1 2 9 0

UNIVERSIDADE D
COIMBRA

Filipe José Good da Silva

Aprendizagem Computacional Automatizada

Criação de um Módulo de Aprendizagem Computacional Automatizada para Cientistas de Dados

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes orientada pelo Doutor Rui Lopes e pela Professora Bernardete Martins Ribeiro e apresentado à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Junho de 2020

1 2 9 0 UNIVERSIDADE D COIMBRA

Faculdade de Ciências de Tecnologias

Departamento de Engenharia Informática

Aprendizagem Computacional Automatizada

Criação de um Módulo de Aprendizagem Computacional Automatizada para Cientistas de Dados

Filipe José Good da Silva

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes, orientado pelo Doutor Rui Lopes e pela Professora Bernardete Martins Ribeiro e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática

Junho 2020

1 2 9 0

UNIVERSIDADE D
COIMBRA

Esta página foi intencionalmente deixada em branco.

Resumo

A área de Aprendizagem Computacional nunca teve tanto interesse e influência como nos dias de hoje. Várias são as outras áreas em que esta pode acrescentar valor e fazer face à crescente necessidade de melhoria, desde a área humana em que as nossas decisões são tomadas por algoritmos informáticos que foram desenvolvidos para executar determinadas tarefas, à área industrial onde as empresas recorrem a Aprendizagem Computacional para obter valor da quantidade enorme de dados que produzem. Contudo, desenvolver sistemas de Aprendizagem Computacional não é trivial, exigindo muito conhecimento e tempo, tornando assim o trabalho limitado a pessoas com experiência na área.

Aprendizagem Computacional Automatizada (AutoML) procura remover limitações associadas ao desenvolvimento de sistemas dotados de inteligência ao automatizar as diferentes fases de um projecto de Aprendizagem Computacional. Esta nova área tenciona fazer face à necessidade crescente de ferramentas que tornam Aprendizagem Computacional mais acessível e menos complexa.

Neste trabalho explorámos as capacidades actuais de AutoML de forma a implementar um módulo de AutoML. O módulo implementado está capacitado para realizar diversas etapas de um projecto de Aprendizagem Computacional de forma automatizada. Além disso, explorámos também um cenário onde AutoML pode ser integrado. Neste sentido, o módulo implementado foi integrado num assistente virtual, criando assim uma prova de conceito que permite a execução de operações de AutoML com recurso à comunicação em linguagem natural.

Os resultados obtidos demonstram que as duas ferramentas implementadas permitem ultrapassar duas dificuldades no que toca à implementação de projectos de Aprendizagem Computacional. Por um lado, o módulo de AutoML reduz a complexidade associada ao desenvolvimento de sistemas inteligentes, permitindo assim que indivíduos sem conhecimento em Aprendizagem Computacional possam beneficiar da mesma. Por outro, o assistente virtual implementado elimina a necessidade de experiência de programação que é, por norma, fundamental em projectos de Aprendizagem Computacional.

Palavras-Chave

Ciência de Dados, Aprendizagem Computacional, Aprendizagem Computacional Automatizada, Processamento de Linguagem Natural

Esta página foi intencionalmente deixada em branco.

Abstract

The area of Machine Learning has never had as much interest as it has today. There are several other areas in which it can add value and address the growing need for improvement, from the human area in which our decisions are made by computer algorithms that were developed to perform certain tasks, to the industrial area where companies make use of to Machine Learning to gain value from the huge amount of data they produce. However, developing a Machine Learning system is not trivial. It is a complex task that requires a large amount of knowledge and time, limiting its development to people with experience in the area.

Automated Machine Learning (AutoML) seeks to remove the limitations associated with developing intelligent systems by automating the different phases of a Machine Learning project. This new area aims to address the growing need for tools that make Machine Learning more accessible and less complex.

In this work, we explored the current capabilities of AutoML in order to develop an AutoML module. The implemented module is able to execute several phases of a Machine Learning project in an automated way. In addition, we also explored a scenario where AutoML could be integrated. In this respect, the implemented module was integrated in a virtual assistant, thus creating a proof of concept that allows the execution of AutoML operations using natural language.

Our results suggest that the two implemented tools enable to overcome two obstacles related with the implementation of Machine Learning projects. On one hand, the AutoML module reduces the complexity associated with the development of intelligent systems, thus allowing individuals without knowledge in Machine Learning to benefit from it. On the other hand, the implemented virtual assistant eliminates the need for programming experience, that is usually vital in Machine Learning Projects.

Keywords

Data Science, Machine Learning, Automated Machine Learning, Natural Language Processing

Esta página foi intencionalmente deixada em branco.

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos orientadores Rui Lopes, Tiago Baptista e Professora Bernardete, pelo apoio e disponibilidade demonstrada ao longo do ano. Em especial agradecer ao Rui Lopes, que embora não tenha me acompanhado no primeiro semestre, mostrou ser uma grande ajuda, estando sempre disponível para discutir as abordagens propostas e fornecendo sempre pontos de vista relevantes.

