ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M
MESTRADO
ENGENHARIA INFORMÁTICA

Sistema Inteligente de Recolha de Resíduos

João Pedro Carneiro Moreira

10/2022

Sistema Inteligente de Recolha de Resíduos
João Pedro Carneiro Moreira

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M
MESTRADO
ENGENHARIA INFORMÁTICA

João Pedro Carneiro Moreira.
Sistema Inteligente de Recolha de Resíduos.

10/2022

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

M

MESTRADO
Engenharia Informática

Sistema Inteligente de Recolha de Resíduos

João Pedro Carneiro Moreira

Orientador: João Ricardo Martins Ramos

Agradecimentos

Para que esta investigação possa ser dada como bem-sucedida é essencial agradecer a todas as pessoas que me acompanharam ao longo deste percurso, pois contribuíram muito para o seu sucesso e para a sua superação.

Esta foi uma caminhada difícil e duradoura, onde sem o apoio e a disponibilidade de todos os envolvidos nada teria sido possível de realizar e alcançar. Assim sendo, cabe-me agradecer:

Aos meus pais e ao meu irmão, que sempre me apoiaram e estiveram presentes nos bons e maus momentos, dando-me apoio, conforto e incentivo, não me deixando enfrentar este desafio sozinho;

À Catarina, que foi incansável e me deu todo o carinho e apoio possível, que sempre esteve presente fazendo-me lutar pelos meus objetivos;

A toda a minha restante família, avós, tios e primos, por também eles estarem sempre do meu lado e me fornecerem o apoio necessário para que fosse possível chegar aqui;

Ao Professor Doutor João Ramos, por ter aceitado o convite para meu orientador de dissertação, por todo o apoio e disponibilidade oferecidos no decorrer destes longos meses e pelos conselhos relativamente ao desenvolvimento desta investigação, que sem dúvida foram fundamentais para o sucesso da mesma.

Por último, agradeço ainda a todos os docentes da ESTG que me auxiliaram desde o início desta viagem académica, tornando possível o alcance desta meta, em particular, de forma satisfatória e bem-sucedida. Também eles contribuíram para a minha evolução enquanto ser humano e enquanto profissional estando, por isso, grato a cada um dos envolvidos por terem participado nesta minha viagem.

A todos vós, o meu muito obrigado!

Resumo

Os problemas de planeamento de rotas de veículos, amplamente estudados na literatura, têm uma aplicabilidade real na sociedade. Na recolha de resíduos urbanos pretende-se, por um lado, otimizar a capacidade de transporte dos veículos, e por outro, minimizar a distância por estes percorrida. Deste modo, pretende-se realizar o menor número de rotas possíveis e que garantam a recolha de todos os contentores.

Os modelos matemáticos de otimização linear para o problema de encaminhamento de veículos garantem soluções ótimas para a definição de rotas utilizadas na recolha de resíduos, no entanto podem ser computacionalmente difíceis de executar. Na presente investigação foram aplicados métodos de Inteligência Artificial para prever a capacidade ocupada no contentor e, com base nessa previsão, calcular as rotas mais eficientes para efetuar a recolha dos resíduos urbanos, considerando apenas os contentores identificados para recolha. Para isso, foram realizadas comparações entre diversos modelos de *Machine Learning*, procedendo-se também a ajustes dos respetivos hiper-parâmetros, de forma a obter uma solução eficiente para o cálculo da ocupação nos respetivos contentores.

Para o cálculo da rota foram implementadas diferentes abordagens exatas e heurísticas do problema de planeamento de rotas de veículos, de modo a decidir e implementar a melhor abordagem.

Como objetivo global desta dissertação, pretende-se apresentar uma boa solução, ou seja, bons tempos computacionais e uma gestão da recolha de resíduos mais eficiente e económica.

Palavras-chave: Gestão de Resíduos; *Machine Learning*; Modelos de Previsão; Otimização; Planeamento de rotas de veículos

Abstract

Vehicle route planning problems, widely studied in the literature, have real applicability in society. In the collection of urban waste, it is intended, on the one hand, to optimize the transport capacity of vehicles, and on the other, to minimize the distance traveled by them. In this way, it is intended to carry out the fewest possible routes and to guarantee the collection of all containers.

The linear optimization mathematical models for the vehicle routing problem guarantee optimal solutions for the definition of routes used in waste collection, however they can be computationally difficult to execute. In the present investigation, Artificial Intelligence methods were applied to predict the occupied capacity in the container and, based on this prediction, calculate the most efficient routes to carry out the collection of urban waste, considering only the containers identified for collection. For this, comparisons were made between different models of Machine Learning, also proceeding with the adjustments of the respective hyper-parameters, in order to obtain an efficient solution for the calculation of the occupancy in the respective containers.

For the route calculation, different exact and heuristic approaches to the vehicle route planning problem were implemented, in order to decide and implement the best approach.

As a global objective of this dissertation, it is intend to present a good solution, that is, good computational times and a more efficient and economical waste collection management.

Keywords: Waste Management; Machine Learning; Forecast Models; Optimization; Vehicle route planning

Índice

Índice de Figuras

Índice Tabelas

Siglas

APA – Agência Portuguesa do Ambiente;

BGPR – *Bayesian Gradient Boosting Regressor;*

CVRP – *Capacitated Vehicle Routing Problem;*

D-VRP – *Dynamic Vehicle Routing Problem*;

DNN - *Deep Neural Network;*

GBR – *Gradient Boosting Regressor*;

IoT – *Internet of Things;*

KNN - K-Nearest Neighbors;

LR – *Linear Regression*;

MAE – Mean Average Error;

MAPE – *Mean absolute percentage error;*

MLP – *Multi-linear Regression;*

MLPR – Multi-layer Perceptron Regressor;

MLR - Multi-layer Regressor;

MSE – Mean Squared Error;

OSM – *OpenStreetMap;*

OSRM – Open Source Routing Machine;

RF - *Random Forest;*

RMSE – Root Mean Squared Error;

SVM - *Support Vector Machines*;

VRP - *Vehicle Routing Problem;*

VRPMD – *Vehicle Routing Problem Multiple Depot;*

VRPPD – Vehicle Routing Problem with Pick-up and Delivering.

Capítulo 1

1. Introdução

Neste primeiro capítulo será feita uma introdução ao problema abordado ao longo deste trabalho que é desenvolvido no âmbito do Mestrado em Engenharia Informática. Será apresentada uma contextualização teórica, assim como a motivação e os objetivos para o desenvolvimento do trabalho. É ainda apresentado o cronograma e a estrutura seguida no desenvolvimento do projeto.

1.1. Contextualização

Numa sociedade e num mundo cada vez mais preocupado com o impacto ambiental provocado pelas ações humanas, os resíduos urbanos continuam a ter relevância neste campo devido ao seu impacto negativo. Promove-se a sustentabilidade, a redução, a reutilização e a reciclagem tanto por parte dos cidadãos, como por parte das empresas. No entanto, à semelhança de outras áreas, também aqui são detetados problemas e inconformidades. Por um lado, pretende-se estimular a separação dos resíduos realizada pelo consumidor final e, por outro, garantir que a recolha destes e manutenção dos espaços destinados à sua colocação pelas entidades responsáveis deste processo. Neste sentido, o presente projeto assenta na otimização de recursos no processo de recolha de resíduos urbanos.

A recolha de resíduos urbanos é necessária em todas as zonas, pois contribui para uma cidade limpa e evita problemas como, por exemplo, falta de higiene ou outros relacionados com a saúde pública. Para além disso, a disponibilização e correta utilização dos pontos de separação de resíduos (pontos de reciclagem) reduz os impactos ambientais que o ser humano provoca no planeta. Quando os resíduos são separados e colocados nos respetivos ecopontos, o seu tratamento torna-se mais fácil e diminui-se, de igual modo, a probabilidade de impactos nocivos para o meio ambiente.

Contudo, um dos vários problemas detetados e inerentes à produção de resíduos é a sua recolha. Isto é, esta recolha contínua acaba por se tornar ineficiente, porque as empresas de recolha nem sempre passam nos mesmos pontos e têm dias em que não exercem o seu trabalho (fins-de-semana, por exemplo), o que resulta numa acumulação de lixo urbano. Por outro lado, a realização diária de rotas fixas para a recolha dos mesmos pontos de reciclagem (ou de colocação de resíduos indiferenciados) garante a não acumulação de resíduos. No entanto, há uma maior probabilidade de se encontrar contentores/ecopontos vazios ou numa situação de subaproveitamento da sua capacidade, levando a um desperdício de recursos e, em termos ambientais, numa maior emissão de dióxido de carbono para a atmosfera.

Segundo a Agência Portuguesa do Ambiente (2021) [1], "*apesar de se verificar uma evolução favorável da recolha seletiva ao longo dos anos, a taxa de crescimento foi francamente baixa face ao que seria esperado, sendo a maioria dos resíduos recolhidos por via da indiferenciada*", tal como evidenciado na Figura 1.

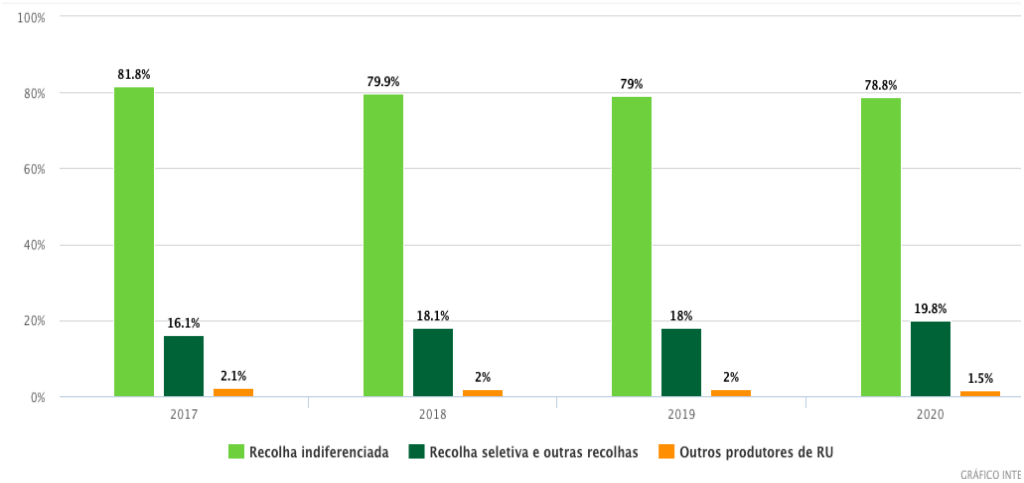


Figura 1 - Recolha de resíduos em Portugal (retirado de APA, 2021)

Apesar disso, é possível verificar que a taxa de preparação para a reutilização e reciclagem de Portugal tem vindo a aumentar (Figura 2), o que reforça a importância deste tema para o país e para o planeta e, consequentemente, a necessidade de melhorar o processo de identificação de pontos de recolha e planeamento de rotas para garantir um processo mais otimizado e com menor impacto ambiental, objetivos desta dissertação. Segundo a Agência Portuguesa do Ambiente, o decréscimo evidenciado de 2019 para 2020 reflete-se devido à pandemia COVID-19.

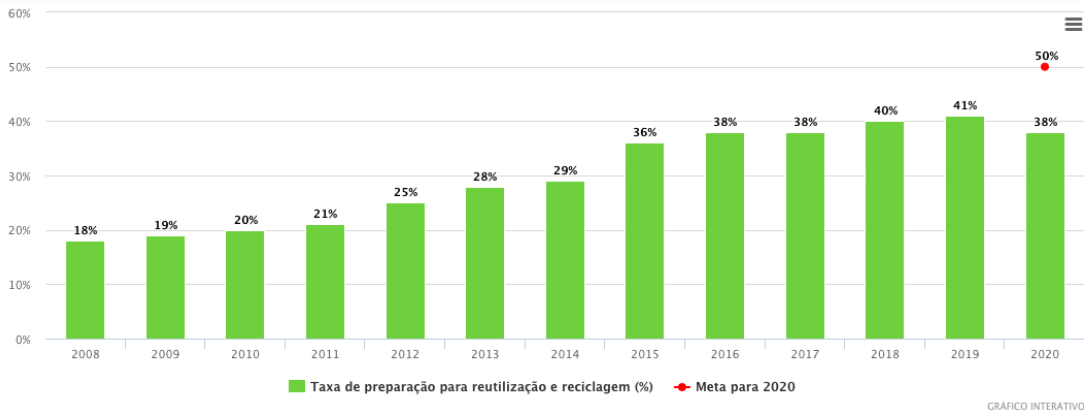


Figura 2 - Taxa de separação de resíduos em Portugal (retirado de APA, 2021)

Atualmente, a recolha de resíduos é realizada recorrendo sempre ao mesmo trajeto rodoviário (circuito). Considerando que na maior parte dos casos este percurso é o mais

otimizado possível, nem sempre há a necessidade de recolher os resíduos de todos os contentores associados a esse trajeto, pois estes podem não estar cheios na sua totalidade. Conforme a página do Porto Ambiente [2], existem cerca de 5200 equipamentos de deposição de resíduos indiferenciados distribuídos pela cidade do Porto e, para a sua recolha, são percorridos diariamente mais de três mil quilómetros.

O facto de se recolher sempre os resíduos, estando ou não o contentor com a sua capacidade máxima esgotada, provoca um consumo desnecessário de recursos (implicando assim um acréscimo no custo) e uma maior libertação de gases para a atmosfera, aumentando a poluição.

Por outro lado, existem picos de produção de resíduos como, por exemplo, nas épocas festivas do Natal e Passagem de Ano que provocam uma aceleração na utilização da capacidade dos contentores e, consequentemente, a acumulação de resíduos no seu exterior. Esta situação aumenta a probabilidade de transmissão de doenças devido à ingestão de resíduos por parte de animais que, consequentemente, pode afetar a pessoa responsável pelo animal ou ainda transmitir a outras pessoas/animais.

Para a realização desta dissertação, o foco prendeu-se na otimização de recursos e na diminuição da possível acumulação desnecessária de resíduos urbanos. Desta forma, o principal objetivo consiste em verificar o estado atual de empresas que recolhem resíduos e estudar o melhor método para colmatar as debilidades identificadas no sistema, tornando-o eficiente e, consequentemente, reduzir os custos para a empresa e a poluição atmosférica causada.

A fim de se atingir estes objetivos pretende-se implementar um sistema baseado em Inteligência Artificial capaz não só de prever a ocupação de cada contentor, mas também de determinar um trajeto otimizado (ou próximo da solução ótima) para a recolha dos diferentes contentores. Pretende-se que o sistema seja autónomo, ou seja, faça um planeamento automático das trajetórias adequadas sem a intervenção humana. Com base na previsão realizada é possível efetuar um planeamento diário mais ajustado, otimizando recursos como a distância (diminuindo o percurso para a recolha de lixo) e o combustível (inerente à otimização do percurso).

Além dos objetivos traçados inicialmente, o resultado final, também designado por solução, pretende evitar situações tanto de acumulação excessiva de resíduos, como de desperdício de combustível, evitando que os camiões do lixo responsáveis pela recolha saiam sem ter necessidade ou não saiam quando existe acumulação excessiva de resíduos. Desta forma, é possível evitar a permanência dos mesmos na base em caso de necessidade (acumulação de resíduos por excederem a capacidade máxima do contentor).

1.2. Motivação e objetivos principais

A otimização de recursos e a diminuição da possível acumulação desnecessária de resíduos são o principal foco do presente projeto. Desta forma, tem-se como objetivo verificar o estado atual de empresas que recolhem resíduos e estudar o melhor método para colmatar as debilidades identificadas no sistema, tornando-o eficiente e, consequentemente, reduzir os custos e a poluição.

De modo a atingir estes objetivos será implementado um sistema baseado em Inteligência Artificial capaz não só de prever a ocupação de cada contentor, mas também de determinar um trajeto otimizado (ou próximo da solução ótima) para a recolha dos diferentes contentores.

Pretende-se que o sistema seja autónomo na recolha do lixo, ou seja, sem a intervenção humana. Com base na previsão realizada é possível efetuar um planeamento semanal mais ajustado, otimizando recursos como a distância (minimizando o percurso para a recolha de lixo) e o combustível (consequente da otimização do percurso).

A recolha de resíduos urbanos pode ser considerada um problema geral que se divide em dois outros problemas:

- Previsão da capacidade utilizada dos contentores e determinação dos que necessitam de ser esvaziados;
- Cálculo de uma rota de boa qualidade para a recolha de todos os contentores necessários.

Primeiramente, para lidar com o problema da previsão dos recipientes cuja capacidade ocupada determina a necessidade da sua recolha irá ser utilizado um modelo de *Machine Learning* que, com base no histórico, irá prever a janela temporal para se realizar a recolha, ou seja, determinar a ocupação prevista para um determinado dia, de modo a que o contentor possua, no mínimo, 75% da sua capacidade utilizada, sem permitir a sua utilização completa (de modo a evitar resíduos no seu exterior). Numa segunda etapa, após identificação dos recipientes que necessitam de ser recolhidos, irá ser utilizado um algoritmo de determinação de rotas de modo a calcular pelo menos uma rota de boa qualidade que permita a recolha de todos os contentores anteriormente identificados. O problema de definição de rotas que considera a minimização da distância percorrida é definido na literatura como VRP (*Vehicle Routing Problem*). Os dois temas enunciados anteriormente, já foram alvo de um estudo exaustivo pela comunidade científica.

O principal contributo deste trabalho é o estudo e integração dos dois modelos, procedendo-se sempre que possível a uma comparação com os métodos exatos identificados na literatura.

1.3. Metodologia de Investigação

A metodologia utilizada neste trabalho incide sobre a metodologia *action-research* que descreve uma espiral de etapas que consistem no planeamento, ação e análise das ações realizadas. Na elaboração deste trabalho optou-se por esta metodologia, pois uma determinada ação pode não produzir os resultados pretendidos, sendo necessário regressar ao passo anterior de modo a rever futuros desenvolvimento. Na ocorrência de um progresso no desenvolvimento, existe também a fase de revisão para garantir que os resultados obtidos são válidos, ou seja, é necessário de testar e implementar várias soluções e por fim, selecionar a solução com o melhor desempenho sendo este um ponto-chave para o sucesso do projeto. Devido a este ser uma metodologia flexível a mudanças, torna-se adequada por permitir a realização constante de análises críticas e possíveis mudanças ao longo da implementação.

1.4. Cronograma de desenvolvimento

Para o sucesso da prova de conceito, o projeto está dividido em diversas fases, que serão descritas detalhadamente ao longo desta dissertação, tais como:

Atividades	**Janeiro Fevereiro**	**Março Abril**	**Maio Junho**	**Julho Agosto**	**Setembro Outubro**
Geração de dados	X				
Análise de dados	X				
Tratamento de dados	X				
Análise de modelos, métricas e parâmetros		X	X		
Treino do modelo		X	X		
Criação dos algoritmos de VRP		X	X		
Comparação das diferentes soluções de VRP		X	X		
Implementação de solução VRP				X	
Configuração do OSRM				X	X
Integração					X

1.5. Estrutura do documento

A presente dissertação está organizada por capítulos, a fim de tornar a sua leitura mais cómoda, acessível e dinâmica, para que todos os interessados consigam retirar os pontos essenciais mais facilmente.

