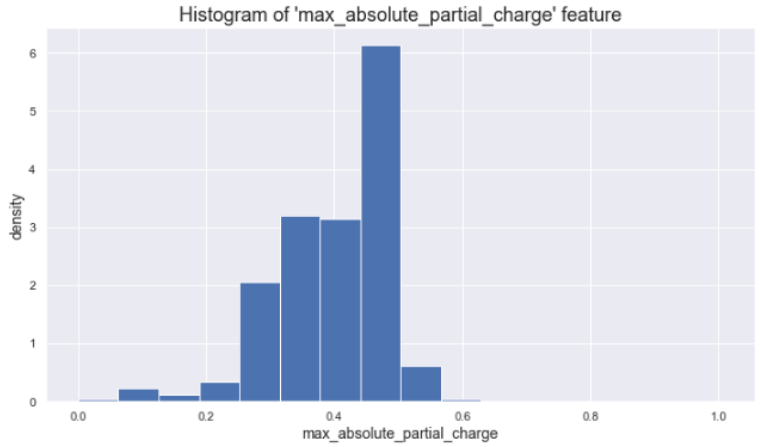

Figura A.2.13: Histograma da variável `max_absolute_partial_charge`

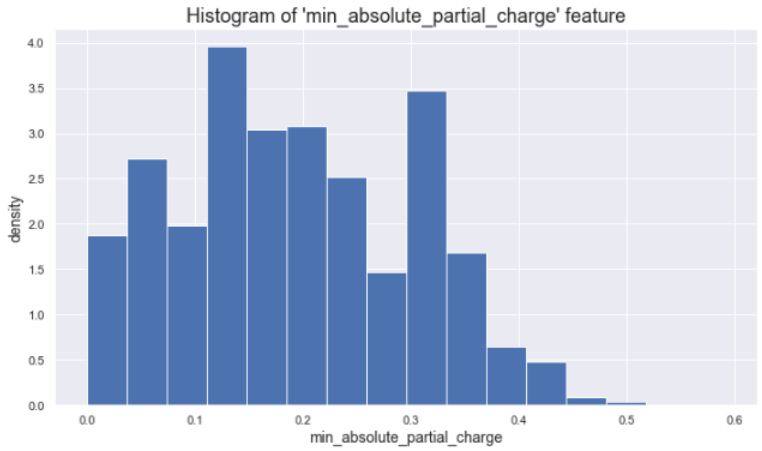

Figura A.2.14: Histograma da variável `min_absolute_partial_charge`

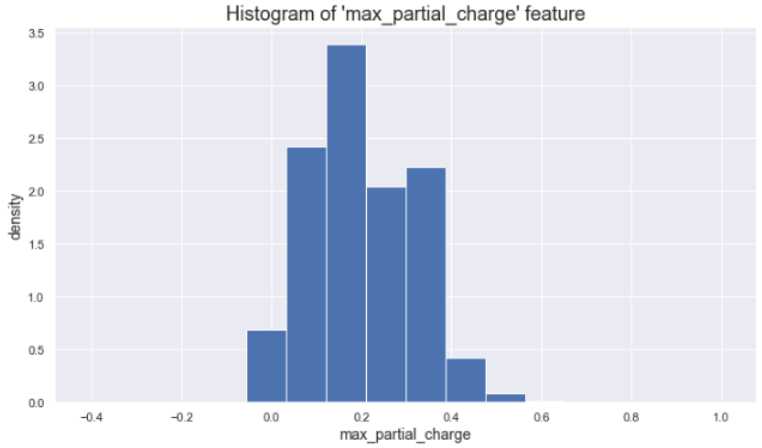

Figura A.2.15: Histograma da variável `max_partial_charge`

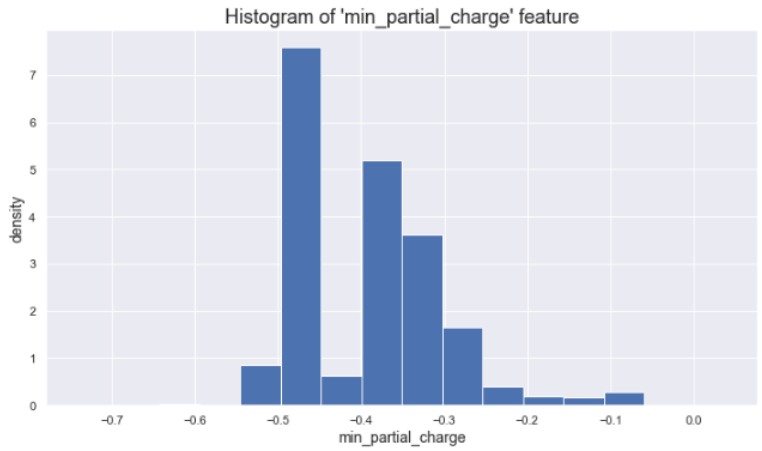