Em seguida, gostaria de agradecer ao meu grupo de amigos, vulgarmente apelidado de Ceia de Pêra. Com este grupo conheci Coimbra e todas as suas maravilhas. Obrigado por todos os momentos que tornaram a minha experiência em Coimbra inesquecível. Momentos esses que recordo com imensa alegria e eterna saudade. Gostava ainda de deixar um agradecimento especial ao Henrique Branquinho, Carlos Diogo e Guilherme Basto. Obrigado por toda a amizade!

Ao Gonçalo Lopes, que apesar de fazer parte do grupo de amigos mencionado acima, merece menção honrosa. Obrigado por todas as correcções e melhorias que tornaram este relatório mais legível. Foste incansável durante a fase final e por isso estarei eternamente grato. A garrafa de vinho que te vou oferecer não chega para exprimir a gratidão que tenho para contigo.

À Diana Sarmento, uma das pessoas mais importantes da minha vida. Quero agradecer não só os pequenos gestos como, por exemplo, as mensagens matinais de motivação, mas também todo o carinho demonstrado ao longo destes últimos dois anos.

Por último mas não menos importante, quero agradecer à minha família, particularmente aos meus pais e à minha irmã. Aos meus pais por me ajudarem a chegar onde estou hoje e por todos os ensinamentos transmitidos ao longo desta caminhada. À minha irmã por ser um grande exemplo de todo o tipo de qualidades que se quer numa pessoa. Os últimos meses deste estágio foram feitos em casa, em virtude da pandemia actual, e se hoje estou a escrever os agradecimentos é porque todo o suporte dado pela minha família nestes últimos meses o permitiu.

Esta página foi intencionalmente deixada em branco.

Conteúdo

Acrónimos

AutoML Aprendizagem Computacional Automatizada. xi, xiii, 2–5, 9–12, 18, 19, 22, 24, 25, 28, 29, 32–35, 38, 40, 43, 58–60, 73, 77, 78, 89, 90, 92, 93, 96–98, 100–102

CASH Combined Algorithm Selection and Hyper-parameter. 18, 24, 25

CPU Unidade central de processamento. 27

DARTS Differentiable Architecture Search. 21

DEI Department of Informatics Engineering. 2

ENAS Efficient Neural Network Architecture Search. 20

GPU Unidade de processamento gráfico. 27

LEAF Learning Evolutionary AI Framework. 21

MAE Mean Absolute Error. xiii, 61, 62, 71–73

MAPE Mean Absolute Percentage Error. xi, 61, 62, 71, 73

MIT Massachusetts Institute of Technology. 17

NAS Neural Architecture Search. 18–21, 24, 26, 28

RL Reinforcement Learning. 20, 21, 23, 24, 28, 39, 48–52, 54, 97, 101

RNN Recurrent Neural Network. 20

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Esta página foi intencionalmente deixada em branco.

Capítulo 1

Introdução

O documento presente especifica o trabalho realizado no estágio para a obtenção do Mestrado em Engenharia Informática pela Universidade de Coimbra. O estágio tomou lugar na Critical Software no ano lectivo 2019/2020. A Critical Software é uma empresa de Coimbra que se foca, há mais de 20 anos, no desenvolvimento de sistemas críticos. Por parte da Critical Software, o estágio foi orientado pelo Doutor Rui Lopes e co-orientado pelo Doutor Tiago Baptista. Por parte do Department of Informatics Engineering (DEI), o estágio foi orientado pela Professora Bernardete Martins Ribeiro. O trabalho exposto ao longo do documento teve como finalidade a implementação de duas ferramentas: um módulo de AutoML e um assistente virtual com competências de um cientista de dados.

1.1 Motivação

A área de Aprendizagem Computacional está cada vez mais presente no nosso dia-a-dia, onde, poucos são os aspectos do nosso quotidiano que não são influenciados por algoritmos de Aprendizagem Computacional [1]. Desde decisões simples como por exemplo, sugerir o próximo livro que vamos ler, a decisões complexas relacionadas por exemplo, com a segurança nacional. Esta área, apesar de ter tido um crescimento relativamente recente, já alterou o paradigma da forma como experienciamos a vida e, tem o potencial de alterar-lo ainda mais.

O recente crescimento desta área deve-se principalmente a dois motivos [2]: o aumento da quantidade de dados e o aumento do poder computacional. Estes dois factores quebraram os obstáculos que investigadores do século passado enfrentaram permitindo assim trazer de volta a investigação em torno de Aprendizagem Computacional. O aumento do poder computacional permite o processamento de grandes conjuntos de dados gerados diariamente, sendo estes por outro lado, essenciais para que as técnicas de Aprendizagem Computacional consigam reproduzir resultados seguros e fiáveis.

Apesar do crescimento do poder computacional e da quantidade de dados disponível terem impulsionado esta área, o desenvolvimento de um sistema dotado de inteligência, capaz de aprender e identificar padrões nos dados, continua a não ser uma tarefa trivial. Um projecto de Aprendizagem Computacional compreende um conjunto de fases que tornam a sua implementação trabalhosa, demorada, complexa e propensa a erros [2]. Cada fase,

necessita do conhecimento de um cientista de dados que tem de fazer diversas escolhas relativas ao problema a resolver, ao conjunto de dados e aos algoritmos a utilizar. Estas decisões são tomadas a partir da experiência e intuição do mesmo, num processo de tentativa-erro, tornando este propício a erros que directamente influenciam a performance e qualidade do trabalho produzido.