No capítulo 1 é apresentada uma pequena introdução sobre o problema retratado na dissertação, assim como uma breve apresentação dos objetivos a alcançar com o trabalho desenvolvido. Além disto, apresenta-se também a motivação que orientou o desenvolvimento deste tema.

Por sua vez, no capítulo 2 é feita uma contextualização teórica sobre técnicas de *Machine Learning* e os seus complementos, onde se apresentam algumas noções teóricas utilizadas no contexto da dissertação, assim como investigações feitas sobre este tema. Neste capítulo é ainda apresentada uma breve revisão de literatura sobre o mesmo.

Seguidamente, no capítulo 3 é abordado o VRP onde são abordados alguns algoritmos (exatos e heurísticos) entre outros assuntos pertinentes à modelação de problemas. Aqui enquadra-se ainda a contextualização teórica sobre os referidos modelos.

No capítulo 4, localiza-se o desenvolvimento de todo o processo de criação do modelo de *Machine Learning*, ou seja, desde o momento da criação do seu *dataset*, aplicação do modelo e tratamento do mesmo.

Já no capítulo 5 é abordado o problema de planeamento de rotas de veículos, através da aplicação do algoritmo de VRP. São testadas as diversas abordagens utilizadas e é ainda realizada a comparação dos modelos exatos e heurísticos desenvolvidos. Por fim, é desenvolvida a implementação final da solução encontrada.

Relativamente ao capítulo 6 é demonstrada a solução no seu todo, ou seja, localiza-se aqui a integração de todas as implementações alusivas ao projeto, nomeadamente, a arquitetura e a configuração do OSRM (*Open Source Routing Machine)*.

O capítulo 7 expõe a conclusão da dissertação, onde são referidos os principais resultados obtidos e os objetivos atingidos, as características mais importantes do projeto, assim como as descobertas mais relevantes sobre a inteligência artificial aplicada à recolha de resíduos urbanos. Por último são apresentados alguns aspetos de melhoria para o projeto, nomeadamente, trabalhos futuros que fariam não só a solução ser mais robusta mas também mais polivalente.

Capítulo 2

2. Machine Learning

As técnicas de *Machine Learning* consistem na execução de algoritmos capazes de criar um modelo que represente conhecimento. Após a construção destes modelos é possível prever tendências futuras com base no histórico dos dados, combinando conceitos como a estatística, a probabilidade e a otimização [3]. Por outro lado, existem também algoritmos cujo objetivo é a realização de classificações [4], ou seja, determinar se os dados apresentados se incluem, ou não, numa determinada classe. Neste tipo de algoritmos destaca-se, por exemplo, a classificação de imagens onde se pretende determinar se um objeto existe ou não nessa imagem.

Devido à natureza deste problema, algoritmos associados a classificação, não foram considerados devido a não se adequarem a este tipo de problema (uma vez que requerem um valor numérico como *output*). Desta forma, todos os algoritmos utilizados nesta dissertação serão algoritmos supervisionados de regressão.

Apesar de todos os benefícios, é importante realçar que os modelos de *Machine Learning* são altamente dependentes da quantidade e qualidade dos dados e que requerem um grande poder computacional. Por este motivo, antes da sua execução é importante garantir que os atributos selecionados são suficientes para caracterizar o problema em análise e que os dados recolhidos (*dataset*) são representativos do caso em estudo.

2.1. Aprendizagem supervisionada

A aprendizagem supervisionada [5] é aplicada quando se tenta encontrar a relação entre as variáveis independentes e a variável dependente. Desta forma é necessário que os dados usados no treino contenham rótulos (sendo este categórico ou numérico), de modo a existir um significado para o mesmo. Um bom exemplo desta aprendizagem seria, neste caso, a existência de um *dataset* com diversas variáveis independentes e uma variável dependente do tipo numérica (ex. taxa de ocupação do contentor). No cenário anterior, o algoritmo iria aprender a classificar novos dados com base nos dados previamente fornecidos para o treino (ex. data, temperatura, cidade, entre outros).

2.1.1. Regressão Linear

A regressão é um dos métodos de previsão mais utilizados em estatística. Esta tem como objetivo verificar a influência de variáveis independentes numa variável dependente [6]

e criar um modelo matemático capaz de prever os valores da variável dependente com os valores das variáveis independentes.

A regressão pode ser de dois tipos: simples e múltipla. A principal diferença entre a regressão linear simples e múltipla é no número de variáveis preditivas. A primeira possui apenas uma, enquanto na segunda existem pelo menos duas. Por outro lado, sendo uma regressão linear, as variáveis preditivas possuem um comportamento linear.

2.1.1.1. Regressão Linear Simples

Por definição, este tipo de regressão [6] origina uma "linha reta, através do conjunto de pontos *n* de tal forma que as distâncias entre os pontos verticais do conjunto de dados e a linha reta são tão pequenas quanto possível". Assim, pode dizer-se que um modelo de regressão linear simples descreve uma relação entre duas variáveis quantitativas, em que uma é considerada uma variável independente e outra a variável dependente, ou seja, consiste apenas na representação da equação de uma reta, onde:

- Y representa o valor da variável dependente;
- X representa o valor da variável independente;
- m representa o declive da reta;
- b indica o valor de Y quando a variável X toma o valor zero.

$$(\mathbf{1})\ \boldsymbol{Y = b + mX}$$

Apesar de ser menos complexa, a regressão linear simples é útil, entre outros fatores, para:

- Perceber o nível de correlação entre duas variáveis;
- Criar modelos base como ponto de partida à inclusão de mais variáveis;
- Saber a equação da reta.

Colocando em prática a equação mencionada acima, um exemplo onde se aplica a regressão em análise é na análise do valor de um imóvel com base na sua área útil. A criação de uma regressão linear simples permite a obtenção de uma reta que estabelece a relação do preço do imóvel em função da área. Poder-se-á verificar que, normalmente, o valor das casas sobe consoante aumenta a sua área, mas, por outro lado, se a análise for realizada apenas para casas, o valor da constante *b* poderá ser relativamente pequeno. Então, no caso prático, X seria o valor da área em terreno, y o preço total da casa, m a relação entre a área da casa e o preço e *b* a constante.

2.1.1.2. Regressão Linear Múltipla

A regressão linear múltipla [7] é uma extensão do modelo de regressão linear simples, isto é, permite descrever uma relação entre um conjunto de variáveis quantitativas independentes e uma variável quantitativa dependente. Neste sentido, representa-se também por uma equação, no entanto, acrescentam-se mais variáveis que representam a relação entre as variáveis independes e a dependente, respetivamente.

A regressão linear múltipla, apesar de mais complexa, tem como objetivo:

- Prever o valor da variável dependente com base nas variáveis independentes;
- Perceber que variáveis são mais influentes para a previsão.

$$(2)\ \boldsymbol{Y} = \boldsymbol{b} + \boldsymbol{m_1 x_1} + \boldsymbol{m_2 x_2} + \cdots + \boldsymbol{m_n x_n}$$

Neste caso, e utilizando o exemplo prático de determinação de um valor de um imóvel, na regressão linear múltipla é possível utilizar vários parâmetros como, por exemplo, área útil, número de quartos, número de pisos, existência de garagem, entre outros. A determinação da reta deste exemplo permite definir qual a variável que possui um maior, ou menor, impacto no valor do imóvel, ou seja, quanto maior (ou menor) o valor de m_i mais (menos) relevante será essa variável na definição do preço final.

2.1.2. Rede Neuronal Artificial

A rede neuronal artificial consiste num modelo de *Machine Learning* cujo objetivo é aprender de forma igual ou semelhante a um cérebro humano. As redes neuronais são representadas como um conjunto de neurónios (multicamada), onde cada neurónio de uma camada possui uma ligação a todos os neurónios da camada seguinte, possuindo um conjunto de pesos que são ajustados durante o processo de adaptação/aprendizagem durante as várias épocas [8].

Estas redes conseguem resolver tanto problemas de classificação, como problemas de regressão, encontrando-se na última camada a diferença mais evidente neste processo.

Nos problemas de classificação, na última camada, o número de neurónios tem de ser igual ao número de classes do problema. Por outro lado, nos problemas de regressão, na última camada, é apenas utilizado um neurónio como resposta final, não necessitando da função *softmax*, sendo esta responsável por forçar a saída de uma rede neuronal a representar uma das classes (problemas de classificação) [9]. Sem esta função, o *output* da rede neuronal representa valores numéricos que apenas resolvem problemas de regressão.

Na Figura 3 apresenta-se uma representação simplificada de uma rede neuronal constituída por duas unidades (neurónios) na camada de entrada (*input layer*), três na camada oculta (*hidden layer*) e um na camada de saída *(output layer)*. Na representação considera-se que a rede neuronal recebe dois valores e que, sobre estes, irá realizar uma regressão de modo a determinar o valor na última camada. A camada intermédia permite a não linearidade entre variáveis e, por este motivo, permite à rede neuronal obter resultados de melhor qualidade face a modelos mais simples como a regressão linear simples/múltipla.

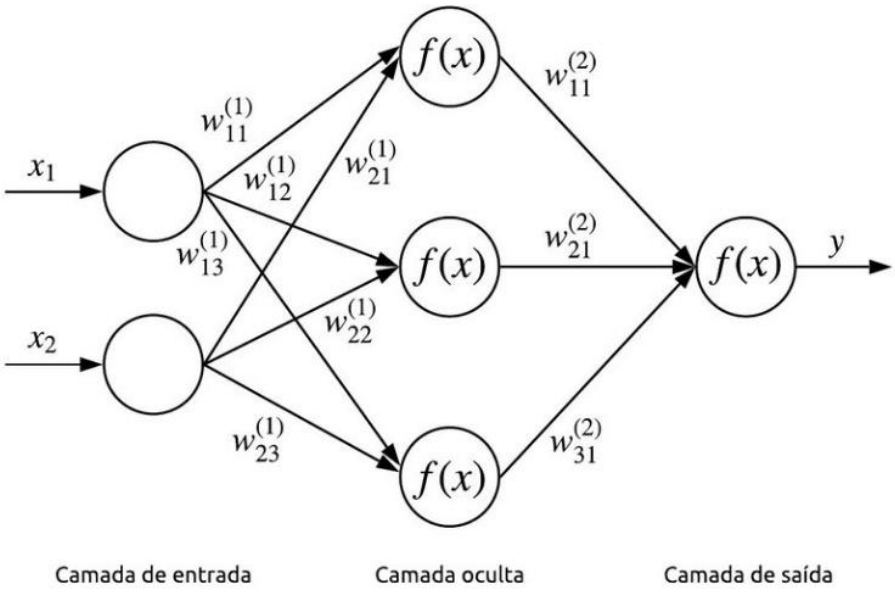


Figura 3 - Exemplo de uma Rede Neuronal

As redes neuronais oferecem várias vantagens tais como a habilidade de detetar relações não lineares e complexas entre as variáveis dependentes e independentes; grande poder de criar conhecimento e tolerante a variações, erros e ruídos. Por outro lado, as redes neuronais apresentam desvantagens, como a tendência de ficarem *overfitted*, a lentidão do seu treino e a sua natureza *black box* (tende a ter dificuldade a explicar os dados adquiridos de uma forma compreensível em "linguagem humana") [10].

2.1.3. Gradient Boosting

O *Gradient Boosting* [11] consiste numa técnica que pode ser usada em problemas de regressão. Esta técnica agrega vários modelos de previsão mais fracos na esperança de se tornar um modelo forte e resistente ao *overfitting.*

O algoritmo inicia com um valor constante (previsão inicial) e com intuito de prever, utiliza as árvores de decisão para efetuar operações na variável a fim de obter o valor que minimiza o erro.

Neste caso é necessário ter em conta que, normalmente, cada árvore tem um peso, logo o valor desta constante, de modo a ser alterado, consiste no valor que a árvore de decisão retorna a multiplicar pelo peso que a árvore de decisão tem para com o modelo total (figura 4).

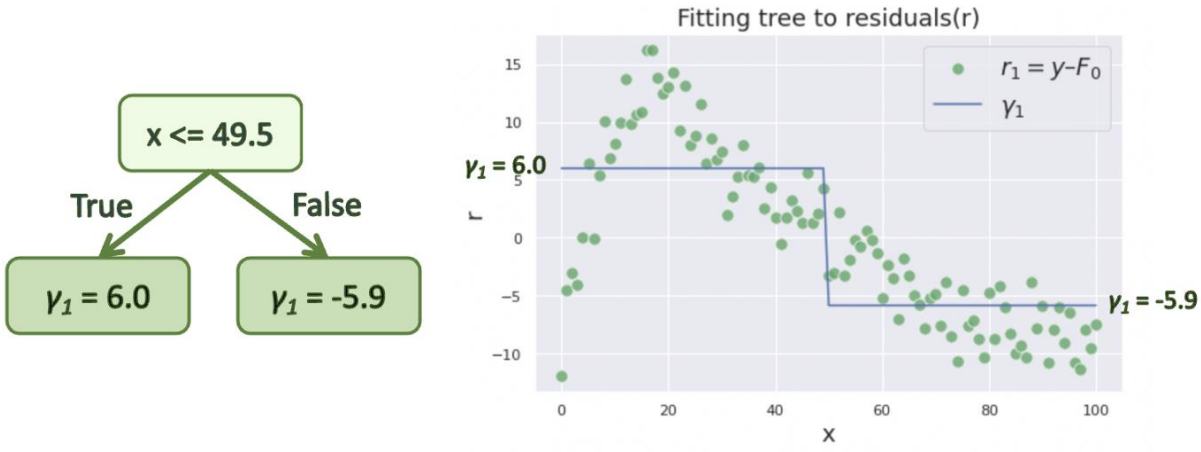


Figura 4 - Gradient Boosting (retirado de Towardsdatascience)

2.1.4. K-Nearest Neighbors

O *K-Nearest Neighbors* trata-se de um algoritmo que pode ser usado em problemas de regressão (figura 5). Este algoritmo [12] apoia-se na semelhança dos recursos do valor a prever com os valores já treinados. Por isso, no momento da previsão, ele irá mapear o novo dado, juntamente com os outros dados, para perceber o comportamento dos seus vizinhos mais próximos.

Na probabilidade de existir mais que um vizinho próximo, este escolhe os *n* pontos mais próximos com base na configuração e na distância atribui pesos diferentes, de forma a calcular o valor a ser previsto.

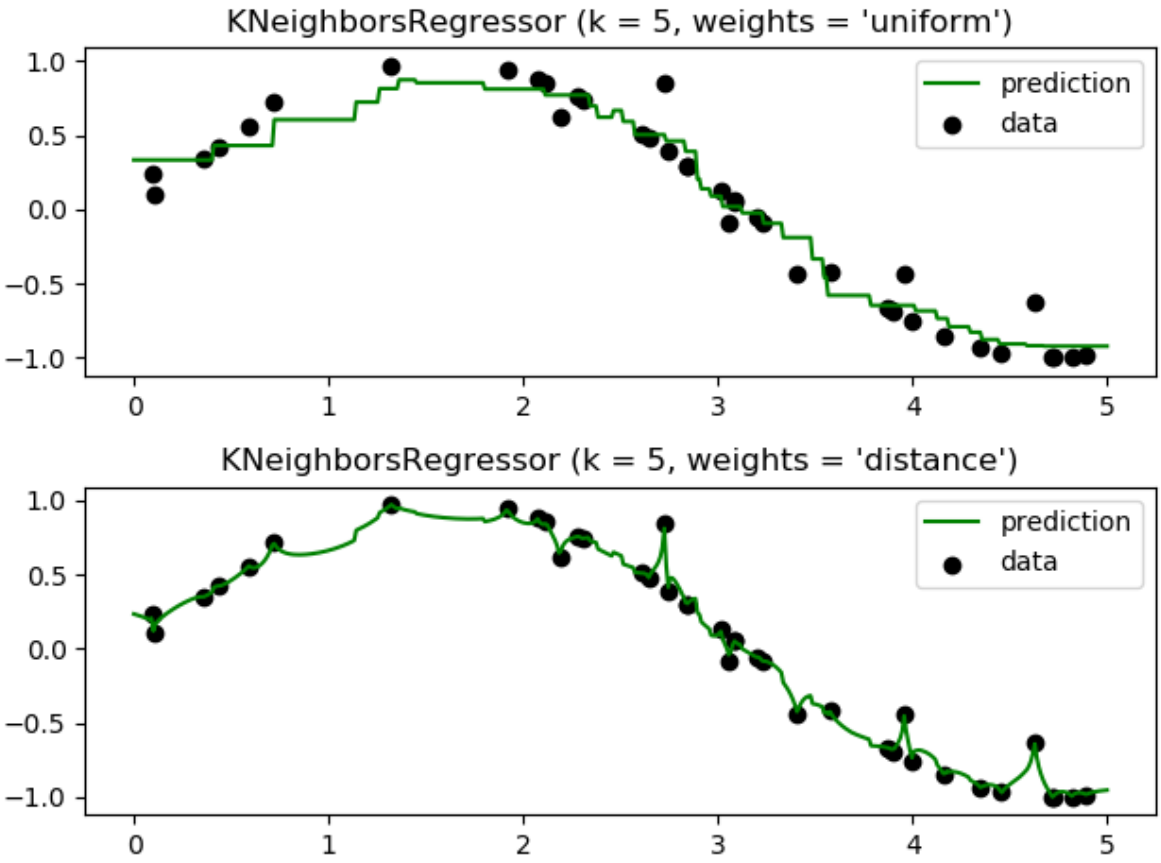


Figura 5 - Exemplo de KNN (retirado de Scikit-learn)

Existem várias funções para o cálculo da distância, tais como:

- **Distância Euclidiana:** consiste na raiz quadrada da soma das diferenças quadradas entre os dois pontos (ponto a prever e um ponto do modelo).

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \quad (3)$$

- **Distância de *Manhattan:*** consiste na distância real dos dois vetores usando a soma da diferença dos seus valores absolutos.

$$\sum_{i=1}^{k}|x_i - y_i| \quad (4)$$

- **Distância de *Hamming:*** apesar de usada apenas em variáveis categóricas, se o valor do ponto a prever for igual ao do vizinho, retorna 0; se não, retorna 1.

2.1.5. Métricas de classificação

De forma a se poder classificar qualquer modelo treinado é importante fazer uma escolha adequada das métricas a utilizar. Desta forma é possível fazer uma avaliação [13], a fim de descobrir que modelos e respetivos hiper-parâmetros são mais adequados ao caso de estudo para se poder avaliar o progresso do trabalho efetuado, tanto na aprendizagem do modelo, como para evitar problemas comuns como, por exemplo, o *overfitting* ou o *underfitting*. Para a avaliação dos modelos foram selecionadas as seguintes métricas:

- R-Squared;
- RMSE (*Root Mean Squared Error*);
- MAE (*Mean Average Error)*.