Figura A.2.16: Histograma da variável `min_partial_charge`

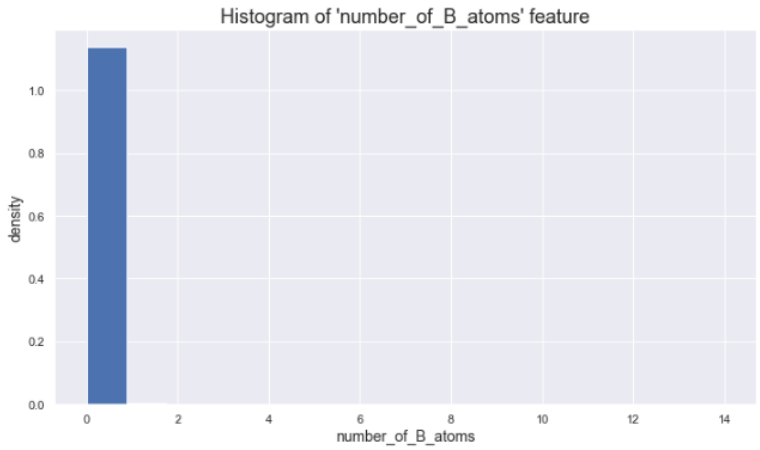

Figura A.2.17: Histograma da variável `number_of_B_atoms`

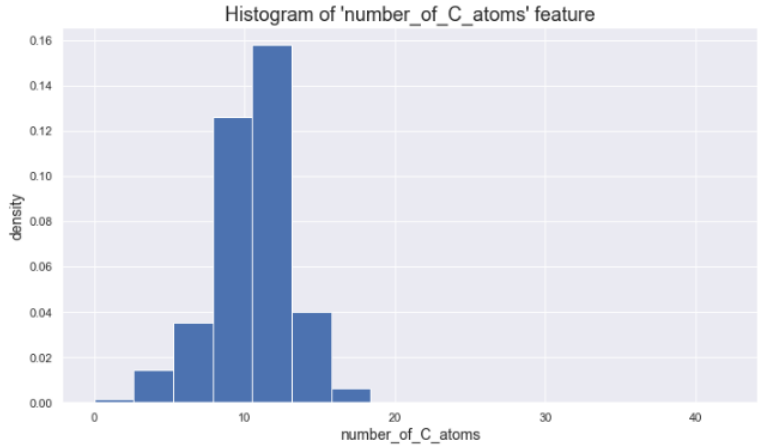

Figura A.2.18: Histograma da variável `number_of_C_atoms`

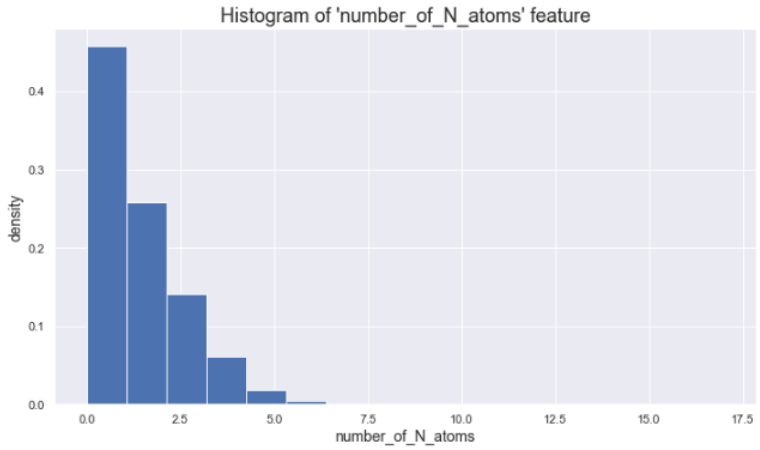

Figura A.2.19: Histograma da variável `number_of_N_atoms`