A complexidade inerente a Aprendizagem Computacional, a falta de profissionais experientes, o custo e o crescente interesse nesta área, tem levado investigadores e empresas a pensar numa forma de facilitar todo este processo [3]. Como consequência surgiu uma nova área: AutoML (Aprendizagem Computacional Automatizada). Esta área pretende minimizar o esforço e conhecimento associado a projectos de Aprendizagem Computacional [4], procurando automatizar as diversas fases de um projecto deste género.

Tal como o nome sugere, o principal objectivo desta área resume-se na possibilidade de realizar diferentes tarefas de um projecto um projecto de Aprendizagem Computacional de forma automatizada e sem assistência humana, garantindo que bons resultados são atingidos. Contudo, apesar da sua definição ser relativamente simples, a sua implementação é complexa. A automatização de diversas tarefas que têm como base o conhecimento humano, torna-se uma tarefa bastante desafiante, particularmente, se for tido em conta o grande nível de intuição que um cientista de dados necessita de ter para realizar diversas decisões.

Apesar da dificuldade prática relativa a AutoML, esta área tem crescido imenso com os diversos artigos publicados anualmente que procuram novas técnicas e abordagens para resolver o problema. Desta forma, este projecto procura avaliar as competências das abordagens de AutoML, assim como, explorar a integração dessas competências noutros sistemas, mais em concreto num assistente virtual.

1.2 Objectivos

O estágio realizado, sustentado por uma forte componente de investigação, focou-se na análise das técnicas de AutoML e na consequente implementação de um módulo capaz de realizar as diferentes tarefas de Aprendizagem Computacional de forma automatizada. Foram ainda explorados cenários onde o conceito podia ser aplicado, resultando na implementação de um assistente virtual que permite a execução dos métodos do módulo AutoML com base em comunicação em linguagem natural.

O primeiro objectivo passou pela avaliação das técnicas de AutoML actuais, descritas no Capítulo 2, seguido pela implementação de um módulo AutoML. Durante a análise das diferentes técnicas e ferramentas de AutoML disponíveis, foi notado que poucas ofereciam a automatização completa das fases mais importantes de um projecto de Aprendizagem Computacional: limpeza inicial de dados, engenharia de atributos e selecção de um algoritmo. Com base nesta avaliação, decidiu-se que o módulo implementado iria automatizar estas três fases. As abordagens das fases de limpeza inicial de dados e de engenharia de atributos foram pensadas e implementadas no decorrer do projecto e, a fase de selecção de um algoritmo eficaz recorre uma ferramenta externa que já automatiza esta tarefa. Neste sentido, as contribuições mais significativas são relativas à automatização da limpeza de dados e da fase de engenharia de atributos.

O segundo objectivo é uma prova de conceito, no que diz respeito à atribuição das competências de um cientista de dados a um assistente virtual. A recente investigação em torno de AutoML, descrita no Capítulo 2, levou a que existissem uma série de novos cenários onde o conceito pode-se ser aplicado. Um destes cenários foi a criação de um assistente virtual capacitado para executar o trabalho de um cientista de dados. O assistente virtual implementado serve-se do módulo AutoML para permitir a execução de diversas funcionalidades do módulo com recurso à comunicação em linguagem natural. Desta forma, é possível o utilizador falar com o assistente virtual, pedindo-lhe para executar as fases implementadas do módulo AutoML, como se estivesse a falar com um cientista de dados.

Com o projecto desenvolvido, é possível implementar projectos de Aprendizagem Computacional de forma simples e intuitiva. O utilizador pode recorrer directamente ao módulo AutoML, caso tenha conhecimentos de programação, ou pode utilizar o assistente virtual para executar as mesmas operações. Em síntese, as vantagens deste trabalho são:

- Reduz o obstáculo de não ter conhecimento extenso na área de Aprendizagem Computacional. O módulo AutoML retira o conhecimento preciso para desenvolver um sistema de Aprendizagem Computacional dando oportunidade a equipas não especializadas de tirar benefícios. A título de exemplo, uma equipa de biólogos poderá utilizar os dados que possui de modo a obter resultados valiosos.
- Quebra a barreira que impede que um individuo, sem experiência em programação, possa implementar projectos de Aprendizagem Computacional. Por norma, projectos de Aprendizagem Computacional têm uma componente forte de programação, limitando ainda mais esta área. Com o assistente virtual, é possível tirar benefícios de Aprendizagem Computacional sem a necessidade da componente de programação.
- É uma mais valia para a equipa da *Critical Software* pois permite a criação de protótipos de forma mais rápida. Com a solução implementada, é possível verificar rapidamente se um determinado problema pode ser resolvido com Aprendizagem Computacional. Se for o caso, os colaboradores poderão utilizar os algoritmos seleccionados pelo módulo e começar o seu trabalho com uma base sólida, em vez de começar do zero.
- Reduz os custos associados a um projecto de Aprendizagem Computacional visto que permite criação de soluções rapidamente, diminuindo o tempo de desenvolvimento.
- Retira algum esforço ao cientista de dados possibilitando que este se foque em tarefas mais críticas em vez de tarefas repetitivas como por exemplo a limpeza de dados.