2.1.5.1. R-Squared

A R-Squared consiste numa métrica capaz de indicar a percentagem de variação dos dados explicada pelas variáveis independentes. Analisando um exemplo: se o valor de R-Squared (R^2) do modelo for de 0,92, então quer dizer que 92% da variação da variável dependente é explicada pelas variáveis independentes [14].

Este valor calcula-se pela subtração de 1 com uma divisão em que no numerador encontra-se o somatório do quadrado da diferença entre os valores previstos e o valor médio e no divisor encontra-se o somatório do quadrado da diferença entre os valores reais e o valor médio ao quadrado (tal como evidenciado na fórmula 5).

$$(5) \quad R^2 = 1 - \frac{\sum(Ypred - Ymean)^2}{\sum(Yactual - Ymean)^2}$$

Nesta métrica, quanto maior for o resultado melhor, dado que valores mais altos indicam um maior grau de explicabilidade da variável dependente.

O R-Squared pode ser interpretado [15], tal como sugere a tabela 1:

Tabela 1 - R-Squared, Grau de qualidade

Valor R-Squared	Grau de qualidade
0.75 - 1	Alto grau de explicabilidade
0.50 - 0.75	Bom grau de explicabilidade
0.25 - 0.50	Pouco grau de explicabilidade
0 – 0.25	Grau de explicabilidade quase nulo

2.1.5.2. MAE

O *Mean Average Error, em* português, designado como Erro Médio Absoluto [16], trata-se de uma métrica responsável por calcular a média do erro absoluto dos valores previstos com os valores reais, ou seja, aplica o somatório do módulo da diferença entre o valor previsto e o valor real. Nesta métrica quanto menor for o valor, melhor é o resultado.

$$(6) \quad MAE = \frac{1}{n}\sum_{j=1}^{n}\left|y_j - \hat{y}_j\right|$$

2.1.5.3. RMSE

O Root Mean Squared Error [16] também designada como raiz quadrada do quadrado do erro absoluto, caracteriza-se como uma métrica responsável por calcular a raiz quadrada do quadrado do erro absoluto dos valores previstos com os valores reais. Nesta métrica quanto menor for o valor melhor é o resultado.

$$(7) \quad RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

2.1.6. Overfitting e Underfitting

Dois problemas muito comuns na execução de algoritmos de *Machine Learning* são o *underfitting* e o *overfitting* [17]*.* Estes problemas são oriundos do treino do modelo.

Um modelo sofre *overfitting* quando este obtém resultados muito positivos nos seus dados de treino, mas os mesmos resultados não se revelam nos dados de teste. Deste modo considera-se que o modelo se adaptou demasiado bem aos dados para o treino e que este é incapaz de prever a situação em análise em qualquer outro tipo de dados. Logo, não é útil a

utilização deste tipo de modelo. Assim, torna-se possível afirmar que o modelo não tem capacidade de generalização e que a sua utilização é apenas útil nos dados com que treinou, pois, ao aplicar as regras identificadas noutros dados, este origina/apresenta um mau resultado.

Este problema pode ocorrer, por exemplo, se o número de iterações do algoritmo for demasiado elevado para o número de dados. O estado de *overfitting* pode ser observado no exemplo apresentado na figura 6.

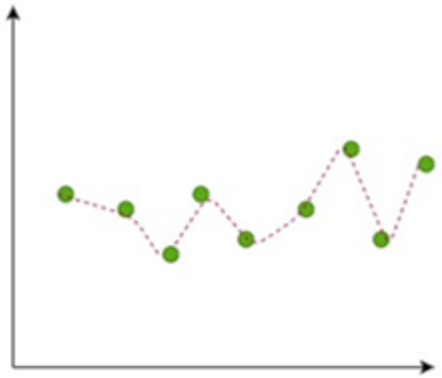


Figura 6 - Exemplo de Overfitting

Por outro lado, existe a possibilidade de existência de *underfitting* e neste, ao contrário do *overfitting,* não é necessário recorrer aos dados de treino para o detetar. O *underfitting* ocorre quando o modelo, apesar de ter sido treinado, não aprendeu as tendências necessárias para prever situações futuras. Isto significa que o modelo não conseguiu encontrar relações entre as variáveis independentes e a variável dependente e, por esse motivo, não consegue prever as situações futuras. Este problema pode ocorrer, por exemplo, se o número de iterações do algoritmo for demasiado baixo para o número de dados.

Na figura 7 encontra-se evidenciado um exemplo de *underfitting*:

Figura 7 - Exemplo de Underfitting

Um modelo com um bom desempenho consiste num modelo que não sofre de *overfitting,* nem de *underfitting,* isto é, trata-se de um modelo que encontrou as tendências necessárias nos dados de treino e, com isso, revelou um bom desempenho. No entanto, o seu desempenho não apresenta alterações quando entra em contacto com outros dados (dados de teste).

Apenas é possível detetar estes casos com recurso a métricas apropriadas a cada modelo.

2.2. Dados

Os dados são a principal razão para o sucesso ou o fracasso de um modelo [18], ficando estes com uma posição de relevo em todo o tipo de modelos criados através de *Machine Learning.*

É improvável a existência de um modelo com baixos erros preditivos se a qualidade dos dados não for boa. Para além disso, os dados, na sua maioria, necessitam quase sempre de tratamento. Este deve-se, por vezes, à necessidade de os adaptar ao modelo que se pretende treinar como, por exemplo, converter variáveis categóricas em numéricas. Por outro lado, existem dados que são trabalhados pelo utilizador de forma a melhorar os resultados dos elementos em análise como, por exemplo, a validação e remoção de *outliers* da base de dados ou tratamento de valores nulos.

Por estas razões, a análise dos dados e a preparação dos mesmos são fundamentais para um bom modelo.

Ao longo desta dissertação serão encontrados dois tipos de dados:

- **Dados categóricos:** dados que, como o nome indica, são responsáveis por caracterizar algo como, por exemplo, indicar um respetivo ID do contentor num caso de este ter um identificador;
- **Dados numéricos:** dados mais mensuráveis, que implicam um maior número de cálculos.

A preparação dos dados revela um papel fundamental também no sucesso de um modelo. Assim, ao longo deste projeto foram aplicadas as algumas técnicas, nomeadamente:

- **Alteração de escala dos dados**: na normalização [19] dos dados estes são convertidos para valores entre 0 e 1, de forma a igualar o efeito que todas as variáveis podem ter num modelo. Assim, caso se tenham duas variáveis onde uma apresenta uma gama de valores entre 1000 e 2000, pretende-se que a sua influência no modelo seja equivalente a outra variável cujos valores se encontrem entre, por exemplo, 10 e

15. No processo de normalização dos dados determinam-se, inicialmente, os valores máximo e mínimo e, posteriormente, a cada valor presente na base de dados dessa variável, é aplicada a fórmula apresentada na equação 8. No final do processo a variável normalizada apresenta valores entre 0 e 1.

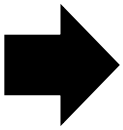
$$X_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (8)$$

- **Transformação dos dados**: caracteriza-se na conversão das variáveis categóricas em variáveis numéricas passíveis de serem utilizadas pelos modelos de *Machine Learning.* A técnica de *One-hot Encoding* [20] é uma possibilidade de utilização que consiste na transformação dos dados onde cada categoria é convertida numa nova variável.

  A título de exemplo, e analisando o exemplo apresentado na Figura 8, verifica-se que a base de dados possui uma coluna com identificação do sexo do utilizador. Esta coluna é transformada em duas colunas que consistem nas variáveis existentes na coluna original. Após isso, quando o valor da coluna sexo for igual ao valor das colunas criadas estas são preenchidas com o valor 1, e no caso de serem diferentes são preenchidas com o valor 0.

Original

Sexo
Masculino
Feminino
Masculino
Masculino



One-hot encoded

Sexo	Masculino	Feminino
Masculino	1	0
Feminino	0	1
Masculino	1	0
Masculino	1	0

Figura 8 - Exemplo de One-hot encoding

- **Limpeza de dados**: consiste na limpeza de linhas/colunas com valores em falta, valores nulos ou até valores sem sentido (por exemplo, uma taxa de ocupação negativa).

  Este tipo de tratamento pode envolver recurso a um modelo de *Machine Learning* para prever o valor a obter (como o KNN, *K-nearest neighbors*), ou pode ser realizado através da substituição do valor nulo pelo valor médio. Outras técnicas utilizadas

consistem na remoção de toda a coluna ou da linha do *dataset*, estando esta escolha limitada pelo número de elementos em falta.

2.3. Casos de aplicação na literatura

A previsão da quantidade de resíduos urbanos produzidos na Índia foi estudada por Jayaraman *et al.* (2021) [21], onde o objetivo era alertar para as implicações da produção excessiva de resíduos para o planeta e sensibilizar para a reutilização de objetos recicláveis. Para este estudo foram utilizados dois algoritmos, o *XGBoost* e o *Sarima.* De acordo com os resultados dos autores, *o XGBoost* apresenta melhores resultados com as definições em *default*. Para a avaliação da performance do algoritmo foram utilizadas as seguintes métricas: RMSE, MAPE (*Mean Absolute Percentage Error*), R2 Score e EVS. Por sua vez, para a avaliação e treino do modelo, o *dataset* foi dividido em 80/20, sendo que 80% do *dataset* foi utilizado para treino e os 20% foram utilizados para teste. Os mesmos autores propuseram como trabalho futuro a utilização de técnicas de *Deep Learning*, a fim de melhorar os resultados obtidos.

Em 2021, Yang *et al.* (2021) [22], analisaram vários algoritmos com o objetivo de prever os resíduos urbanos produzidos. Estes utilizaram as técnicas de *Cross Validation* e *Grid Search* em todos os modelos em análise, nomeadamente, SVM (*Support Vector Machines*), RF (*Random Forest*), *XGBoost*, KNN e DNN *(Deep Neural Network*). Para a avaliação do desempenho destes modelos recorreram a métricas como o R2, RMSE e MAE. Tendo em conta as métricas usadas, os modelos que revelaram melhor desempenho na resolução do problema foram o DNN e o RF, com resultados de R2 superiores a 0.9. Estes modelos conseguiram resolver o problema sem sofrer de *overfitting.* Por outro lado, o modelo SVR verificou-se como o menos capaz para a resolução do problema.

Segundo Kannangara *et al.* (2018) [23], os modelos como Árvores de Decisão ou Redes Neuronais ajustam-se melhor às necessidades deste tipo de problemas, quando comparados com modelos de regressão. Este ajuste de melhor qualidade deve-se ao comportamento não linear dos modelos. Os autores classificaram as redes neuronais como o melhor dos dois modelos, através da análise das métricas MSE (*Mean Squared Error*), R2 e MAPE conseguindo um resultado de 0.72 com a métrica R2.

Tendo em consideração os indicadores sócio-económicos foi realizado um estudo por Ceylan (2020) [24] para a previsão da produção de resíduos. No seu trabalho, Ceylan (2020) utilizou *Bayesian* GPR, MLR (*Multi-layer Regressor*) e *Bayesian* SVR tendo o modelo BGPR (*Bayesian Gradient Boosting Regressor)* mostrado maior desempenho com resultados de R2 superiores a 0.98. Este valor verificou-se tanto no *dataset* de treino como no de teste. Na

opinião do autor estes resultados devem-se ao facto de o GPR ser adequado a *datasets* pequenos.

Capítulo 3

3. Problema de planeamento de rotas de veículos

O problema de planeamento de rotas de veículos (VRP) é um problema de otimização combinatória, que se caracteriza em determinar uma lista de caminhos/ rotas/ trajetórias para percorrer todos os pontos necessários com o menor custo possível, ou seja, planear uma trajetória onde se realize o percurso no menor tempo e/ou no menor número de quilómetros [25].

Os problemas reais de planeamento de rotas de veículos [26] são caracterizados por diversas restrições de negócio, que dificultam a sua resolução tornando os VRP um dos problemas mais difíceis de otimização combinatória.

O problema de VRP é NP-difícil, tornando as soluções exatas muito demoradas nos casos em que a rota tem muitos pontos a visitar. Contudo, outros métodos não exatos (como, por exemplo, as heurísticas e/ou meta-heurísticas) são utilizados de forma alternativa que, apesar de não garantirem a solução ótima, são passíveis de serem executados em tempos computacionais menores. Apesar de não ser possível garantir que se encontra a solução ótima é possível obter soluções muito próximas desta.

Na Figura 8, apresenta-se o resultado da execução do VRP num caso demonstrativo. No exemplo, a linha contínua representa o melhor trajeto e a linha tracejada um trajeto que, sendo também possível de ser realizado, apresenta um maior custo financeiro (mais quilómetros percorridos).

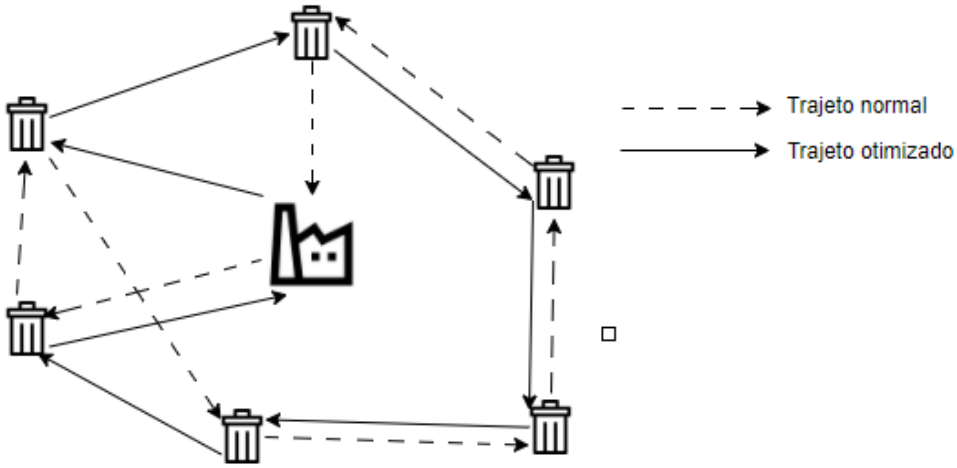


Figura 9 - Exemplo de um problema de VRP

Logo, o problema de planeamento de rotas de veículos caracteriza-se por ser, por si só, um problema, devido à necessidade de construção de uma ou várias rotas da forma mais otimizada possível atendendo a diversos objetivos, como por exemplo:

- Minimizar o custo (financeiro) do transporte, no seu todo;
- Minimizar o número de veículos necessários para o transporte;
- Minimizar o tempo total de viagem;
- Minimizar o número de condutores necessários;
- Minimizar as manutenções dos veículos;
- Aumentar a produtividade da recolha.

O VRP dependendo das restrições de cada problema pode assumir diferentes variantes, tornando-o num problema muito variado e complexo. Nesta dissertação serão abordados os problemas inerentes ao tema da recolha do lixo.

3.1. Capacitated VRP

O problema de planeamento de rotas de veículos com capacidade limitada [27] é uma variante do VRP quando os veículos têm uma limitação quanto à capacidade/ quantidade que podem e/ou devem transportar. Este tipo de problemas é recorrente em camiões de recolha/transporte de resíduos pois, por vezes, o seu depósito enche ou, então, em situações da indústria em que os camiões não devem exceder determinado peso de carga.

3.2. VRP Multi-Trip

O problema de planeamento de rotas de veículos [28] com múltiplas viagens é uma vertente que permite aos veículos fazer mais do que uma rota, ou seja, permite que estes sejam alocados inicialmente a múltiplas rotas. Podendo regressar ao ponto inicial para iniciar outra rota desde que o total de rotas realizadas cumpra com restrições como, por exemplo, distância total percorrida ou tempo total de viagem.

3.3. VRPPD

O VRPPD (*Vehicle Routing Problem with Pick-up and Delivering)* também designado por problema de planeamento de rotas de veículos com recolha e entrega [29] é uma vertente em que as mercadorias necessitam de ser movidas entre os locais de recolha e os locais de entrega. O objetivo desta vertente consiste em encontrar as rotas mais otimizadas, tendo em atenção o cumprimento das necessidades.

3.4. Modelos e Algoritmos para a resolução do VRP

O problema de VRP é do tipo NP-difícil devido ao esforço computacional requerido para a sua resolução crescer exponencialmente com a dimensão do problema, e cuja dimensão é caracterizada pelas suas condições e/ou número de pontos a percorrer. Sendo este um problema difícil de resolver, na maior parte das vezes, é tolerável e desejável, devido ao grande tempo computacional, que as soluções aproximadas se sobreponham sobre as soluções exatas, pois estas últimas sentem uma verdadeira dificuldade para dar resposta em tempo útil.

Por outro lado, as soluções aproximadas, garantem uma solução suficientemente boa, mas em tempos computacionais bem menores (quanto maior for a dificuldade do problema maior serão as diferenças em tempos computacionais).

3.4.1. Algoritmos exatos

Os algoritmos exatos [30] são métodos que procuram sempre a solução ótima, mas que têm um custo computacional de processamento muito maior em relação a outros métodos, sendo normalmente úteis em problemas com uma menor dimensão.

3.4.2. Heurísticas

As heurísticas [31] vieram dar resposta à inviabilidade das soluções exatas nas soluções mais complexas. Dado que a solução oferecida por estes algoritmos é devolvida num curto espaço de tempo computacional, continuando a ser uma solução boa para o problema e ajustada mais facilmente às restrições de negócio.

A título de exemplo, o algoritmo guloso [32], como heurística construtiva, constrói gradualmente uma solução possível, sendo uma técnica para resolver problemas de otimização. Este algoritmo caracteriza-se por em cada passo, escolher sempre a melhor opção no momento esperando que, ao proceder nesse caminho, a escolha leve até à melhor solução possível. Um exemplo típico é na seleção do ponto mais próximo (ou com menor custo) dando-se a rota como terminada quando não é possível adicionar mais pontos por não cumprimentos das restrições ou pela inexistência de mais pontos a visitar.

Trata-se de um algoritmo fácil de implementar e de rápida execução (devido a não necessitar de vários ciclos). Contudo, apresenta apenas uma desvantagem, nomeadamente, o facto de nem sempre conduzir à solução ótima ou a uma solução perto do ideal.

3.4.3. Meta-heurísticas

As meta-heurísticas [33] são soluções computacionais que definem um processo iterativo e evolutivo de gerações, resolvendo os problemas de forma genérica. Estas utilizam recursos como escolhas aleatórias e o conhecimento histórico de resultados anteriores para se adaptarem e mutarem, a fim de melhorar os seus resultados nas gerações/iterações seguintes.

Os algoritmos genéticos [34] são algoritmos evolutivos que usam técnicas inspiradas em biologia evolutiva como a hereditariedade, a mutação, a seleção natural e a recombinação.

Estes algoritmos são implementados por simulação sendo que a primeira geração é criada aleatoriamente. A evolução surge a partir da primeira geração para a segunda, da segunda para a terceira e assim sucessivamente. A cada geração é avaliada a sua qualidade, assim como também é feita a seleção dos melhores indivíduos para efetuar a mutação ou recombinação. Os indivíduos selecionados são sempre a entrada das iterações seguintes.