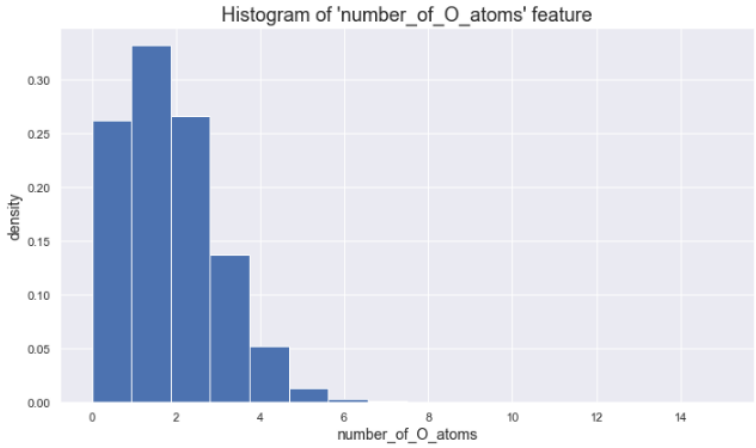

Figura A.2.20: Histograma da variável `number_of_O_atoms`

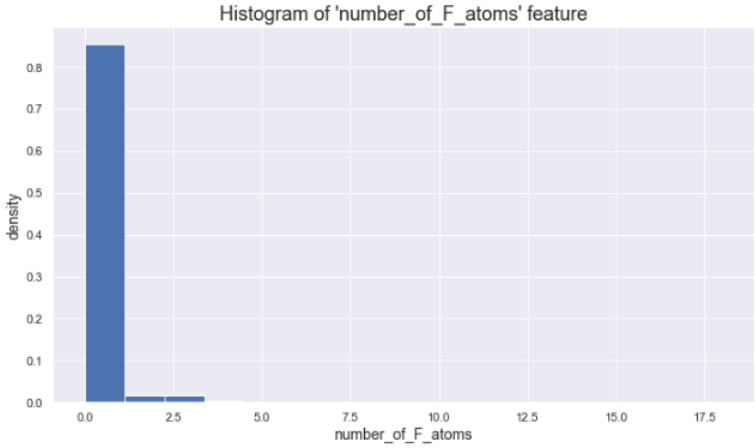

Figura A.2.21: Histograma da variável `number_of_F_atoms`

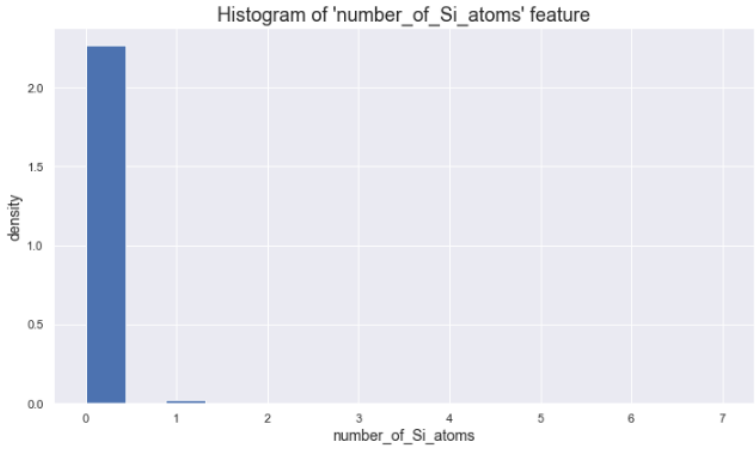

Figura A.2.22: Histograma da variável `number_of_Si_atoms`

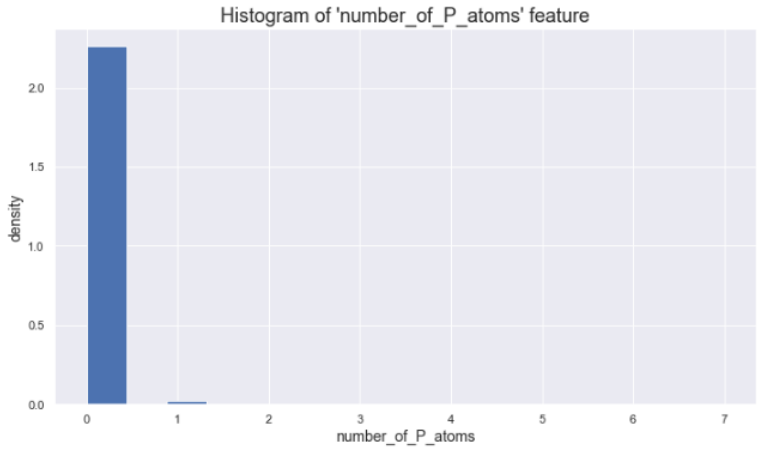