Sumariando, os objectivos do projecto passaram pela implementação de um módulo AutoML e pela integração do módulo desenvolvido num assistente virtual. Estes dois objectivos possibilitam derrubar duas barreiras no que toca à implementação de projectos de Aprendizagem Computacional: não ter experiência em Aprendizagem Computacional e não ter experiência em programação, respectivamente.

1.3 Estrutura do documento

Este documento está dividido em sete capítulos e segue a seguinte estrutura:

- **Capítulo 2 -** este capítulo explora as diferentes técnicas utilizadas na área de AutoML. Começa por uma introdução acerca das dificuldades de Aprendizagem Computacional. De seguida, é introduzido o conceito de AutoML e é explicado, para cada fase, os progressos que foram feitos para automatizar cada fase;
- **Capítulo 3 -** este capítulo serve o propósito de descrever as funcionalidades das duas ferramentas desenvolvidas para que, no capítulo seguinte, seja possível detalhar a forma como estas funcionalidades foram implementadas. Primeiro, é feita uma descrição do módulo AutoML, seguida por uma descrição do assistente virtual. A seguir, são apresentados os requisitos do projecto assim como a arquitectura do módulo AutoML;
- **Capítulo 4 -** neste capítulo detalhamos os aspectos das abordagens utilizadas e a implementação das mesmas. Primeiro, são dedicadas uma secção para cada submódulo implementado - submódulo de limpeza de dados, submódulo de engenharia de atributos e submódulo de selecção do modelo. A seguir, os detalhes da implementação relativa ao assistente virtual são descritos;
- **Capítulo 5 -** neste capítulo são apresentados os testes que foram realizados para validar tanto o módulo AutoML como o assistente virtual;
- **Capítulo 6 -** este capítulo começa por descrever a metodologia de desenvolvimento utilizada pela equipa onde o estagiário foi inserido. De seguida, são apresentados os diagramas de *Gantt* relativos ao primeiro e segundo semestres;
- **Capítulo 7 -** por último, este capítulo conclui o relatório. Primeiro, são apresentadas as conclusões em relação ao trabalho proposto, sumariando todos os aspectos relativos ao mesmo. De seguida, são sugeridas ideias para iterações futuras das ferramentas implementadas

Esta página foi intencionalmente deixada em branco.

Esta página foi intencionalmente deixada em branco.

Capítulo 2

Conhecimento Prévio e Estado da Arte

Aprendizagem Computacional é uma área de Inteligência Artificial que se foca em desenvolver sistemas capazes de aprender e identificar padrões nos conjuntos de dados com o intuito de realizar uma determinada tarefa. O crescimento e a evolução desta área, proporcionado pelo aumento do poder computacional e da quantidade de dados disponíveis, fez despoletar um grande interesse na comunidade científica e no mundo empresarial. Investigadores começaram a explorar diferentes técnicas e abordagens e, as empresas verificaram que Aprendizagem Computacional podia trazer valor ao seu negócio. Contudo, o processo de construção de algoritmos de Aprendizagem Computacional, capazes de gerar valor é complexo. Esta complexidade advém maioritariamente por se tratar de um processo iterativo e cumulativo de possíveis erros, exigindo profissionais experientes para que as diferentes escolhas sejam tomadas da melhor forma.

Para desenvolver correctamente um projecto de Aprendizagem Computacional, é necessário passar por um conjunto de etapas. Primeiro, o cientista de dados realiza uma análise exploratória de dados de forma a analisar as características principais do conjunto de dados. De seguida, precisa de trabalhar os dados de modo a extrair informação de qualidade dos mesmos. Por fim, necessita de escolher algoritmo de aprendizagem e configurar os respectivos parâmetros. Apesar de cada etapa conseguir ser resumida em poucas palavras, cada uma destas apresenta os seus próprios desafios técnicos que tornam o desenvolvimento de um projecto de Aprendizagem Computacional mais carecedor de profissionais com experiência na área.

Em consequência da complexidade e do crescimento notado, existiu uma necessidade de ferramentas que tornam a implementação de projectos de Aprendizagem Computacional mais acessível e escalável. Contudo, apesar desta necessidade ter sido parcialmente preenchida com ferramentas como o *tensorflow*[1], a utilização eficaz das mesmas requer ainda conhecimento da área de Aprendizagem Computacional, do domínio do problema e da ferramenta em si.