Um algoritmo genético não se baseia na otimização, mas sim na codificação das soluções possíveis. Estes algoritmos não necessitam de nenhum conhecimento do problema, apenas de uma forma que permita avaliar os resultados.

Este algoritmo necessita então de três funções essenciais ao seu bom funcionamento:

- **Uma função de mutação**: responsável por fazer mutações a uma solução explícita em busca de melhores resultados vindos dessas pequenas alterações. Consiste em mudar uma pequena parte da solução, na esperança de que esta melhore significativamente;
- **Uma função de seleção natural**: responsável por selecionar os melhores exemplares de cada geração, permitindo que apenas estes sobrevivam para a próxima geração. Normalmente, estes exemplares são utilizados para sofrerem mutações e recombinações;
- **Uma função de recombinação**: consiste em fazer uma combinação entre dois exemplares selecionados pela seleção natural, na expectativa de que os seus filhos resultem numa melhor solução.

Desta forma, abrangem o conceito de hereditariedade e é produzida uma nova solução com características dos dois exemplares selecionados.

3.5. Casos de aplicação na literatura

Park *et al.* (2021) [35], resolveram um problema D-VRP (*Dynamic Vehicle Routing Problem*) de entregas e recolhas recorrendo a um algoritmo genético. Devido ao percurso ser dinâmico este adaptava o caminho em tempo real utilizando uma estratégia de espera. Esta

estratégia atua como uma tomada de decisão, pois recalcular o caminho era uma tarefa computacionalmente difícil e nem sempre uma otimização para cada necessidade chegaria em tempo útil, sendo assim, esta estratégia decidia se valia a pena atender o pedido, ou se procedia com o resto da rota calculada anteriormente. Na opinião dos autores este método revela-se mais eficaz do que os métodos exatos, pois para casos em que o problema é grande, ou seja, o número de pontos pelo qual o veículo tem de passar é muito extenso, a solução exata pode necessitar de tempos de computação elevados onde o resultado ótimo pode não justificar esse tempo de espera, quando comparado com uma solução heurística de boa qualidade. O balanceamento entre tempo de execução e qualidade de solução é de extrema importância, uma vez que o VRP é um problema difícil de resolver (NP-Hard).

Erdelíc *et al.* (2019) [36] resolveram um problema E-VRPTW que consiste num problema de veículos elétricos com janelas de tempo, utilizando a meta-heurística r*uína-recreate* que consiste em eliminar parte de uma solução e reconstruí-la utilizando um algoritmo guloso. Comparam a solução desenvolvida por eles com um software comercial tendo tido resultados semelhantes.

Sai *et al.* (2020) [37], desenvolveram uma solução para recolha de resíduos com inclusão de dispositivos IOT (*Internet of Things*) cujo objetivo era identificar contentores com necessidade de recolha e posterior otimização de rotas. Os autores colocaram um sensor de proximidade (sensor de ultra-sons) para verificar o nível (altura) dos resíduos dentro do contentor e uma célula de carga para verificar o peso. Através dos dados recolhidos, identificavam os contentores a serem recolhidos primeiro. Para a resolução do problema de cálculo de rotas foi utilizado o algoritmo de *Dijkstra,* implementado em MATLAB, em que o peso entre os nodos era decidido baseado na prioridade de recolha dos contentores.

Jairo R. Montoya-Torres *et.al. (2015)* [38], numa revisão na literatura de VRPMD (*Vehicle Routing Problem Multiple Depot*), evidenciam a importância da distribuição física no comércio, de forma que os produtos fiquem disponíveis para os clientes. Por este motivo, os mesmo salientam que resolver o problema de VRP, de forma eficiente e eficaz é imprescindível para o sucesso de qualquer empresa. O trabalho realizado assentava numa revisão de literatura de 173 documentos, desde revistas, conferências, teses e livros, e cerca de 84.44% dos documentos estudados consideravam única e exclusivamente um objetivo de otimização, enquanto que a restante percentagem considerava múltiplos objetivos como, por exemplo, a distância mínima percorrida e ainda o mínimo uso possível de veículos. Ainda associado ao artigo dos mesmos autores, nele foram abordados três tipos distintos de soluções para o problema: o método exato, as meta-heurísticas e as heurísticas. O aumento do número de artigos publicados sobre o tema do VRP tem vindo a aumentar, uma vez que a indústria também tem aumentado, verificando-se que, e de acordo com as informações descritas, apenas seis artigos foram publicados entre 1980 e 1990, no entanto, entre 2000 e

2010 foram publicados mais de 70 artigos. O principal motivo que despoletou a escrita destes artigos e o seu desenvolvimento foi sobre a distância, o custo ou o tempo em estudo, isto é, cerca de mais de 80% dos artigos publicados eram sobre estes temas, enquanto o restante abordava questões de número de veículos (quantidade) / balanceamento de cargas (peso). No que diz respeito à distribuição das soluções, 42% delas foram apresentadas recorrendo a meta-heurísticas, 25% foram apresentadas através de métodos exatos e 33% recorrendo às heurísticas. Alguns exemplos de meta-heurísticas foram realizados fazendo recurso a algoritmos genéticos ou a pesquisa tabu. Em múltiplos objetivos os autores consideraram que, primeiramente, deve focar-se num objetivo e otimizá-lo ao máximo e, posteriormente, passar para os outros de forma sequencial, mantendo sempre esta lógica em todo o processo.

Capítulo 4

4. Modelo de Machine Learning

Neste capítulo são apresentadas as fases do projeto relacionadas com a criação do *dataset*, análise e tratamento de dados, criação e validação do modelo de *Machine Learning* utilizado na previsão de acumulação de resíduos nos diversos tipos de contentores.

4.1. Criação do dataset

Uma vez que não foram encontrados *datasets* disponíveis na literatura que representassem de forma real o problema abordado, foi necessário gerar um *dataset*. Para aproximar o tipo de variáveis necessários à realidade do problema, foram estabelecidos contactos com empresas que lidam diariamente com esta problemática e foi feito um trabalho de observação para uma recolha aproximda dos dados.

Desta forma, foi adotado um sistema de três tipos distintos de contentores. Em particular, considerou-se que um contentor ficava com a sua capacidade totalmente utilizada em três dias, outro em dois dias e finalmente um contentor cuja capacidade se esgotava em apenas um dia.

De forma a tornar o *dataset* mais ajustado à realidade, foi introduzida alguma lógica ao mesmo como, por exemplo, diminuir ligeiramente as quantidades de resíduos no mês de agosto (uma vez que, normalmente, são períodos de praia onde a maior parte das pessoas deixa a localidade de residência e vai para outro lugar), e aumentar de forma significativa a produção de resíduos no período de Natal e Ano Novo (onde se juntam as famílias ao longo das comemorações, produzindo expressivamente mais lixo).

O *dataset* criado é composto por nove variáveis independentes e uma variável dependente. As variáveis utilizadas são apresentadas na tabela 3, sendo indicado o tipo de variável e o valor mínimo e máximo considerado. As variáveis “Ocupação” representam o peso em quilogramas (Kg) que cada contentor tem armazenado e a variável “População” representa o número de residentes na cidade. A variável “Nome” representa o código do contentor, ou seja, a sua identificação.

O tipo de contentores neste *dataset* foi equilibrado, sendo que para a reprodução deste foram criadas quatro instâncias de contentores que enchem rapidamente (um dia para ficarem cheios), seis instâncias de contentores que enchem de forma moderada (dois dias para ficar cheios) e, por fim, três instâncias de contentores que necessitam de três dias para ficarem cheios na sua totalidade perfazendo um total de treze contentores em estudo.

Todos os valores das ocupações diárias têm variações, devido ao *random* que é aplicado, sendo que, então:

- Para contentores que enchem rápido, os seus valores podem variar entre 85% de ocupação e 105% de ocupação (casos de transbordo);
- Para contentores que enchem de forma moderada, os seus valores podem variar entre os 45% e 70%;
- Para contentores que enchem de uma forma mais lenta, os seus valores podem variar entre os 10% e 25%.

Para além disto, outros fatores são influenciadores da ocupação dos contentores, tais como:

- O tamanho da população que:
  - no caso de ser menor ou igual a 9000, não existe alteração na ocupação dos contentores;
  - no caso de ser maior que a 9000, a ocupação aumenta entre 1 e 3%;
  - no caso de ser maior ou igual a 10500, a ocupação aumenta entre 2 e 4%.
- A precipitação, em caso de real, origina um decréscimo de 5 a 10%;
- O dia seguinte ao dia de Natal, assim como o dia seguinte ao Ano Novo aumenta a ocupação entre 5 e 10%;
- Em agosto, os dias têm uma variação na ocupação entre menos 5% e mais 5%;
- O dia da semana, pois dependendo do dia da semana os valores da ocupação de lixo podem ou não aumentar, dado que a este valor, não se aplicou um valor aleatório, mas sim um valor concreto, como se apresenta na tabela 2:

Tabela 2 - Interferência dos dias da semana nos resíduos urbanos

Dia da semana	Interferência no valor
Segunda-feira	-2%
Terça-feira	+3%
Quarta-feira	+1%
Quinta-feira	0%
Sexta-feira	1%
Sábado	-2%
Domingo	+5%

A necessidade de existirem valores aleatórios deve-se ao facto de haver necessidade do futuro modelo não sofrer de *overfitting,* mas também para simular, dentro do possível, um caso real, uma vez que aqui ainda existem vários fatores que poderiam influenciar os resultados e não são apresentados.

Tabela 3 - Descrição do dataset

Categoria	Nome da Variável	Tipo de Variável	Valor mínimo	Valor máximo
Independente	Ocupação	Numérica	3	114
Dependente	Dia da semana	Categórica	1	7
	Dia	Numérica	1	31
	Mês	Numérica	1	12
	Ano	Numérica	2015	2021
	Chuva	Categórica	0	1
	População	Numérica	9000	11500
	Latitude	Numérica	≈ -8.41	≈ -8.43
	Longitude	Numérica	≈ 41.21	≈ 41.23

4.2. Análise do dataset

Relativamente à base de dados utilizada nesta investigação, importa salientar que alguns dos fatores foram alvo de análise, nomeadamente, a taxa de ocupação dos diferentes contentores respetivos aos resíduos urbanos, tendo esta sido analisada em diferentes perspetivas: máximos, mínimos e média. Por sua vez, o facto de ocorrer precipitação (ou não)

e a sua implicação na variação da taxa de ocupação dos contentores também foi um elemento alvo de análise. Para além disso, ainda se analisou a taxa de ocupação de acordo com os diferentes meses (janeiro a dezembro) e, dias da semana (segunda-feira a domingo), com intuito de se verificar o comportamento da taxa de ocupação, atendendo aos diferentes condicionantes e parâmetros.

Todos os dados mencionados foram analisados recorrendo à ferramenta Microsoft Power BI. Na imagem seguinte (figura 10), verifica-se a taxa de ocupação média em relação aos vários meses do ano. Aqui é possível verificar que as quantidades de lixo são muito similares em todos os meses com exceção do mês de agosto. Este deve-se ao facto de, tal como descrito anteriormente, haver uma atenção no algoritmo de geração do *dataset* para simular situações como férias. Também se pode verificar neste gráfico a interferência da chuva. Em dias de chuva é expectável que muitas pessoas prefiram aguardar para deitar os seus resíduos no dia seguinte. Essa influência foi possível verificar, uma vez que quando chove os valores da taxa de ocupação dos contentores eram sensivelmente mais baixos do que em dias não chuvosos.

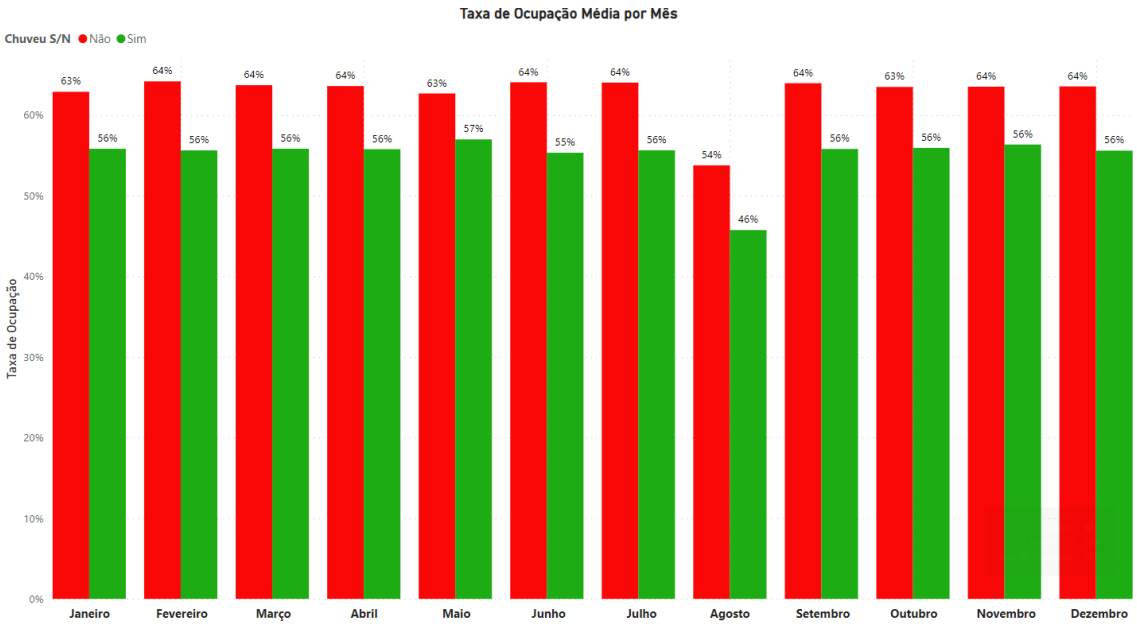


Figura 10 - Taxa de ocupação média, por mês

Na figura 11 verifica-se a taxa de ocupação nos diferentes tipos de contentores. Como foi mencionado, existem três tipos distintos de contentores, ou seja, contentores que ficam cheio na sua totalidade de forma rápida, outros de forma moderada e outros de forma lenta. No gráfico apresentado verifica-se que isso foi conseguido devido ao facto de ser possível perceber com exatidão estes três diferentes grupos.

Os contentores identificados com a letra A, B, C e D correspondem ao primeiro tipo, ou seja, ficam cheio na sua totalidade de forma rápida; por sua vez, os contentores designados por E, F, G, H, I e J correspondem aos contentores que enchem de forma moderada; e os últimos contentores, nomeadamente, K, L e M caracterizam-se por serem contentores com a taxa de ocupação mais lenta.

Com estes gráficos também é possível perceber que o dia da semana tem influência na ocupação, verificando que o domingo e a terça-feira são os dias em que mais há despejo de lixo e a segunda-feira e o sábado correspondem aos dias em que há menos despejos.

Neste gráfico é possível determinar quais as ocupações mínimas, médias e máximas de cada contentor, sendo que os contentores rápidos são os únicos que apresentam transbordo. Nos contentores lentos, o máximo de capacidade que estes conseguem atingir é, em média, metade da sua capacidade.

Apesar deste *dataset* ter sido criado é possível verificar um cuidado na sua criação, a fim de simular a realidade da maneira mais próxima possível.

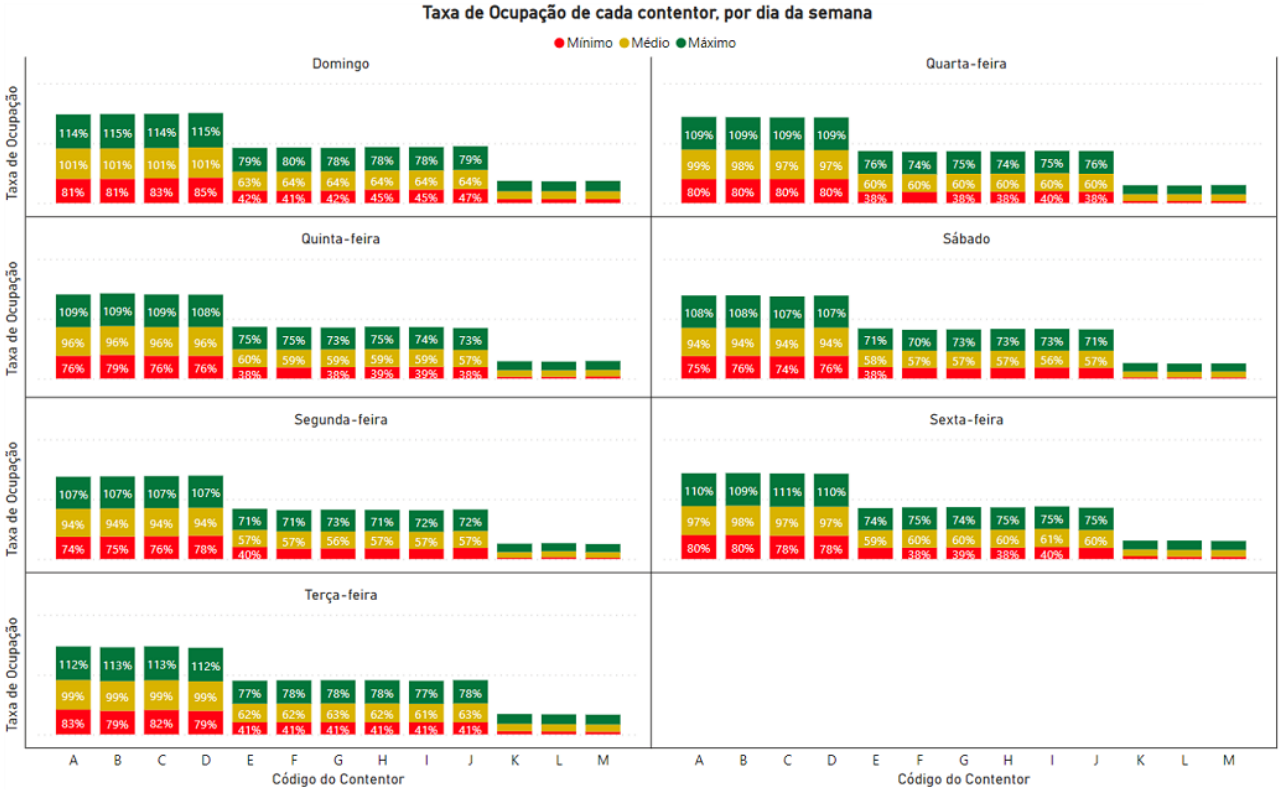


Figura 11 - Taxa de ocupação, por contentor e por dia da semana

4.3. Tratamento do *dataset*

Apesar de todos os dados expostos terem sido criados por um algoritmo, a criação do mesmo foi efetuada da forma mais realista possível. No entanto, o algoritmo de criação de dados originou dados com algumas inconsistências, nomeadamente, taxas de ocupação negativas, e/ou dados difíceis de analisar, através dos modelos matemáticos.

De salientar que foram tidos em conta determinados fatores para a melhoria e análise dos dados presentes no *dataset* e, consequentemente, para a obtenção de melhores resultados dos modelos. Assim, os fatores considerados foram:

- Dados categóricos;
- Dados em falta/ *outliers;*
- Normalização.