Figura A.2.23: Histograma da variável `number_of_P_atoms`

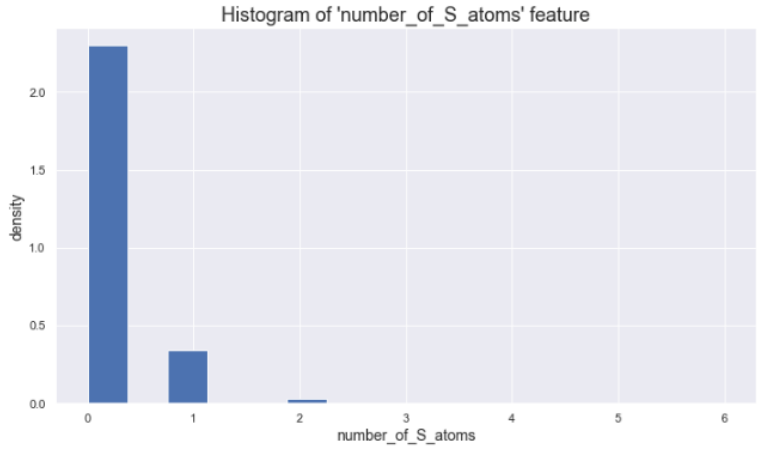

Figura A.2.24: Histograma da variável `number_of_S_atoms`

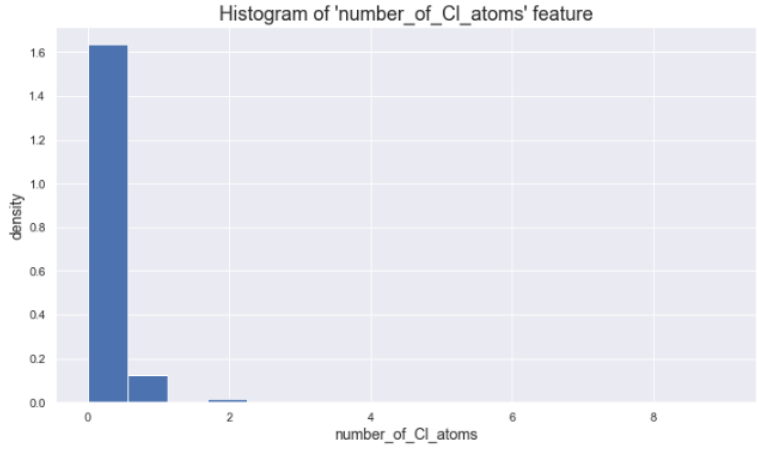

Figura A.2.25: Histograma da variável `number_of_Cl_atoms`

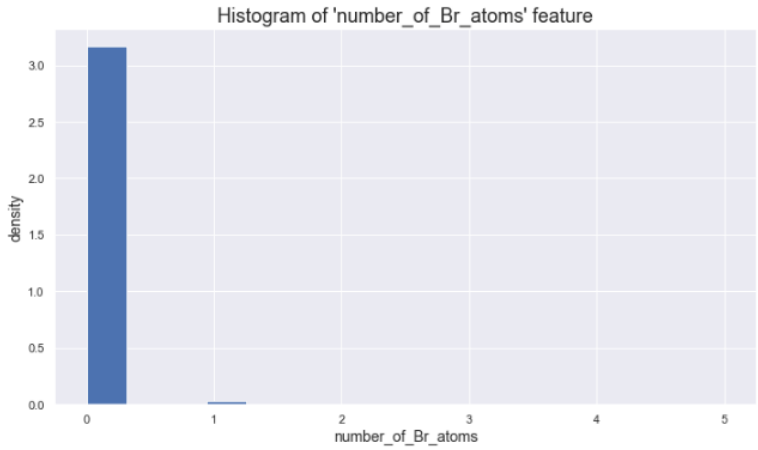

Figura A.2.26: Histograma da variável `number_of_Br_atoms`