As complicações e limitações associadas a um projecto de Aprendizagem Computacional, levaram a que investigadores pensassem em formas de automatizar as diferentes fases críti-

[1] https://github.com/tensorflow/tensorflow

cas neste tipo de projectos, criando uma nova área. Esta nova área, ficou conhecida como Aprendizagem Computacional Automatizada (AutoML)

Neste capítulo são descritos os diferentes conceitos em torno de AutoML e é revista a literatura relativa ao mesmo. A secção 2.1 começar por introduzir os conceitos de Aprendizagem Computacional e AutoML de modo a dar uma visão geral sobre os mesmos. De seguida, a subsecção 2.1.1 apresenta as diferentes técnicas que foram estudadas para automatizar as etapas do processo de Aprendizagem Computacional. Esta subsecção está dividida pelas diversas etapas do processo de Aprendizagem Computacional, começando pelo tratamento dos dados até à construção do modelo. Na subsecção 2.1.2 é apresentada uma estrutura geral das abordagens de AutoML com o intuito de resumir a subsecção anterior. Na subsecção 2.1.3 são descritas as principais ferramentas que pretendem trazer AutoML de forma acessível e simples. Finalmente, na subsecção 2.1.4 são apresentados os diferentes desafios futuros em relação a AutoML.

2.1 AutoML

AutoML é uma sub-disciplina cujo conceito é bastante simples e intuitivo, porém a sua implementação é complexa e traz grandes desafios. De forma genérica, podemos caracterizar AutoML como uma área científica que tem como objectivo automatizar o processo de um projecto de Aprendizagem Computacional.
Tal como foi referido no início do capítulo, a recente atenção e investigação em torno de AutoML, advém principalmente dos problemas e limitações associados com Aprendizagem Computacional. Abordando estas limitações de forma mais detalhada, constata-se que interesse pode, então, ser derivado dos seguintes factores:

- `Vasta aplicação de Aprendizagem Computacional na indústria:` empresas das diversas industrias, reconhecem que podem beneficiar de Aprendizagem Computacional. Diariamente, produzem grandes quantidades de dados e, por sua vez, podem utilizar esses dados para acrescentar valor ao seu negócio;

- `Escassez de cientistas de dados e especialistas:` de acordo com relatório realizado pelo *LinkedIn*[2], uma rede social de negócios e de emprego, o trabalho de cientistas de dados foi o mais promissor em 2019, crescendo cada vez mais o número de vagas. Porém, não existem suficientes profissionais experientes para satisfazer a procura;

- `Complexidade de um projecto de Aprendizagem Computacional:` um projecto de Aprendizagem Computacional é complexo e a experiência do cientista de dados é fundamental. Este, ao longo do projecto irá tomar várias decisões com base no seu conhecimento, intuição e experiência na área. Naturalmente, estas decisões são propensas a erros que se propagam ao longo das tarefas seguintes. A complexidade não só limita esta área a pessoas experientes como também leva a que sejam feitos erros desnecessários que podem comprometer a performance esperada;

- `Custos associados a um projecto de Aprendizagem Computacional:` são várias as despesas associadas a um projecto deste género. Os salários dos cientistas de dados, os recursos computacionais necessários e o tempo que leva a desenvolver um projecto são alguns dos custos associados;

[2] `https://blog.linkedin.com/2019/january/10/linkedins-most-promising-jobs-of-2019`

- `O aumento do poder computacional e da quantidade de dados:` a área de Aprendizagem Computacional começou por ser explorada no século passado. No entanto, não tinha grande relevância devido às grandes limitações no poder de processamento das máquinas da época. Com os avanços recentes na área das tecnologias, esta limitação foi-se tornando cada vez mais pequena. Hoje, temos máquinas capazes de processar grandes quantidades de dados, revelando-se essencial para o crescimento do interesse nesta área;

Devido às razões acima descritas e à constante necessidade de tornar Aprendizagem Computacional mais acessível, o conceito de AutoML, em inglês *Automated Machine Learning*, tem ganho bastante apreço. Quanming Yo et al. [3] refere que o conceito abrange dois outros conceitos: `Automatização` e `Aprendizagem Computacional`.

Automatização que se foca em realizar diversos processos e tarefas sem a necessidade de intervenção humana. E, Aprendizagem Computacional que possibilita que sistemas tenham a habilidade para aprender e aperfeiçoar através da experiência. São um conjunto de métodos que detectam padrões e relações nos dados e, utilizam esse conhecimento para realizar previsões. Estes métodos recebem um conjunto de dados e tentam criar uma relação, com recurso a funções matemáticas, entre os dados de entrada e os dados de saída.

O processo de Aprendizagem Computacional envolve um conjunto de fases onde o cientista de dados tem de fazer uma série de decisões acerca do tipo de dados e dos algoritmos a utilizar. As fases de um processo típico de Aprendizagem Computacional, exemplificados na Figura 2.1, podem ser sumariados da seguinte forma:

1. `Formulação do Problema:` antes de partirmos para a resolução do problema temos que definir bem o problema que queremos tratar. É essencial perceber que tarefa pretendemos resolver para que nas fases posteriores seja possível atacar o problema de forma directa.

2. `Recolha dos dados:` com o problema definido, é necessário recolher os dados que nos ajudem a realizar a tarefa da melhor forma. É um passo crítico que envolve a recolha dos dados relevantes de várias fontes. A qualidade e a quantidade dos dados vai afectar severamente a performance do modelo, pois são os dados que fornecem a informação ao modelo acerca da tarefa a realizar.