4.3.1. Dados categóricos

Tal como descrito no capítulo 2, um modelo de *Machine Learning* tem dificuldade em utilizar variáveis categóricas. Desta forma, uma das técnicas mais utilizadas para resolver este tipo de problemas é o *One-hot Encoding,* que consiste em multiplicar o número de colunas pelo número de opções distintas e categorizar com valores booleanos.

No *dataset* em questão existia a necessidade de efetuar essas modificações nas variáveis “Dia da semana” e “Identificador do contentor”, sendo que a nomenclatura utilizada para as colunas atualizadas foi a seguinte:

- (nome da coluna)_(valor da variável)

Neste caso e colocando em prática através de um exemplo, as colunas com o identificador da variável ficou com valores ID_A, ID_B, ID_C e assim, sucessivamente. Onde, no caso de a linha correspondente ser o ID_A, esta coluna estaria preenchida com 1 e as restantes colunas que começam por “ID” seriam preenchidas por 0.

4.3.2. Dados em falta / *Outliers*

O *dataset* continha dados em falta, ou seja, existiam linhas com valores em falta (em branco), sendo que estas linhas continham apenas uma ou múltiplas falhas.

Apesar de haver soluções como, por exemplo, preencher com a média ou recorrendo à utilização de outros modelos de *Machine Learning* para preencher valores omissos, a solução escolhida acabou por ser a remoção das linhas que continham este tipo de problemas.

No que diz respeito a *outliers,* a mesma solução foi equacionada e escolhida, ou seja, foi feita também a remoção de todas as linhas, pois estes valores por vezes representavam

valores muito fora da escala, o que iria prejudicar a análise dos dados no modelo, por outro lado, também foram eliminados valores que não faziam sentido para o *dataset* como, por exemplo, taxas de ocupação negativa.

4.3.3. Normalização

Nas redes neuronais a normalização torna-se crucial. É possível que o modelo, acabe por não aprender, no caso deste não estar normalizado, isto porque estas podem ficar retidas e não conseguirem produzir as relações para formarem conhecimento, e por fim (futuramente) conseguirem prever.

Devido a esta característica das redes neuronais, todas as colunas do *dataset* passaram por um processo de normalização, ou seja, os seus valores ficaram compreendidos num raio entre 0 e 1.

Com esta alteração, o desempenho da solução foi melhorado percentualmente, de acordo com a métrica R2.

4.3.4. Análise de modelos, métricas e parâmetros

Como identificado e descrito na secção 2.1, existem diversas abordagens e algoritmos para a previsão de problemas de regressão.

De seguida, irão ser mencionadas todas as abordagens utilizadas e testadas no desenvolvimento do melhor modelo possível. Nesta fase será ainda descrito como o algoritmo final foi selecionado, e mencionados todos os critérios usados para a seleção do mesmo. Os testes realizados para a seleção serviram como decisão para a implementação da solução final do problema.

Para todos os algoritmos testados, o *dataset* sofreu sempre o mesmo tratamento, ou seja, as variáveis categóricas foram transformadas com o *One-hot Encoding*, todas as colunas foram normalizadas e os valores irrealistas sofreram eliminação, não sendo, por isso, utilizados na previsão do modelo, ou seja, foram eliminados valores em que a taxa de ocupação era negativa, dado que isto é impossível de acontecer.

Todos os algoritmos de *Machine Learning* foram também executados na mesma máquina e no mesmo ambiente, evitando que outros fatores externos fossem capazes de influenciar os resultados obtidos.

As especificações da máquina de testes são as seguintes:

- CPU: Ryzen 3600 Hexa Core;
- RAM: 16GB RAM 3200mhz;
- SSD: 512GB;
- OS: Windows 11 Pro.

Apesar de existirem várias bibliotecas na linguagem *python* sobre *Machine Learning* como *keras ou pytorch* a escolha para todos os testes e para a implementação do modelo recaiu sobre o *sklearn.*

O *Scikit-Learn* é um módulo *python open-source* de fácil utilização e com variadas ferramentas de ajuda. Desde que os dados estejam organizados e corretamente formatados, o processo para colocar a correr qualquer algoritmo de classificação/ regressão ou *clustering* para efetuar o modelo é relativamente semelhante. Esta é também uma biblioteca capaz de resolver problemas de aprendizagem supervisionada, não supervisionada e aprendizagem por reforço.

Um dos principais objetivos quando se está a fazer um projeto de *Machine Learning* é obter a maior rentabilidade. Em função dos algoritmos é necessário selecionar as colunas a utilizar, mas também os melhores parâmetros, de modo a personalizar o modelo da melhor forma possível, isto é, alcançando o melhor desempenho.

Não há como saber qual o melhor parâmetro/ configuração para retirar o maior partido do modelo que está a usar utilizado, admitindo-se mesmo que a melhor forma seria a de tentativa e erro, ou seja, experimentar todas as combinações possíveis (ou pelo menos as que fizessem mais sentido) para os parâmetros de cada modelo em questão. Assim, foi usado o *GridSearchCV,* neste projeto de forma a tentar extrair o máximo de desempenho possível sobre cada modelo.

Para além disto o *Scikit-Learn* contém uma função chamada *RandomSeachCV* que cria soluções aleatórias na procura do melhor parâmetro possível. Sendo este um método mais aleatório, depende dos valores aleatórios atribuídos, podendo não levar a um bom resultado. Por essa razão este método não foi considerado para este trabalho.

4.3.5. GridSearchCV

O *GridSearchCV* [39] consiste numa função fornecida pela biblioteca *Scikit Learn,* onde dado um conjunto de parâmetros previamente definidos, esta testa todas as combinações possíveis.

Após fazer o teste para todas as combinações, todos os modelos originados pela função são avaliados e, esta função retorna o resultado obtido pelo melhor modelo, assim como os parâmetros utilizados. Desta forma, em vez de ser requerido ao programador que teste todas as combinações, este tem uma função capaz de o fazer.

Apesar de esta função ser muito útil e facilitar na escolha dos melhores parâmetros, pois não requer ao programador que faça uso do método tentativa e erro, ela é computacionalmente difícil de executar, isto porque, se um modelo de *Machine Learning* necessita normalmente de um grande poder computacional devido às suas características

inerentes, é compreensível que o teste de várias possibilidades para os parâmetros seja um processo demorado na sua execução. Assim, se esta função:

- Contiver 4 parâmetros;
- Se cada parâmetro tiver 4 valores a testar;
- Existem 256 = 4^4 combinações para testar;

Esse valor é ainda aumentado em *n* vezes o número escolhido para a validação cruzada, ou seja, se for escolhido 4, o algoritmo teria de ser executado 1024 vezes.

A função *GridSearchCV* contém parâmetros que a tornam configurável e, apesar de existirem vários, as únicas configurações existentes que foram utilizadas para este projeto foram as seguintes:

- **param_grid:** consiste num dicionário que contém como chave: o nome dos parâmetros; e como valor: as definições correspondentes;
- **n_jobs:** representa o número de núcleos utilizados por esta função, neste caso foi sempre selecionado "-1", que permite que use todos os núcleos existentes de forma a efetuar os testes o mais rápido possível;
- **estimator:** o algoritmo de *Machine Learning* escolhido para ser testado e efetuar o modelo;
- **cv:** número de metades usadas para a validação cruzada, que no caso foi selecionado 4 em vez do 5 por *default,* de forma a diminuir o número de iterações;
- **verbose:** nível de verbosidade disponível ao longo da execução, selecionado como 3.

4.3.6. Treino dos modelos

Tal como referido anteriormente, todos os modelos foram previamente avaliados utilizando o *GridSearch*, exceto a regressão linear.

Esta foi a maneira menos enviesada e a forma mais eficiente de procurar o melhor modelo que se ajustasse à melhor solução pretendida, de forma a que esta fosse o mais eficiente e, principalmente, o mais precisa possível.

É necessário ter em consideração que no caso de uma má previsão, no sentido em que se prevê que não existem resíduos no contentor e, no entanto, este está em excesso de ocupação é muito penoso para o bem-estar comum de qualquer habitante de uma cidade e a existência frequente deste acontecimento invalidava de todo a solução proposta.

Posto isto, considera-se que para esta solução em caso de erro, é muito menos prejudicial prever um contentor vazio como um contentor cheio, do que ao contrário, dado que a primeira situação apenas causaria um prejuízo em termos monetários enquanto a outra, tal como referido anteriormente, pode ter implicações no bem-estar e na saúde da população abrangente dessa localização.

4.3.7. Multi-layer Perceptron Regressor

O *Multi-layer Perceptron Regressor* [40] é uma das implementações de redes neuronais que o *Scikit Learn* oferece. Neste algoritmo é permitido efetuar alterações em variados parâmetros, no entanto, os parâmetros a serem testados pela função *GridSearch* foram os seguintes:

- *hidden_layer_sizes;*
- *activation;*
- *solver;*
- *alpha;*
- *learning_rate;*

O pârametro *hidden_layer_sizes* é responsável por definir o número de neurónios na camada oculta. Por outro lado, o parâmetro *activation* é responsável pela função de ativação para a camada oculta, neste caso, esta pode ser definida de quatro formas diferentes, sendo elas as seguintes:

- ***Identity*:** sem função de ativação:

$$(9)\ f(x) = x$$

- ***Logistic*:** uso da função sigmoide:

$$(10)\ f(x) = \frac{1}{(1 + \exp(-x))}$$

- **Tanh:** uso da função tangente hiperbólica

$$(11)\ f(x) = \tanh(x)$$

- **Relu:** uso da função de unidade linear retificada

$$(12) \quad f(x) = \max\,(0, x)$$

Por outro lado, há o parâmetro *solver* que é responsável por escolher, o solver para a otimização dos pesos, tendo este três possibilidades, nomeadamente:

- **Lbfgs:** optimizador na família de métodos *Quasi-Newton*;
- **Sgd:** optimizador de descida de gradiente estocástica;
- **Adam:** optimizador de descida de gradiente estocástica (diferente);

O parâmetro *alpha* refere-se ao tamanho do regularizador e corresponde a um valor decimal.

Já o parâmetro *learning_rate* corresponde à velocidade de aprendizagem da rede, ou seja, à velocidade com que os pesos são atualizados. Este parâmetro tem três possibilidades, sendo elas:

- **Constant:** em que o rácio de aprendizagem é sempre constante;
- **Invscaling**: em que reduz gradualmente o rácio de aprendizagem;
- **Adaptive:** mantém o rácio de aprendizagem constante, desde que o erro continue a diminuir. De cada vez que duas épocas falhem a diminuição de erro, o rácio é dividido por 5.

Da análise das configurações possíveis para serem selecionados e alteradas realizou-se a configuração apresentada na Figura 12 para o treino e validação do modelo.

```
{
"hidden_layer_sizes": [(1,),(10,),(30,),(50,)],
"activation": ["identity", "logistic", "tanh", "relu"],
"solver": ["lbfgs", "sgd", "adam"],
"alpha": [0.00005,0.0005,0.0001],
"learning_rate": ["constant","invscaling","adaptive"]
}
```

Figura 12 - Parâmetros testados no MLP Regressor

Após a execução do método de procura das melhores configurações, obteve-se como resultado da função de *GridSearch* o resultado apresentado na Figura 12, ou seja, o melhor modelo possui a função de ativação *logistic,* o parâmetro *alpha* com o valor de 0.0005, o *hidden_layer_sizes* com valor de 50, a *learning_rate* como *adaptive* e o solver escolhido o sgd.

```
The best parameters across ALL searched params:
{'activation': 'logistic', 'alpha': 0.0005, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'sgd'}
R2 - Train : 0.9477
R2 - Test : 0.9419
MSE - Train : 0.0244
MSE - Test : 0.0287
```

Figura 13 - Output da função GridSearch na execução do MLP Regressor

Para este conjunto de parâmetros foram verificadas várias métricas de avaliação ao modelo, obtendo-se um:

- R2 score
  - Dados de treino ≃ 0.948;
  - Dados de teste ≃ 0.942.

- MSE
  - Dados de treino ≃ 0.024;
  - Dados de teste ≃ 0.029.

4.3.8. Gradient Boosting Regressor

O *Gradient Boosting Regressor* [41] é uma das implementações *Gradient Boost* que o *Scikit Learn* oferece. Neste algoritmo é permitido efetuar alterações em variados parâmetros.

Contudo, os parâmetros a serem testados pela função *GridSearch* foram os seguintes:

- *learning_rate;*
- *subsample;*
- *n_estimators;*
- *max_depth.*

Primeiramente, o parâmetro *learning_rate* define a contribuição de cada modelo fraco para a aprendizagem. Em segundo lugar, o parâmetro *subsample* define a fração de amostras utilizadas para ajustar a aprendizagem. Em terceiro lugar, o parâmetro *n_estimators* define o número de fases de reforço a serem executadas. Por outras palavras define o número de árvores de decisão. Por último, o parâmetro *max_depth* define a profundidade máxima de cada árvore, limitando assim os seus nós.

As configurações possíveis de serem selecionadas e alteradas estão representadas na Figura 14 para o treino e validação do modelo.

```
{
  'learning_rate': [0.01,0.02,0.04],
  'subsample': [0.9, 0.5, 0.2],
  'n_estimators': [100,500,1000, 1500],
  'max_depth': [4,6,8,10]
 }
```

Figura 14 - Parâmetros testados no GB Regressor

Após nova execução do método de procura das melhores configurações neste caso para o algoritmo GBR (*Gradient Boosting Regressor),* obteve-se o resultado apresentado na Figura 15.

```
The best parameters across ALL searched params:
{'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 1000, 'subsample': 0.5}
R2 - Train : 0.9453
R2 - Test : 0.9449
MSE - Train : 0.0259
MSE - Test : 0.0275
```

Figura 15 - Output da função GridSearch na execução do GB Regressor

Assim, verifica-se que a melhor configuração dos parâmetros, ou seja, a que permite obter um modelo de melhor qualidade é a seguinte:

- *learning_rate* → 0.01;
- *max_depth* → 4;
- *n_estimators* → 1000;
- *subsample* → 0.5.

Para este conjunto de parâmetros obtiveram-se, como resultado das métricas de avaliação ao modelo, os seguintes valores:

- R2 score
  - Dados de treino ≃ 0.945;
  - Dados de teste ≃ 0.945.
- MSE
  - Dados de treino ≃ 0.026;
  - Dados de teste ≃ 0.027.

4.3.9. K-Neighbors Regressor

O *K-Neighbors Regressor* [42] consiste na implementação de *K-Nearest Neighbors* que o *Scikit Learn* oferece orientado à regressão. Neste algoritmo é permitido efetuar alterações em variados parâmetros. No entanto, os parâmetros a serem testados pela função *GridSearch* foram os seguintes:

- *n_neighbors;*
- *algorithm;*
- *weights;*

O parâmetro *n_neighbors* representa o número de vizinhos que serão usados pelas *k-neigjbors queries.* O parâmetro *algorithm* corresponde ao algoritmo utilizado para efetuar a computação do vizinho mais próximo. Este parâmetro tem três possibilidades, designadamente:

- **ball_tree:** utiliza o algoritmo *BallTree;*
- **kd_tree:** utiliza o algoritmo *KDTree;*
- **brute:** utiliza a procura na forma de "força-bruta".

O parâmetro *weights* corresponde à função de peso usada no treino do modelo. Este parâmetro pode assumir dois valores distintos:

- **uniform:** pesos uniformes, todos os pontos vizinhos são ponderados igualmente
- **distance**: pesos consoante a sua distância, neste caso, vizinhos mais próximos tem uma influência maior sobre vizinhos mais distantes

Após isto, as configurações dos parâmetros utilizadas para este modelo foram definidas como se apresenta na figura 16.

```
{
"n_neighbors": [2,5,7, 10],
"algorithm": ["ball_tree", "kd_tree", "brute"],
"weights": ["uniform","distance"]
}
```

Figura 16 - Parâmetros testados no KNN Regressor

A seguinte figura representa o output dos melhores parâmetros que a função *GridSearch* identificou para o algoritmo em questão.

```
The best parameters across ALL searched params:
{'algorithm': 'brute', 'n_neighbors': 10, 'weights': 'uniform'}
R2 - Train : 0.9383
R2 - Test : 0.9259
MSE - Train : 0.0384
MSE - Test : 0.0476
```

Figura 17 - Output da função GridSearch na execução do KNN Regressor

Para este algoritmo, os parâmetros que ofereciam melhores resultados eram os seguintes:

- n_neighbors → 10;
- algorithm → brute;
- weights → uniform.

Para este conjunto de parâmetros, obtiveram-se os seguintes resultados das métricas de avaliação ao modelo:

- R2 score
  - Dados de treino ≃ 0.938;
  - Dados de teste ≃ 0.926.
- MSE
  - Dados de treino ≃ 0.038;
  - Dados de teste ≃ 0.048.

4.3.10. Linear Regression

O *Linear Regression* [43] trata-se da implementação de regressão linear que o *Scikit Learn* oferece. Ao contrário dos modelos anteriores, não é adequado colocar parâmetros no modelo de regressão linear.

Após a execução do modelo e posterior verificação das métricas de *performance*, obtiveram-se os resultados apresentados na figura 18:

```
R2 - Train : 0.9349
R2 - Test : 0.9365
MSE - Train : 0.0429
MSE - Test : 0.0458
```

Figura 18 - Output da perfomance do modelo Linear Regression

- R2 score
  - Dados de treino ≃ 0.935;
  - Dados de teste ≃ 0.936.
- MSE
  - Dados de treino ≃ 0.043;
  - Dados de teste ≃ 0.046.

4.3.11. Avaliação de todos os modelos

Depois de tirar o máximo partido de cada modelo, revelou-se ainda necessário comparar não só as performances obtidas pelas métricas de *Machine Learning*, mas também saber o tempo computacional gasto em cada uma. Neste sentido, ter-se-ia uma noção de quão custoso seria treinar um novo modelo e qual era o modelo que melhor se adaptaria a novas realidades, ou seja, que tivesse a necessidade de ser treinado novamente.

A recolha de resíduos urbanos caracteriza-se como um fator muito volátil, dado que pode variar com situações como obras, pandemias ou qualquer outro fator.

Para uma melhor comparação e síntese dos dados, apresenta-se na tabela 4 os dados recolhidos de cada modelo treinado e respetivos tempos de computação.

A figura 19 corresponde ao output da consola de cada tempo computacional, em segundos, gasto no treino de cada modelo, com os parâmetros otimizadas (pelos valores que foram obtidos com o *GridSearch).*

```
0.0009999275207519531 KNNR
5.9700000286102295 GBR
0.007999658584594727 LR
38.32306742668152  MLPR
```

Figura 19 - Output de tempos computacionais de criação de modelos de regressão

Na tabela comparativa (tabela 4) é salientado, a negrito, o melhor valor para cada um dos valores recolhidos e analisados de cada métrica. Os tempos de execução foram determinados em segundos.

Tabela 4 - Tabela das métricas para os diferentes algoritmos

Algoritmo	Dados de treino		Dados de teste		Tempo Computacional de Treino(segundos)
	R Score	MSE	R Score	MSE	
MLPR	**0.948**	**0.024**	0.942	0.028	38.3231
LR	0.935	0.043	0.936	0.046	0.0079
GBR	0.945	0.026	**0.945**	**0.027**	5.9700
KNN	0.938	0.038	0.926	0.048	**0.0001**

Como se pode observar na tabela 4, o algoritmo que se adaptou melhor aos dados de treino acabou por ser o algoritmo baseado numa rede neuronal (MLPR). Este algoritmo foi também o algoritmo que demorou mais a treinar. Apesar disso, este algoritmo revelou ainda ser capaz de prever com alto grau de certeza os valores dos dados de teste.

O algoritmo baseado no *Gradient Boost* revelou o melhor resultado nos dados de teste, superando os seus valores até na métrica MSE, em relação aos dados de treino (isto deve-se maioritariamente a questões de arredondamento). No entanto, nos dados de teste mostrou estar mais abaixo do que as redes neuronais quanto aos dados de treino. Apesar disso, teve sem dúvida um melhor desempenho do que as redes neuronais, dado que é uma diferença maior que trinta segundos, ou seja, com os seus parâmetros ajustados o melhor possível, dentro das hipóteses dadas, o GBR foi capaz de treinar muito mais rápido.

Existe um algoritmo que foi claramente o vencedor em termos de tempo gasto, o *K-Nearest Neighbors,* de forma que a sua execução era praticamente instantânea. Este algoritmo revelou ser capaz de fazer previsões corretamente tanto nos dados de treino, como nos dados de teste, não afetando muito a sua performance apesar de, ainda assim, ter menos desempenho que o MLPR e o GBR. Apesar de tudo, foi o que revelou piores resultados inerentes aos dados de teste. A regressão linear, apesar de não se ter destacado foi, à semelhança do KNN, muito rápida ao executar. No entanto, esta solução foi a que apresentou o pior valor nos dados de treino.

Seguindo a linha de pensamento, existem dois modelos candidatos para a escolha do modelo. Apesar de, no geral, o GBR ter apresentado melhores resultados que o MLPR, o algoritmo escolhido acabou por ser o MLPR, na esperança de que este, sendo um algoritmo

mais complexo, fosse capaz de no futuro continuar a apresentar excelentes resultados (mesmo demorando mais tempo a treinar), dado que não foi considerado excessivo o tempo de treino menos de um minuto.

No entanto, se o consumo dos dados fosse cada vez maior e houvesse uma dificuldade em termos de tempos computacionais a treinar o modelo em tempo útil, o GBR avançaria como um bom *backup,* pois revelou-se ser um modelo capaz de lidar com este tipo de problemas.

4.3.12. Treino do modelo final

Os modelos de *Machine Learning*, por vezes, podem demorar horas ou dias a correr, dependendo do tamanho do *dataset.* Como demonstrado anteriormente, a tomada de decisão ocorreu num modelo que se apresentou como o mais lento a treinar, contudo isso não significa que a solução é inviável.

Após a escolha do modelo e também dos parâmetros, o objetivo seria não necessitar de executar o treino do modelo sempre que fosse requisitada uma previsão. Para além disso, era necessário que o modelo permanecesse treinado, mesmo depois da execução do programa. Para que isto fosse possível, recorreu-se a uma biblioteca chamada *pickle.*

Esta biblioteca é responsável por guardar o modelo treinado num ficheiro, permitindo, posteriormente, o seu carregamento em futuras execuções. Assim, sempre que fosse necessário realizar uma nova previsão e/ ou uma nova execução, não seria preciso correr o algoritmo de novo, bastando carregar o modelo.

Esta biblioteca é uma biblioteca genérica de serialização de objetos, sendo útil tanto para “serializar” um objeto como para reverter a ação. Apesar de ser amplamente ligada a *Machine Learning,* ela é capaz de o fazer em qualquer objeto.

Para além disto, e tal como referido anteriormente, todas as variáveis sofreram normalização, tendo o modelo sido treinando com os dados normalizados. Para isto, é importante também guardar todos os fatores, pois, aquando de uma nova utilização do modelo é necessário proceder à normalização dos dados de *input* e só assim conseguir prever os dados.

De maneira a tornar isto possível foi necessário obter o valor mínimo e máximo para cada coluna para, posteriormente, ao utilizar o modelo, introduzir os valores adaptados à realidade com que o modelo foi treinado.

Capítulo 5

5. Algoritmo de VRP

Existem diversas abordagens para a resolução do problema de planeamento de rotas de veículos. De seguida, irão ser apresentadas todas as abordagens utilizadas nesta dissertação.

Nesta primeira comparação os algoritmos são responsáveis por resolver apenas o problema de planeamento de rotas simples para os veículos, não tendo em consideração outros fatores como, por exemplo, a carga do veículo. Estes testes iniciais/ comparações serviram como decisão para a implementação da solução final do problema.

Para qualquer teste feito foi fornecido a todos os algoritmos uma *hashtable* com a distância entre todos os pontos possíveis. Desta forma, o único fator diferenciador para a avaliação destes três métodos (Algoritmo Exato, Guloso e Genético) seria apenas a sua capacidade e a rapidez de criar solução, isolando o cálculo das distâncias.

Os algoritmos foram ainda executados todos na mesma máquina e no mesmo ambiente, evitando ao máximo que fatores externos tivessem a capacidade de influenciar nos testes desenvolvidos.

As especificações da máquina de testes eram as seguintes:

- CPU: Intel 8550U Quad Core;
- RAM: 16GB RAM 2400mhz;
- SSD: 256GB;
- OS: Linux Ubuntu.

5.1. Diferentes abordagens

5.1.1. Algoritmo exato

A primeira abordagem utilizada para a resolução do problema de planeamento de rotas de veículos acabou por ser um algoritmo exato. O desenvolvimento desse algoritmo consistiu na criação de todas as rotas possíveis e uma futura avaliação de todas elas, ou seja, para cada conjunto de pontos, o algoritmo gerava o número de soluções possíveis e depois percorria todas estas para encontrar a melhor solução.

Apesar desta devolver a solução ótima para este problema, tornava-se ineficiente, tanto em termos de memória como em termos de tempo. De facto, a solução revelou-se incapaz de responder a problemas com mais de 10 pontos, devido aos excessivos consumos

de memória RAM, pois esta necessitava de permanecer em memória com todas as soluções possíveis, juntamente com os seus cálculos, sendo que os 16GB de RAM utilizados pelo computador utilizado em testes não foram suficientes.

O algoritmo [44] retornava todas as soluções possíveis dado como parâmetro o tamanho das soluções requisitadas e um *array* com os valores possíveis. Este array apenas continha todas as opções possíveis, ou seja, não tinha qualquer tipo de processamento como por exemplo a sua ordenação.

De seguida, com a lista retornada pelo método anterior, foram efetuados os cálculos de distâncias e, posteriormente, feita a seleção do melhor caminho a percorrer, ou seja, do caminho em que era percorrido a menor distância.

Esta solução mostrou sem dúvida ser a menos otimizada e revela grandes problemas em resolver problemas maiores, mostrando-se assim impossível de ser equacionada.

5.1.2. Algoritmo genético

A segunda abordagem para a resolução do algoritmo de VRP foi um algoritmo genético. Esta abordagem apesar de não ser tão exata como a anterior era uma abordagem capaz de efetuar boas soluções em curtos intervalos temporais. Esta abordagem, tal como dito anteriormente, assenta na Teoria da Evolução de Darwin.

O número de gerações, assim como o limite de convergência, foram escolhidos com valores relativamente baixos, de forma a evitar que o algoritmo executasse durante longos períodos de tempo.

Para este projeto era importante que as soluções apresentadas fossem rápidas não sendo, mesmo assim, tão próximas da solução ótima. Então, a opção recaiu em ir diminuindo o limite da convergência e o limite de gerações, sem perder qualidade na solução obtida, de forma a obter o maior equilíbrio possível entre a qualidade da execução e a velocidade da mesma.

O tamanho da solução, por sua vez, também foi diminuído pelas mesmas razões. O número de mutações foi consideravelmente elevado em relação ao número de cruzamentos, devido a este último ter uma taxa de produção de boas soluções relativamente baixa. Já o aumento do número de mutações revelou-se sensivelmente importante para a aprendizagem, diminuindo o tempo de execução do algoritmo e aumentando a qualidade da solução oferecida.

Na tabela 5, encontra-se representadas as combinações referentes ao algoritmo genético que identificaram a melhor combinação, devido ao impacto das configurações, e depois de realizados alguns testes:

Tabela 5 - Configurações para o algoritmo genético

Tamanho da população	100
Tamanho da seleção	20
Limite para gerações	100
Limite de convergência	20
Número de mutações	40
Número de cruzamentos	10

Existem quatro funções que são essenciais para um algoritmo genético ser capaz de evoluir, sendo elas:

- a função de seleção natural;
- a função de cruzamento;
- a função de mutação.

5.1.2.1. Função de seleção natural

A função de seleção natural *(fitness)* era responsável por avaliar cada indivíduo e perceber a qualidade do mesmo.

Esta é a função responsável por correr passo a passo cada ponto do caminho de cada indivíduo e retornar o peso total do caminho, ou seja, o seu tamanho / custo.

5.1.2.2. Função de mutação

A função de mutação, tal como o nome indica, é responsável por uma mutação na esperança de esta representar um melhor resultado no final da mesma.

A lógica por detrás de toda esta função é a seguinte:

- Escolhe índices aleatórios, entre zero até ao tamanho da lista;
- Recolhe os pontos aleatórios representados por cada índice;
- Troca os pontos para cada índice.

Num caso prático, na existência de um caminho em que os pontos eram identificados por letras e o caminho gerado aleatoriamente, seria representado por:

A → B → C → D → E → F → G

- Dois índices eram escolhidos aleatoriamente, por exemplo, o número dois e o número cinco;

- Estas letras representadas no exemplo acima, seriam a letra C e a letra F respetivamente;
- De seguida, o algoritmo troca as duas letras (que representam diferentes pontos no caminho);
- A solução final obtida seria:

A → B → F → D → E → C → G

Apesar de estar a ser apresentada esta abordagem, outras abordagens podiam ter sido seguidas e chegaram a ser equacionadas. Uma das abordagens equacionadas consistia em escolher um ponto aleatório no meio do caminho, dividir o caminho em duas partes e invertê-las. Assim, se o caminho gerado fosse representado por:

A → B → C → D → E → F → G

- Um índice era escolhido aleatoriamente, por exemplo, o quatro;
- O índice quatro representa a letra D;
- O caminho iria repartir-se em dois diferentes caminhos:
  - A → B → C
  - E → F → G
- Os dois caminhos trocam a ordem;

E → F → G → D → A → B → C

Esta solução apesar de ter sido a primeira a ter sido implementada, era mais lenta a executar e necessitava também de mais gerações para produzir resultados semelhantes com a solução referida anteriormente, razão pela qual foi substituída.

5.1.2.3. Função de cruzamento

A função de cruzamento, tal como o nome indica, era responsável por reproduzir dois filhos com base nos pais aplicando a genética de ambos nos dois descendentes.

O objetivo deste tipo de funções é tentar encontrar uma solução melhor, com base em duas boas soluções já existentes. De forma simples, o objetivo passava em escolher em dois pontos aleatórios, perceber qual a sua posição entre os dois pais e depois invertê-los. Estes são os passos a seguir:

- Escolhe dois pontos de forma aleatória;
- Verifica qual a posição de estes pontos em cada pai;
- Copia os dois caminhos para os dois filhos;

- Inverte aqueles dois pontos em específico para cada filho.

De uma forma mais simples e com um exemplo seria da seguinte forma:

- Se os pontos possíveis são A, B, C, D, E, F, G;
- Os pontos F e C são escolhidos;
- Imaginando que o caminho dos dois pais fossem:
  - C → B → A → D → F → E → G
  - F → B → G → C → D → E → A
- Então os dois filhos iriam herdar um caminho cada um;
- Seguindo o raciocínio os índices dos pontos F e C são
  - Cinco e um respetivamente;
  - Um e quatro respetivamente;
- Removendo os pontos escolhidos de cada filho:
  - B → A → D → E → G
  - B → G → D → E → A
- Acrescentado os dois pontos de novo, mas com os índices provenientes do cromossoma ficaria:
  - F → B → A → C →D → E → G
  - C → B → A → D → F → E → G

Para a função de cruzamento existem outras abordagens que seriam exequíveis como a junção dos caminhos pelas metades (com a correção de pontos repetidos), no entanto, optando por esta, iria existir perda de performance, devido à existência de mais condições a validar por algoritmo. Por este motivo, houve a permanência desta abordagem para a solução final.

5.1.3. Algoritmo Guloso

A terceira abordagem utilizada para a resolução do problema de planeamento de rotas de veículos acabou por ser um algoritmo guloso. O desenvolvimento deste algoritmo consistiu na capacidade de aproveitar sempre o ponto mais próximo do ponto atual, de forma a tentar criar a solução mais otimizada nunca recuando ou olhando para trás de modo a tentar otimizar ainda mais.

Esta solução era uma solução muito rápida, pelo facto de ter apenas que iterar uma vez sobre a lista de pontos sempre que queria adicionar mais uma localização à solução dada, pois apenas procurava o local mais próximo.

A função era responsável pela procura do local mais próximo possível e retorná-lo à função responsável pela ordem dos pontos. Esta função sempre que recebia um ponto retirava-o da lista de pontos que ficavam ainda por recolher.

Apesar de ser à primeira vista uma abordagem simples, tal como é característico das heurísticas construtivas, esta solução revelou-se extremamente capaz de resolver o problema de forma eficiente e rápida.

5.2. Comparação dos algoritmos desenvolvidos

Tal como descrito anteriormente, foram apresentadas três formas distintas de resolver o algoritmo de VRP.

De forma a escolher o melhor algoritmo em questão era necessário haver uma comparação das três soluções desenvolvidas tanto a nível de performance (o tempo que cada uma demoraria a executar), como a nível da solução obtida (verificar se seria a solução ótima ou a sua proximidade à ótima).

A fim de possibilitar uma melhor análise foi criado um cenário simples, mas capaz de mostrar as diferenças entre os algoritmos. As distâncias eram previamente carregadas em memória, de forma que este não fosse um parâmetro diferenciador na solução, ou seja, que determinada solução fosse beneficiada por uma resposta mais lenta da base de dados.

Para uma melhor análise o algoritmo foi inicialmente testado com um pequeno número de pontos a percorrer. Após esta etapa inicial, o número de pontos foi aumentando de forma a verificar o impacto que este fator teria para os resultados e nos tempos computacionais.

Na tabela 6, estão demonstradas as comparações realizadas aos diferentes algoritmos, onde a primeira coluna indica o número de pontos a percorrer para o algoritmo resolver e, em seguida, agrupado em dois, tem o nome do algoritmo e, em baixo, o tempo e o resultado que tiveram.

Tabela 6 - Comparação dos diferentes algoritmos VRP

	Algoritmo Exato		Algoritmo Guloso		Algoritmo Genético	
Nº Pontos	Tempo de Execução (s)	Resultado Obtido (km)	Tempo de Execução (s)	Resultado Obtido (km)	Tempo de Execução (s)	Resultado Obtido (km)
5	0,00017	27	0,00000243	27	3,70	27
7	0,0087	36	0,00000622	36	3,81	36
9	0,89	44	0,00011	45	0,86	51
10	9,43	45	0,00016	46	0,86	59
11	110,10	50	0,00022	52	0,83	66
12	-	-	0,00018	55	0,81	70
13	-	-	0,00024	56	0,86	77
14	-	-	0,00027	63	0,93	83

Como se pode verificar, a solução exata implementada demonstrou ser uma solução incapaz de resolver problemas grandes em tempo computacional aceitável. Isto deve-se não só ao tempo que esta demorava, mas também há quantidade excessiva de RAM que esta necessitava, pois gerava todas as soluções possíveis e, por fim, iria avaliá-las uma a uma.

Quando o número de pontos a percorrer começou a ser igual ou maior que doze, os 16GB de RAM da máquina não eram capazes de proceder ao armazenamento e o algoritmo entrava em erro de memória. Apesar disto, esta solução revelou-se boa para soluções relativamente pequenas, ou seja, até nove pontos. No entanto, estes números de pontos não foram considerados suficientes para o estudo em questão e desde cedo a solução exata foi descartada.

Avaliando a solução do algoritmo genético é possível notar que esta teve pior desempenho num pequeno número de pontos em relação a números de pontos com valor maior. Isto deve-se à forma como o algoritmo estava configurado, ou seja, à variável limite de convergência, pois esta não deixava o algoritmo terminar se este fosse encontrando sucessivamente melhores soluções. Como é possível verificar, as duas primeiras linhas foram as únicas em que este algoritmo foi capaz de devolver a solução exata. Assim, é possível afirmar que este estava num processo contínuo de evolução, razão pela qual demorava mais a executar. Os resultados desta solução foram medianos na medida em que não se destacaram nem de forma positiva, nem de forma negativa, apresentando soluções próximas das outras soluções e em tempos computacionais curtos. É possível verificar o desvio em relação à solução do algoritmo guloso a aumentar à medida que o número de pontos aumentava. Devido a estes fatores esta solução também foi descartada.

Considerando agora a solução do algoritmo guloso, esta surpreendeu em muito com os seus resultados. Para além de ter conseguido nalgumas ocasiões encontrar a solução ótima, quase sempre apresentou uma solução muito boa falhando por muito pouco, como, por exemplo, no caso de o número de pontos ser nove, dez ou onze. No entanto, cedo se revelou a solução com melhor desempenho em tempos computacionais capaz de entregar resultados quase de forma instantânea. É possível notar que o aumento do tempo computacional foi gradual, mas impercetível em termos de utilização para um humano, pois a diferença entre o melhor registo em tempo computacional em comparação com o pior registo representava valores que não causariam qualquer impacto.

Devido ao que foi mencionado anteriormente a escolha recaiu no algoritmo guloso devido a este ser um algoritmo mais promissor para o sucesso da aplicação como um todo.

Com o recurso à tabela dos testes efetuados foi possível perceber qual o algoritmo desenvolvido com maior desempenho, no entanto, este algoritmo por si só, não era capaz de resolver problemas como a recolha dos resíduos urbanos.

O problema da recolha dos resíduos não se trata apenas de efetuar a recolha, pois há limitações inerentes ao camião que devem ser contabilizadas para a resolução deste problema. Entre as várias limitações como, por exemplo, o número de contentores ou a capacidade de o mesmo percorrer quilómetros sem abastecer, existe uma limitação que obrigatoriamente tem de ser contabilizada para que a solução desenvolvida seja capaz de resolver o problema no dia a dia, nomeadamente, a capacidade que o depósito do camião tem para transportar e armazenar o lixo dentro do próprio. Se este fator não for considerado no algoritmo em questão, supõe-se que o depósito do camião nunca é ocupado na sua totalidade, o que não retrata a sua realidade.

Desta forma, para o algoritmo ser válido para a consideração do problema, este deve ter em conta, o valor máximo que o camião pode carregar e também a quantidade de lixo total em cada contentor a ser transportada, pois só assim pode considerar todos os fatores para simular o caso na realidade.