3. `Compreensão dos dados:` antes de trabalharmos os dados é necessário estudar e compreender os dados recolhidos. Nesta fase são utilizadas várias ferramentas de visualização e técnicas de análise exploratória de dados de modo a perceber as principais características dos dados.

4. `Tratamento dos dados:` os dados recolhidos podem estar num formato incorrecto, não organizados ou com valores em falta. Deste modo, é necessário realizar um conjunto de operações tais como: eliminar *outliers* e valores duplicados, tratar de dados em falta, normalizar todas as dimensões, e outras operações que digam respeito ao problema em si (por exemplo, balanceamento de classes). Depois de ter os dados organizados é fundamental extrair o máximo de informação possível dos dados. Para tal, existem diversos métodos de extracção, transformação e selecção de atributos. Com esta fase completa, teremos um conjunto de dados mais estruturado que nos dá o máximo de informação acerca do objecto que pretendemos prever.

5. `Construção do Modelo:` neste passo escolhemos o algoritmo e determinamos os valores dos respectivos parâmetros com base nos nossos dados e no que queremos prever. Existem numerosos algoritmos (modelos) que atingem bons resultados, contudo não existe um único algoritmo que é melhor que os outros todos. Habitualmente, a escolha do algoritmo é feita através da experiência e conhecimento do cientista de dados.

6. `Treino do Modelo:` neste passo o modelo utiliza os dados para melhorar incrementalmente a habilidade de aprender e prever. Primeiro não sabe nada acerca dos dados, mas com o tempo vai-se ajustando de modo a obter a melhor performance possível.

7. `Avaliação:` nesta fase testamos o modelo com o conjunto de teste. Este conjunto de dados nunca foi visto pelo modelo. Permite então então verificar como é que o modelo se comportará perante novos dados. Este conjunto de dados é representativo de dados no mundo real. Se o modelo obter uma boa eficiência, podemos colocá-lo em produção, caso contrário será necessário analisar o que está a correr mal e corrigir.

8. `Deploy e Manutenção:` finalmente, o modelo criado deve ser integrado no mundo real. O modelo vai passar a ser utilizado por utilizadores reais que vão oferecer *feedback* à equipa. Como tal, a manutenção do modelo é de extrema importância.

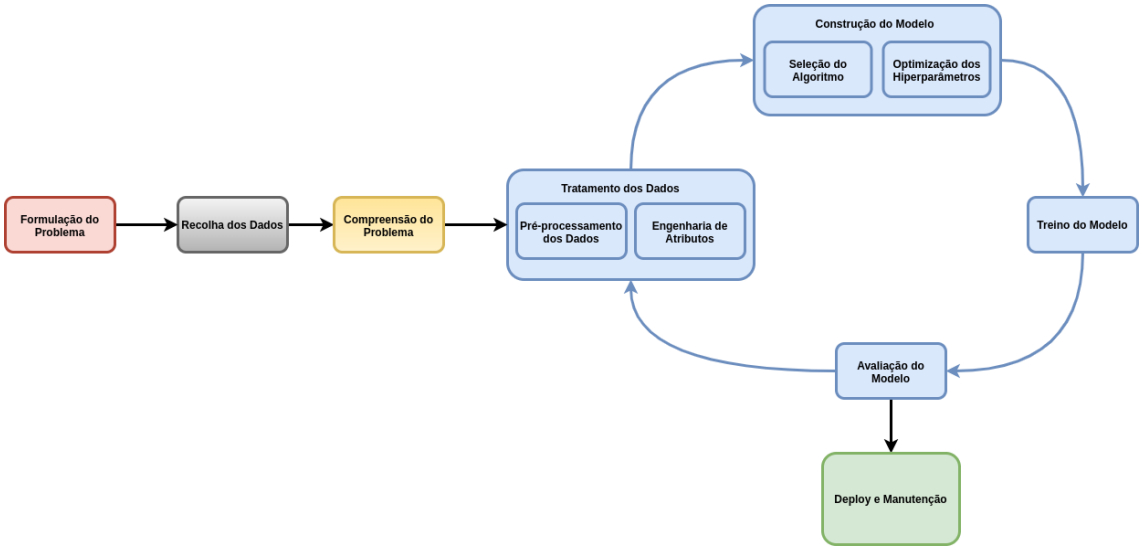


Figura 2.1: Procedimento típico em Aprendizagem Computacional

Como é possível verificar através da observação da Figura 2.1 e da pormenorização detalhada das fases anteriormente descrita, é um processo bastante demorado em que o cientista de dados tem de lidar com um grande número de escolhas complexas que vão desde o processamento dos dados à escolha do melhor algoritmo. Devido a esta complexidade, a ideia de automatizar o processo de Aprendizagem Computacional, i.e AutoML, tem ganho bastante estima. Não só pela comunidade científica mas também pelo mundo empresarial que pretende diminuir a complexidade e os custos associados a um projecto deste género.