A comparação efetuada no capítulo anterior foi útil na medida em que foi um ponto de partida para a solução final desenvolvida. Assim, na implementação de um algoritmo que fosse capaz de resolver o problema do CVRP (*Capacitated Vehicle Routing Problem*), ou seja, um problema capacitado pela quantidade de lixo que o camião consegue transportar, a solução que se decidiu adaptar a fim de acrescentar esta variável foi a heurística do algoritmo guloso.

A implementação base permaneceu na mesma lógica da anterior, mas foi necessário ter alguns pontos em consideração:

- Era necessário saber a capacidade máxima do camião;

- Era necessário saber as quantidades em cada contentor.

Existiu a necessidade de ter sempre em consideração que a quantidade do contentor mais a quantidade que já se encontrava no camião nunca poderia exceder a capacidade máxima do camião e esta foi a maior mudança em termos de lógica deste algoritmo. Para além disto, existiu a necessidade de criar uma condição em que obrigasse o camião a despejar a carga de novo, no centro de tratamento, de forma a conseguir, posteriormente, carregar mais.

O algoritmo está planeado para que se recolha a maior quantidade de resíduos urbanos possível em cada viagem, não atendendo única e exclusivamente ao custo da viagem neste caso, ao número de quilómetros. Ou seja, primeiramente, procura-se a solução em número de quilómetros que mais seja rentável (ou seja a menor distância possível); no caso de haver apenas uma e, essa não ser tão rentável, esta irá entrar de qualquer das formas, mesmo que seja demasiado custosa.

Este tipo de solução também faz com que seja possível dividir as recolhas por diferentes camiões e diferentes contentores em caso de recolha de lixo numa situação urgente, pois o *output* deste algoritmo consiste na criação de *n* caminhos capazes de recolher todos os contentores requisitados. Desta forma, em vez de um utilizador ter de dividir manualmente, a solução está preparada para o fazer à priori, pois basta simplesmente atribuir um dos *n* caminhos a *m* camiões que estejam disponíveis para a recolha.

Apesar de este acréscimo de condições feitas ao algoritmo guloso, mais uma vez, de forma a preencher os requisitos do problema de planeamento de rotas de veículos com limite de capacidade, este continuou com um excelente desempenho não tendo agravado a velocidade com que entrega as suas soluções. Este é um bom indicador, pois na eventualidade do número de contentores subir este algoritmo estaria preparado para continuar a entregar soluções.

Capítulo 6

6. Integração de todo o software

Ao longo deste capítulo serão mostrados todos os passos até à solução final, assim como uma breve descrição sobre a arquitetura escolhida para o projeto. Desta forma, será apresentado o projeto no seu todo, mostrando todas as integrações que foram necessárias com software de terceiros (*open source)* e com as integrações entre o software produzido.

6.1. Docker

O Docker [45] é uma tecnologia *open source* que permite a criação e gestão de ambientes isolados. Com esta tecnologia é possível lidar com os *containers* de forma simples, permitindo facilmente copiar ou migrar para outro ambiente, desde que este possua *Docker* instalado no seu sistema.

Ao contrário da virtualização, onde é necessário carregar um sistema operativo, no *Docker* não existe essa necessidade, dado que este utiliza o *Kernel* do Linux e, consequentemente, os seus recursos.

6.1.1. OSRM

Open Source Routing Machine [46] consiste numa implementação gratuita de um mecanismo de roteamento para caminhos mais curtos em redes rodoviárias.

Este serviço está disponível para plataformas como Linux, Windows e MacOS. Encontra-se ainda disponível em *Docker,* existindo forma de importar os dados para o computador, através do download de dois *containers*, o *OSRM-backend* e o *OSRM-frontend.* Este projeto é compatível com OSM (*OpenStreetMap*), logo estes arquivos podem ser facilmente importados.

6.2. Arquitetura

De um ponto de vista global pretende-se que, no final do projeto, seja desenvolvido um software capaz de identificar (através de previsão) os contentores que necessitem de ser recolhidos e, desta forma, delinear um plano diário destas necessidades. Com o apuramento anterior, pretende-se a criação de rotas otimizadas para que o camião de recolha execute. Este processo será provido de uma interface externa chamada OSRM, de forma a traçar o percurso e os seus passos.

A arquitetura proposta, na figura 20, divide-se em dois grandes componentes: *frontend* e *backend.*

Para o primeiro será usado um serviço *open source* chamado OSRM que será responsável por traçar um trajeto entre os diferentes pontos escolhidos pela aplicação principal. Sendo assim, este serviço não é responsável pela ordem dos contentores a recolher, mas sim por traçar um trajeto e ilustrá-lo no mapa.

Para o segundo, está implementada toda a lógica por trás do projeto. A comunicação com o serviço OSRM será através de um link enviado para o browser. Neste módulo estão incluídas todas as comunicações necessárias com a base de dados, assim como a utilização do algoritmo guloso e/ ou rede neuronal.

A base de dados, não só é responsável pela gravação de todos os dados anteriores do modelo, mas também pelo armazenamento de todas as distâncias entre os diferentes pontos.

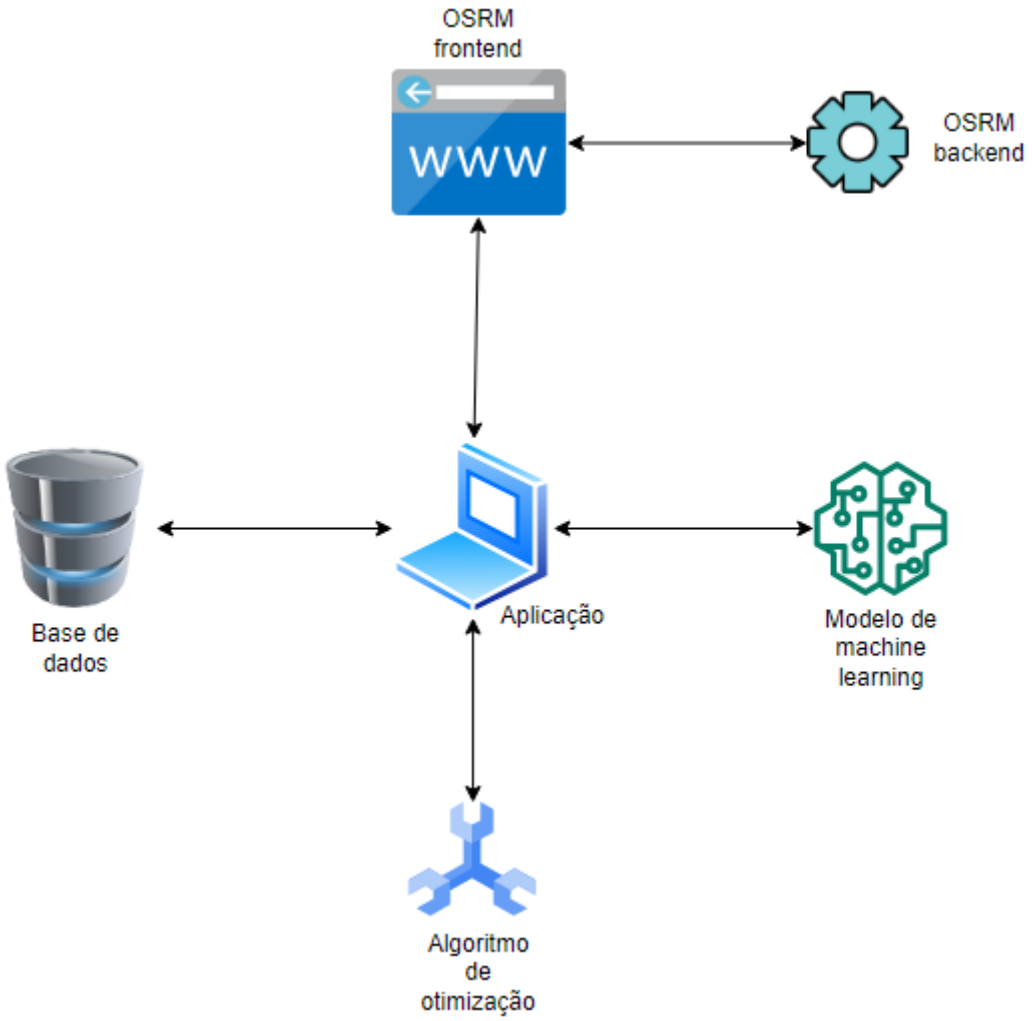


Figura 20 - Arquitetura da solução

Na tabela 7 estão descritos todos os componentes da arquitetura desenvolvida, assim como o seu respetivo papel.

Tabela 7 - Responsabilidades de cada componente da solução

Componente	Papel
Aplicação	Responsável por providenciar uma interface interativa.
Rede neuronal	Responsável pela previsão dos contentores a recolher.
Algoritmo guloso	Responsável pela ordem dos pontos a recolher.
Base de dados (MongoDB)	Responsável pelo armazenamento da distância entre os pontos.
OSRM backend	Responsável pelo trajeto entre cada ponto.
OSRM frontend	Responsável pela interface gráfica com o mapa.

Sendo assim, e de uma forma mais simplificada, o fluxo da aplicação de gestão de resíduos urbanos consiste nos seguintes passos:

- Utilização da rede neuronal para previsão dos contentores a ser recolhidos;
- Escolha por parte do utilizador dos contentores a ser recolhidos;
- Envio de um link corretamente formatado para o browser;
- Verificação da rota a percorrer.

6.3. Configuração do OSRM

O OSRM é possível de ser instalado de duas formas distintas: tanto localmente, como num contentor Docker. Para a instalação local, seria necessário instalar localmente o Node.js, assim como algumas bibliotecas C++, que este projeto utiliza na sua implementação.

Neste capítulo é apresentada a sua configuração em Docker, o que facilita a sua utilização, pois evita qualquer instalação de dependências e/ ou problemas de compatibilidade, devido a este estar num *container* isolado. Desta forma, o serviço é totalmente independente do computador em que está sendo, unicamente, requerido que este contenha *Docker* e que os mapas para a cidade ou país, em questão, estejam em volumes *Docker.*

Para além disso, é ainda explicado como é possível interagir com este tipo de serviço. De salientar que este serviço em *Docker* é dividido em dois componentes, isto é, no *OSRM-backend* e no *OSRM-frontend*, neste caso os dois com funções bem distintas, que serão mencionadas no decorrer do documento.

6.3.1. Configuração do OSRM-Backend

O *OSRM-backend* será responsável pelo trajeto entre os diferentes pontos utilizados, ou seja, dado um ponto A até ao ponto B, esta componente determinará em quais ruas o camião deve ou não passar, de forma a chegar ao contentor da maneira mais rápida e otimizada possível.

Nesta solução não está contemplada a questão do trânsito, uma vez que a recolha dos resíduos urbanos ocorre, normalmente, no período da noite. Logo, o fator "trânsito" não pode ser considerado um fator influenciador nestas condições.

Para configurar o *OSRM-backend,* primeiramente, é necessário efetuar o download do ficheiro *osm.pbf*. Aqui é possível escolher os ficheiros por continente ou país, relembrando que mediante a escolha feita o tamanho do ficheiro varia (devido à quantidade de informação inerente).

Este ficheiro é um tipo de ficheiro proprietário do OSM formatado como XML e é responsável pelo armazenamento de datas na forma de pontos, ligações e relações. No caso de fazer o download do ficheiro de Portugal são mapeados todos os pontos e ligações entre eles, assim como as relações que mapeiam e indicam o que significa cada ponto.

O *setup* deste projeto foi feito em Linux, logo a obtenção deste ficheiro foi efetuada com o seguinte comando:

- `wget https://download.geofabrik.de/europe/portugal.osm.pbf`

Após efetuar o download é necessário criar um volume *Docker* com a informação contida no ficheiro e fazer o download da versão mais recente da imagem do *osrm-backend.* É possível ainda, dependendo do tipo de rotas que é necessário usar, escolher o perfil adequado à funcionalidade requerida. Deste modo, torna-se possível escolher o perfil de:

- Carro: car.lua;
- Passeio a pé: foot.lua;
- Bicicleta: bicycle.lua.

Então, de forma a associar a imagem *Docker* e o volume que contém as informações, foi usado o seguinte comando:

- *docker run -t -v c:/docker:/data osrm/osrm-backend osrm-extract -p /opt/car.lua /data/portugal.osm.pbf*

Após esta configuração é possível correr o servidor de *backend* do servidor com o seguinte comando:

- *docker run --name osrm -t -i -p 5000:5000 -v c:/docker:/data osrm/osrm-backend osrm-routed --algorithm mld /data/portugal-latest.osrm*

Após a execução dos comandos anteriores, a configuração está terminada e é possível fazer pedidos ao servidor de *backend.* Os pedidos devem ser feitos através da porta 5000 (porta configurada no comando *docker)* e devem conter o link do servidor, ou seja, neste caso *localhost (*pois o servidor está a correr internamente no computador), a porta, o serviço e as coordenadas (figura 21).

```
GET  /{service}/{version}/{profile}/{coordinates}[.
     {format}]?option=value&option=value
```

Figura 21 - Exemplo de pedido get (retirado de OSRM)

Na figura 21 é possível perceber como o pedido *get* deste serviço funciona e que é possível requerer vários tipos de serviços, tais como, *route ou nearest*. É ainda possível escolher a versão do mesmo, assim como o perfil: o carro, a bicicleta ou passeio a pé.
De seguida é composto por um conjunto de coordenadas geográficas (separadas entre si por ponto e vírgula), no caso da longitude e da latitude estas estão separadas através de vírgulas.

Na parte final devem estar as coordenadas dos pontos a percorrer, separadas por ";" como demonstrado no exemplo do seguinte comando (figura 22):

- *curl* "http://127.0.0.1:5000/route/v1/driving/41.21737143364389,-8.41589657588237; 41.31737143364389,-8.31589657588237?steps=true"

Como resposta, o servidor retorna um JSON com os pontos, as coordenadas e o tempo que irá demorar entre cada ponto.

```
{
  "distance": 90.0,
  "duration": 300.0,
  "weight": 300.0,
  "weight_name": "duration",
  "geometry": {"type": "LineString", "coordinates":
[[120.0, 10.0], [120.1, 10.0], [120.2, 10.0], [120.3,
10.0]]},
  "legs": [
    {
      "distance": 30.0,
      "duration": 100.0,
      "steps": []
    },
    {
      "distance": 60.0,
      "duration": 200.0,
      "steps": []
    }
  ]
}
```


Figura 22 - Exemplo de resposta JSON (retirado de OSRM)

Apesar de ser um ficheiro de fácil leitura, por si só, esta solução não era suficiente e houve a necessidade de configurar o serviço seguinte, isto é, o OSRM-*frontend.*

6.3.2. Configuração do OSRM-frontend

Para que existisse uma interface gráfica com um mapa foi necessário configurar o OSRM-*frontend.* Para este serviço é necessário única e exclusivamente correr um comando *Docker*:

- docker run -p 9966:9966 osrm/osrm-frontend

Desta forma, foi possível obter a correspondência do JSON para um objeto mais *user-friendly,* como um mapa, tal como se demonstra na figura 23:

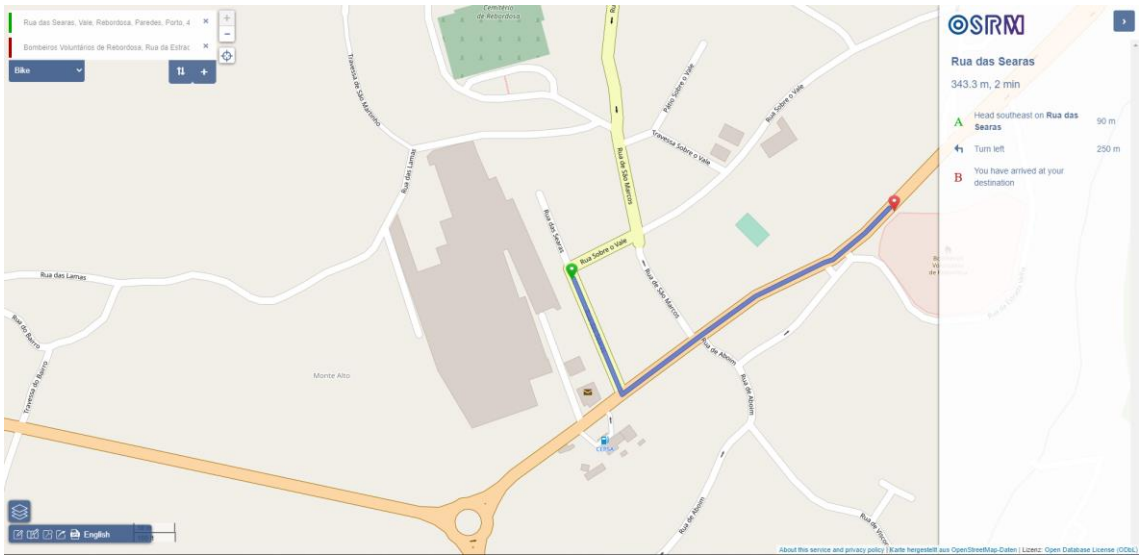


Figura 23 - Interface OSRM frontend

Do lado esquerdo é dada uma vista geral do mapa, com os pontos do trajeto assim como o caminho traçado para percorrer esses pontos. Por sua vez, no lado direito, estão disponíveis algumas informações acerca do trajeto a ser percorrido como a distância que irá ser percorrida e o tempo estimado. Para além disso, existe uma lista de indicações passo a passo que guiam o caminho.

Existe a possibilidade de comunicar com este *frontend,* através do url do browser. Assim, abrindo um browser e colocando um link, utilizando o servidor configurado na *Docker* é possível utilizar este serviço para ilustrar qualquer rota que esteja planeada.

6.4. Integração

A arquitetura desenvolvida para este projeto requer muita integração entre os diferentes componentes de software, sendo eles produzidos ao longo deste projeto ou então de componentes *open source.*

A aplicação desenvolvida está dividida em três grandes funções, ou seja, esta é responsável por fazer a previsão dos contentores a recolher, responsável por escolher a ordem dos contentores a recolher e ainda é responsável pela comunicação com o software em *Docker* (OSRM).

Esta aplicação é provida de um menu, que contém três opções distintas:

- Previsão de ocupação de contentores;
- Cálculo da melhor ordem a escolher para o trajeto da recolha dos resíduos;
- Treino do modelo de *Machine Learning.*

A primeira opção é responsável por carregar um modelo já previamente treinado e fazer a previsão da taxa de ocupação dos contentores. A previsão é feita com apenas um *input* manual, sendo este: se chove no dia em questão ou se há previsão para isso. Para além disso, todos os outros parâmetros são preenchidos de forma automática, tais como:

- Identificador de contentor;
- Longitude;
- Latitude;
- Dia;
- Mês;
- Ano;
- População.

Com base nestes *inputs* automáticos e no *input* manual, a previsão é feita para cada contentor (uma a uma). Devido à natureza do *dataset*, cada contentor é representado por uma linha deste e, deste modo, para a previsão de todos os contentores é necessário recorrer ao modelo de *Machine Learning* o número de vezes igual ao número de contentores que é necessário prever. Apesar das chamadas recorrentes ao modelo de *Machine Learning,* a aplicação consegue fornecer a informação em tempo útil e sem que o utilizador se aperceba das chamadas recorrentes.

Por fim, e após o sistema obter todas os valores para cada contentor, este recomenda ao utilizador recolher determinados contentores. No entanto, possibilita que, de forma manual, o utilizador possa incluir outros contentores para fazer o cálculo das rotas e, desta forma, o sistema torna-se um auxílio para o trabalhador, não impedindo a sensibilidade e experiência humana capaz de pensar em cenários que a máquina não prevê.

Após a escolha destes contentores é dado ao utilizador a opção de prosseguir com a previsão das rotas.

A segunda opção, cálculo da melhor ordem a escolher para o trajeto da recolha dos resíduos, consiste no melhor cálculo possível das rotas para determinados contentores com recurso a *input* do utilizador, ou seja, para casos em que é necessário recolher os contentores, não tendo em conta a sua ocupação. Aqui é possível, mais uma vez, escolher todos os contentores para recolher ou então escolher um determinado conjunto.

A terceira opção, treino do modelo de *Machine Learning*, consiste em fazer o treino do modelo de *Machine Learning*, ou seja, caracteriza-se na leitura do *dataset* e consequente criação de conhecimento. Apesar de neste caso não existir, devido a ser um cenário de teste, há a capacidade por parte da aplicação de treinar o modelo, de modo a aperfeiçoar o seu conhecimento. Dito isto, esta parte da aplicação consiste em fazer alterações ao

conhecimento do modelo com base em adição de linhas no *dataset.* É espectável que, com o uso da aplicação, os futuros dados do *dataset* possam mudar.

Desta forma, era necessário ajustar o modelo à nova informação mantendo o modelo atualizado para uma diferente realidade. O treino do modelo implica também todas as fases de pré-processamento do *dataset* como *one hot encoding* e a normalização dos dados.

6.5. Principal fluxo da aplicação

Fazendo uso de todas as funcionalidades da aplicação, o fluxo da aplicação passava por primeiro fazer o *loading* do modelo e construir todo o conhecimento de novo. Para isto, o modelo necessitava de ler todo o *dataset,* fazer todos os tratamentos e guardar os valores de máximo e mínimo para cada coluna para, posteriormente, efetuar as previsões seguintes e normalizar o *input* antes de o enviar.

De seguida, o utilizador tem tudo para começar a resolver o problema da recolha de resíduos. Deve, primeiramente, selecionar a opção de previsão de ocupação dos contentores, e de seguida, é-lhe mostrado todos os contentores assim como a previsão de taxa de ocupação para cada um, relembrando que o modelo de *Machine Learning* é acedido *n* vezes, sendo este *n* definido pelo número de contentores pelo qual se está a prever a taxa de ocupação. O utilizador tem a possibilidade de fazer a sua análise e escolher quais contentores devem ser recolhidos apesar do sistema dar uma sugestão dos mesmos.

Após o processo de previsão, este tem a opção de solicitar a funcionalidade de geração de rotas, que irá efetuar as operações necessárias a fim de identificar o caminho mais curto (tendo em conta a ordem com que se vai recolher o contentor). Este serviço requer à base de dados todas as distâncias entre os pontos e mapeia essas distâncias num dicionário, desta forma não é necessário recalcular as distâncias de um ponto ao outro sempre que é necessário calcular a rota mais eficiente. Logo, torna a resposta ao utilizador mais rápida.

Posteriormente é integrado a ordem da rota com a solução externa em *Docker* (OSRM), sendo aberta uma página web no navegador, com um link pré-formatado que contém todos os pontos a recolher pela ordem correta, desta forma a solução de OSRM *frontend* é capaz de descodificar e fazer o pedido ao OSRM *backend,* de forma a fornecer do lado esquerdo um mapa com o caminho tracejado e os diferentes pontos marcados e do lado direito um pequeno resumo da viagem, assim como as indicações passo a passo para a mesma (como está representado na figura 23).

Capítulo 7

7. Conclusão

Com o aumento da população há, naturalmente, um aumento do consumo e, consequentemente, um aumento da quantidade de resíduos que são produzidos.

O tema da recolha de resíduos urbanos consiste num tema cada vez mais recorrente na atualidade, pois a presença (ou não) deste nas ruas é capaz de influenciar a opinião sobre uma cidade e, por esse motivo, ter impacto em questões como, por exemplo, o turismo.

Com o decorrer deste projeto é possível evidenciar que este tema, apesar de ser um tema comum a toda a sociedade, apresenta muitas particularidades que o tornam um tema interessante, contudo um problema difícil de resolver.

Considerado o propósito deste trabalho o resultado foi satisfatório, tanto a nível da resolução do primeiro problema (previsão da taxa de ocupação), como a nível do segundo problema (o trajeto da recolha do lixo). Considerando o primeiro problema foi possível perceber que todos os algoritmos tiveram um bom desempenho ao realizarem as previsões quanto à ocupação dos contentores, pois obtiveram todos uma classificação acima de 90% na métrica de *R-Score* e apresentaram valores de erro na métrica MSE consideravelmente baixos. Por sua vez, o algoritmo baseado em redes neuronais e o algoritmo baseado em *Gradient Boost* foram os dois algoritmos que obtiveram os melhores resultados, tendo sido considerado para a solução final o algoritmo das redes neuronais.

Considerando o segundo problema é importante salientar que vários aspetos não foram considerados, sendo que o problema foi apenas resolvido considerando a capacidade do contentor. Não obstante, foi possível fazer uma comparação entre diferentes algoritmos chegando à conclusão que só dois dos algoritmos implementados eram exequíveis para o projeto (sendo que o algoritmo exato não conseguia retornar resultados em tempo útil). Quanto aos outros dois algoritmos, foi possível verificar que o algoritmo guloso, dentro dos testes efetuados foi o que deu mais segurança para a solução. No entanto, o algoritmo genético era capaz de entregar soluções em tempo útil, mas com um défice na questão da qualidade das mesmas. Assim, a opção escolhida recaiu sobre o algoritmo guloso, tendo sido os resultados muito animadores (na maior parte dos casos em que foi comparado com a solução exata, encontrou resultados ótimos ou muito próximos disso e em tempos computacionais muito mais curtos) e a solução dada por este apresentou-se como uma boa adição ao projeto.

Todo este projeto não teria tanto impacto se não existissem soluções como o OSRM. Este serviço *open-source em Docker* permitiu alcançar outro patamar no projeto, apesar de, inicialmente, ter havido algumas dificuldades na configuração do mesmo, este entrega uma excelente interface gráfica e uma ótima compatibilidade com rotas em todo o mundo.

Com o final deste projeto é possível concluir que o sistema tem falhas e estas são facilmente identificáveis. Esta solução mostra alguns processos que podem ser melhorados de forma a poupar combustível, evitar maior poluição devido a quilómetros desnecessários, assim como evitar grandes aglomerados de resíduos à volta dos contentores. O objetivo deste sistema seria sem dúvida facilitar a vida a qualquer trabalhador que faz parte do processo da recolha e através deste projeto, este objetivo principal foi atingido.

7.1. Trabalho Futuro

Apesar do objetivo principal do projeto ter sido concluído, como trabalho futuro tem-se como objetivos melhorar um conjunto de aspetos que tornem a solução mais robusta, mas também mais fácil de usar.

Assim, um outro objetivo seria a integração de equipamentos IOT, de forma a monitorizar em tempo real a ocupação dos contentores e, desta forma, conseguir evitar situações anómalas como, por exemplo, obras. Um modelo de *Machine Learning*, apesar de ser capaz de prever a maior parte dos casos, tem dificuldade em prever situações nunca conhecidas sendo que, situações destas podiam ser facilmente monitorizadas com a integração de pequenos sensores infravermelhos dentro dos contentores do lixo, a fim de conhecer a ocupação do mesmo.

Aliado à solução anterior, seria também conveniente implementar uma solução web, capaz de demonstrar os dados obtidos pelos sensores (em tempo real), assim como o fornecimento de uma interação ativa com o modelo de *Machine Learning* e o próprio algoritmo.

Outro objetivo seria alterar o algoritmo de cálculo de rotas e considerar questões como o trânsito que no decorrer do projeto foi desvalorizado assumindo que o lixo seria recolhido de noite e que essa variável iria ter pouca interferência, mas que, noutras situações em que o lixo necessita de ser recolhido durante o dia seria um fator relevante.

O desenvolvimento de um *cronjob* que treinaria o modelo de forma periódica e iria atualizando o modelo sem interferência para o utilizador.

Para além do anterior mencionado, seria útil que fossem treinados novos modelos para um planeamento semanal (em vez do diário) considerando as previsões para os dias seguintes (todos os dias da semana) e não apenas do dia seguinte.

Capítulo 8

8. Referências

[1] «Home | Agência Portuguesa do Ambiente». https://apambiente.pt/ [Acedido em 12 Março de 2022].

[2] «Resíduos Urbanos | Porto Ambiente». https://www.portoambiente.pt/residuos-urbanos/residuos-urbanos [Acedido em 12 Abril de 2022].

[3] D. Sarkar, R. Bali, e T. Sharma, *Practical Machine Learning with Python*. Apress, 2018. doi: 10.1007/978-1-4842-3207-1.

[4] «View of Classification Techniques in Machine Learning: Applications and Issues | Journal of Basic & Applied Sciences». https://setpublisher.com/pms/index.php/jbas/article/view/1715/1579 [Acedido em 24 Março de 2022].

[5] P. Cunningham, M. Cord, e S. J. Delany, «Supervised learning», *Cognitive Technologies*, pp. 21–49, 2008, doi: 10.1007/978-3-540-75171-7_2.

[6] S. I. Bangdiwala, «Regression: simple linear», *Int J Inj Contr Saf Promot*, vol. 25, n. 1, pp. 113–115, Jan. 2018, doi: 10.1080/17457300.2018.1426702.

[7] L. E. Eberly, «Multiple Linear Regression», *From: Methods in Molecular Biology*, vol. 404.

[8] K. (Kevin N.) Gurney, *An introduction to neural networks*. UCL Press, 1997.

[9] S. Sakib, A. 1#, A. Jawad, K. 2@, e H. Ahmed, «An Overview of Convolutional Neural Network: Its Architecture and Applications», 2019, doi: 10.20944/preprints201811.0546.v4.

[10] J. v Tu, «Advantages and Disadvantages of Using Artificial Neural Networks versus Logistic Regression for Predicting Medical Outcomes», 1996.

[11] A. Natekin e A. Knoll, «Gradient boosting machines, a tutorial», *Front Neurorobot*, vol. 7, n. DEC, 2013, doi: 10.3389/fnbot.2013.00021.

[12] Y. Song, J. Liang, J. Lu, e X. Zhao, «An efficient instance selection algorithm for k nearest neighbor regression», *Neurocomputing*, vol. 251, pp. 26–34, Ago. 2017, doi: 10.1016/J.NEUCOM.2017.04.018.

[13] M. Spuler, A. Sarasola-Sanz, N. Birbaumer, W. Rosenstiel, e A. Ramos-Murguialday, «Comparing metrics to evaluate performance of regression methods for decoding of neural signals», *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, vol. 2015-November, pp. 1083–1086, Nov. 2015, doi: 10.1109/EMBC.2015.7318553.

[14] A. Gelman, B. Goodrich, J. Gabry, e A. Vehtari, «R-squared for Bayesian Regression Models», *American Statistician*, vol. 73, n. 3. American Statistical Association, pp. 307–309, Jul. 03, 2019. doi: 10.1080/00031305.2018.1549100.

[15] «What is a good R-Squared value? (simply explained)». https://stephenallwright.com/good-r-squared-value/ [Acedido em 26 Junho de 2022].

[16] T. Chai e R. R. Draxler, «Root mean square error (RMSE) or mean absolute error (MAE)? -Arguments against avoiding RMSE in the literature», *Geosci Model Dev*, vol. 7, n. 3, pp. 1247–1250, Jun. 2014, doi: 10.5194/gmd-7-1247-2014.

[17] R. Zaman Khan, haider allamy, W. Hong, e H. D. Khalaf Jabbar Rafiqul Zaman Khan, «METHODS TO AVOID OVER-FITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY) Related papers Neural Net works for Classificat ion: A Survey METHODS TO AVOID OVER-FITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY)», 2015.

[18] X. L. Dong e T. Rekatsinas, «Data Integration and Machine Learning: A Natural Synergy», 2018, doi: 10.1145/3183713.3197387.

[19] P. Jamal, M. Ali, R. H. Faraj, P. J. M. Ali, e R. H. Faraj, «1-6 Data Normalization and Standardization: A Technical Report», *Machine Learning Technical Reports*, vol. 1, n. 1, pp. 1–6, 2014

[20] L. Yu, R. Zhou, R. Chen, e K. K. Lai, «Missing Data Preprocessing in Credit Classification: One-Hot Encoding or Imputation?», *Emerging Markets Finance and Trade*, vol. 58, n. 2, pp. 472–482, 2022, doi: 10.1080/1540496X.2020.1825935.

[21] V. Jayaraman, S. Parthasarathy, A. R. Lakshminarayanan, e H. K. Singh, «Predicting the Quantity of Municipal Solid Waste using XGBoost Model», em *Proceedings of the 3rd International Conference on Inventive Research in Computing Applications, ICIRCA 2021*, Set. 2021, pp. 148–152. doi: 10.1109/ICIRCA51532.2021.9544094.

[22] L. Yang, Y. Zhao, X. Niu, Z. Song, Q. Gao, e J. Wu, «Municipal Solid Waste Forecasting in China Based on Machine Learning Models», *Front Energy Res*, vol. 9, Nov. 2021, doi: 10.3389/fenrg.2021.763977.

[23] M. Kannangara, R. Dua, L. Ahmadi, e F. Bensebaa, «Modeling and prediction of regional municipal solid waste generation and diversion in Canada using machine learning approaches», *Waste Management*, vol. 74, pp. 3–15, Abr. 2018, doi: 10.1016/j.wasman.2017.11.057.

[24] Z. Ceylan, «Estimation of municipal waste generation of Turkey using socio-economic indicators by Bayesian optimization tuned Gaussian process regression», *Waste Management and Research*, vol. 38, n. 8, pp. 840–850, Ago. 2020, doi: 10.1177/0734242X20906877.

[25] P. Toth, D. Vigo, e Society for Industrial and Applied Mathematics, *Vehicle routing : problems, methods, and applications*.

[26] «Problema de Roteamento de Veículos», Available: http://dis.unal.edu.co/~gjhernandezp/TOS/ROUTING/VRP1.pdf [Acedido em 10 Agosto de 2022].

[27] Y. Tao e F. Wang, «An effective tabu search approach with improved loading algorithms for the 3L-CVRP», *Comput Oper Res*, vol. 55, pp. 127–140, 2015, doi: 10.1016/J.COR.2013.10.017.

[28] A. Olivera e O. Viera, «Adaptive memory programming for the vehicle routing problem with multiple trips», *Comput Oper Res*, vol. 34, n. 1, pp. 28–47, Jan. 2007, doi: 10.1016/J.COR.2005.02.044.

[29] H. Zhang, Z. Wang, M. Tang, X. Lv, H. Luo, e Y. Liu, «Dynamic memory memetic algorithm for VRPPD with multiple arrival time and traffic congestion constraints», *IEEE Access*, vol. 8, pp. 167537–167554, 2020, doi: 10.1109/ACCESS.2020.3023090.

[30] F. K. Miyazawa, «Algoritmos Exatos». https://www.ic.unicamp.br/~fkm/lectures/slides-exatos.pdf [Acedido em 10 Setembro de 2022].

[31] M. O. Ball, «Heuristics based on mathematical programming», *Surveys in Operations Research and Management Science*, vol. 16, n. 1, pp. 21–38, Jan. 2011, doi: 10.1016/J.SORMS.2010.07.001.

[32] A. Vince, «A framework for the greedy algorithm», *Discrete Appl Math (1979)*, vol. 121, n. 1–3, pp. 247–260, Set. 2002, doi: 10.1016/S0166-218X(01)00362-6.

[33] H. A. Amr, M. Khajehzadeh, M. Taha, Z. Beheshti, S. Mariyam, e H. Shamsuddin, «A Review of Population-based Meta-Heuristic Algorithm A Survey on Met a-Heurist ic Global Opt imizat ion Algorit hms A Review of Population-based Meta-Heuristic Algorithm», *Int. J. Advance. Soft Comput. Appl*, vol. 5, n. 1, pp. 2074–8523, 2013,

[34] A. F. Gad, «PyGAD: An Intuitive Genetic Algorithm Python Library».

[35] H. Park, D. Son, B. Koo, e B. Jeong, «Waiting strategy for the vehicle routing problem with simultaneous pickup and delivery using genetic algorithm», *Expert Syst Appl*, vol. 165, Mar. 2021, doi: 10.1016/j.eswa.2020.113959.

[36] T. Erdelić, T. Carić, M. Erdelić, e L. Tišljarić, «Electric vehicle routing problem with single or multiple recharges», em *Transportation Research Procedia*, 2019, vol. 40, pp. 217–224. doi: 10.1016/j.trpro.2019.07.033.

[37] R. Sai, N. Varma, P. J. Swaroop, B. K. Anand, N. Yadav, e N. Janarthanan, «IoT Based Intelligent Trash Monitoring System with Route Optimization Method», *International Journal of Electrical Engineering and Technology (IJEET)*, vol. 11, n. 4, pp. 57–65, 2020,

[38] J. R. Montoya-Torres, J. López Franco, S. Nieto Isaza, H. Felizzola Jiménez, e N. Herazo-Padilla, «A literature review on the vehicle routing problem with multiple depots», *Computers and Industrial Engineering*, vol. 79. Elsevier Ltd, pp. 115–129, 2015. doi: 10.1016/j.cie.2014.10.029.

[39] «sklearn.model_selection.GridSearchCV — scikit-learn 1.1.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html [Acedido em 11 Setembro de 2022].
[40] «sklearn.neural_network.MLPRegressor — scikit-learn 1.1.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html [Acedido em 11 Setembro de 2022].
[41] «sklearn.ensemble.GradientBoostingRegressor — scikit-learn 1.1.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html [Acedido em 11 Setembro de 2022].
[42] «sklearn.neighbors.KNeighborsRegressor — scikit-learn 1.1.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html [Acedido em 11 Setembro de 2022].
[43] «sklearn.linear_model.LinearRegression — scikit-learn 1.1.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html [Acedido em 11 Setembro de 2022].
[44] «Generate all possible combinations of at most X characters from a given array - GeeksforGeeks». https://www.geeksforgeeks.org/generate-all-possible-combinations-of-at-most-x-characters-from-a-given-array/ [Acedido em 7 Janeiro de 2022].
[45] «Docker». https://www.docker.com/ [Acedido em 11 Setembro de 2022].
[46] «Project OSRM». https://project-osrm.org/ [Acedido em 2 Fevereiro de 2022].