Juntando as definições de Automatização e de Aprendizagem Computacional, podemos verificar que AutoML pretende fazer as melhores escolhas nas diferentes fases, de modo a obter uma boa performance (do ponto de vista da Aprendizagem Computacional), assim como, obter esta performance sem a intervenção humana (do ponto de vista da automatização). De uma perspectiva mais clara, AutoML pretende que parte ou todos os

procedimentos no processo de Aprendizagem Computacional sejam automatizados.

No artigo realizado por Quanming Yo et al. [3], AutoML é definido como um problema de optimização. Os autores do artigo mencionado descrevem que o objectivo é atingir a máxima performance possível mas com duas restrições: sem assistência humana, e com um limite no esforço computacional. Reunindo as definições acima descritas, os autores apresentam uma definição mais geral que engloba todas as condições necessárias:

Definição 1. *AutoML pode ser caracterizado como o conjunto de mecanismos para automatizar as etapas de Aprendizagem Computacional, com o objectivo de obter a melhor performance possível, sujeito a: sem intervenção humana e limite nos recursos computacionais e no tempo.*

A automatização do processo de Aprendizagem Computacional oferece diversas vantagens, tais como:

- Implementação de projectos de aprendizagem computacional sem conhecimento extenso na área;
- Reduzir o tempo despendido na resolução de tarefas, permitindo ter um algoritmo confiável em poucas horas ou dias;
- Criação de protótipos com o intuito de verificar se um determinado problema pode ser resolvido com aprendizagem computacional;
- Redução dos custos associados a projectos de Aprendizagem Computacional, conseguindo assim a obtenção resultados de forma mais rápida e com menos dispendiosa;
- Indivíduos especializados não precisam de perder tempo em tarefas repetitivas, podendo assim concentrarem-se em tarefas mais criativas que dêem mais valor à empresa;

Contudo, tratando-se de uma matéria relativamente recente, precisa de muito estudo para termos aplicações totalmente autónomas, pois apesar de existirem implementações bem sucedidas, não existem algoritmos que conseguem atingir uma boa performance em todos os problemas. Sendo que cada problema tem dados diferentes e objectivos ou finalidades diferentes, cada um exige uma configuração cuidadosamente pensada e trabalhada.

O objectivo será que um dia, todo o processo, de uma ponta a outra, seja automatizado. Para tal, vários algoritmos para cada um dos passos foram propostos. De seguida iremos falar de cada um destes passos e perceber o que já foi feito.

2.1.1 O que automatizar e como automatizar

Nesta secção irão ser apresentadas as diferentes abordagens para automatizar o processo de Aprendizagem Computacional. Como já foi visto, é um processo iterativo com um conjunto de passos que envolvem a escolha dos melhores algoritmos de modo a obter a performance desejada num determinado problema.

Uma das formas de automatizar o processo de Aprendizagem Computacional passa pela divisão do mesmo. Isto é, todas as fases de Aprendizagem Computacional são automatizadas

individualmente e no fim é feito um agrupamento das técnicas de cada uma das fases para automatizar o processo inteiro. Neste sentido, a presente secção aborda a automatização de cada uma das fases individualmente.

Primeiro iremos abordar o tratamento dos dados. Começamos com uma breve introdução acerca dos problemas relativos ao processamento de dados e de seguida são explicadas as técnicas existentes para o **pré-processamento e engenharia de atributos**. A seguir, com os dados já tratados passaremos para a construção do modelo. Nesta fase iremos apresentar as técnicas implementadas para a automatização da escolha do modelo que mais se adequa a um determinado problema e a respectiva configuração dos hiperparâmetros.

2.1.1.1 Tratamento dos Dados

Os dados são um dos aspectos mais importantes de um projecto de Aprendizagem Computacional. São os dados que dão informação ao modelo para que este possa ter a performance desejada. Desta maneira, o processo de recolha e tratamento dos dados é de extrema importância. Simultaneamente, a fase de tratamento dos dados é a que impõe mais desafios e consome mais tempo. De acordo com o estudo realizado pela *Cognilytica Research* [5], tarefas relacionadas com o tratamento dos dados ocupam cerca de 80% do tempo utilizado em projectos de Aprendizagem Computacional.
Analisando detalhadamente o estudo, cujos resultados estão expostos na Figura 2.2, é possível constatar que, metade do tempo é logo empregue no tratamento inicial dos dados, mais concretamente na limpeza dos dados e na atribuição das classes aos dados. De seguida, cerca de 30% do tempo é consumido por tarefas de transformação de dados que pretendem extrair o máximo de informação possível dos atributos.

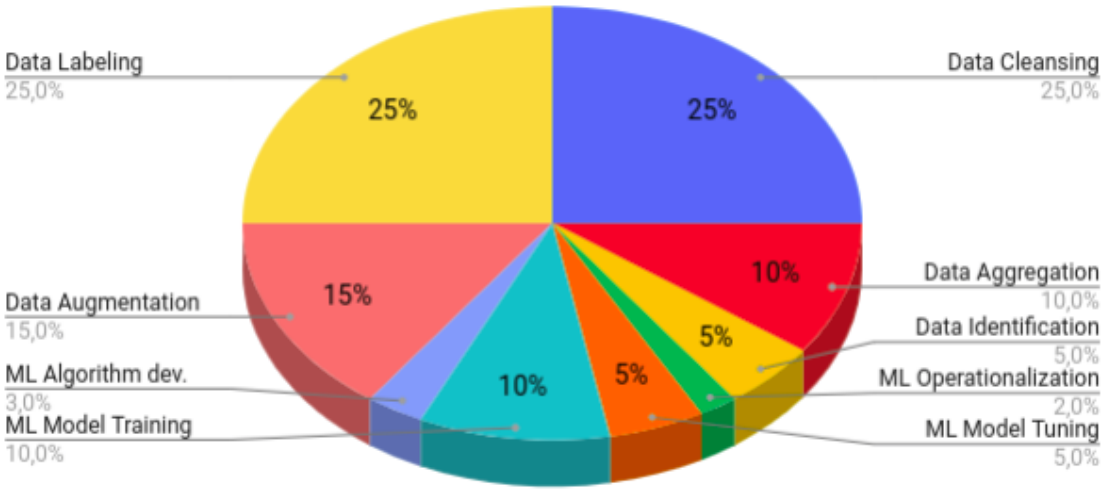


Figura 2.2: Percentagem de tempo alocado nas diversas fases de um projecto de Aprendizagem Computacional

A automatização desta fase seria um avanço considerável uma vez que o tratamento dos dados é de extrema importância e é esta fase que consome mais tempo ao longo de um projecto de Aprendizagem Computacional.

Esta fase pode ser dividida em duas: o pré-processamento dos dados e a engenharia de atributos. Comecemos pela análise do pré-processamento dos dados, na qual será descrito que tipos de defeitos podem ter os dados e que tipos de técnicas foram estudadas para contrariar esses defeitos.

Pré-Processamento

Antes de trabalhar sobre os dados e extrair informação, é fundamental organizar os mesmos e remover quaisquer erros que possam interferir na tarefa de previsão.
Existem diversos problemas e desafios em relação aos dados:

- `Falta de dados` - um dos maiores desafios é obter dados que nos dêem informação relevante acerca do problema. Apesar de cada vez termos mais sistemas, dispositivos e indivíduos a produzir quantidades absurdas de dados existem vários problemas na recolha dos mesmos;

- `Formatos incompatíveis` - os dados são recolhidos de diversas fontes e é necessário juntar os mesmos. Portanto, é preciso converter o formato dos dados para que estes fiquem consistentes;

- `Dados não tratados` - outro grande desafio é lidar com dados que podem estar incompletos, com valores duplicados ou com valores anormais (*outliers*).Todos estes erros levam a que os dados representem de forma errada o problema a resolver dificultando assim a tarefa de previsão;

- `Dados mal classificados ou desequilibrados` - os dados podem estar mal classificados, i.e, determinadas classes podem estar mal atribuídas, ou podem estar desequilibrados, i.e, termos mais valores de uma determinada classe;

Todos estes problemas diferem de conjunto de dados para conjunto de dados, desta forma é difícil automatizar um processo que, por exemplo, verifique que dados estão mal classificados. O pré-processamento é uma fase inicial importante, contudo depende profundamente do projecto e do tipo de dados no qual estamos a trabalhar. Certas tarefas estão dependentes do conhecimento do domínio do problema e consequentemente é difícil a automatização das mesmas.
Existem contudo, uma variedade de operações que são frequentemente aplicadas e que podem, de certa forma, ser automatizadas. Entre elas, são destacadas as seguintes: formatação (verificar que todas as variáveis estão no mesmo formato), tratamento de valores em falta (preencher os valores em falta com a respectiva média, moda ou mediana, dependendo do tipo dos atributos), transformação da representação (por exemplo, converter um atributo numérico num atributo categórico), tratamento de valores absurdos e inconsistências (remover ou substituir por outro valor), tratamento de valores duplicados e redundantes, balanceamento de classes e normalização (para que os dados fiquem todos na mesma escala).

A literatura relativa à automatização do pré-processamento dos dados é pouco extensa. A automatização desta fase não obteve grande atenção por parte dos investigadores porque efectivamente difere de problema para problema, estando muito dependente do domínio do problema. Contudo, existem duas abordagens interessantes que serão descritas de seguida.

Active Clean [6], é uma ferramenta que ajuda um analista de dados na tarefa de limpeza dos dados. Este sistema apenas foca-se na identificação de dados potencialmente sujos, deixando a parte da limpeza para o utilizador. O utilizador fornece o conjunto de dados e um algoritmo de classificação (e.g *Support Vector Machine*). De seguida, o sistema divide os dados em subconjuntos e iterativamente treina o algoritmo de classificação identificando dados que possam conter erros. A cada iteração, o algoritmo identifica potenciais erros e pede ao utilizador para limpar esses erros. Depois da limpeza do subconjunto, o algoritmo de classificação é treinado de novo a fim de verificar se a limpeza foi bem sucedida. Este sistema apenas automatiza a parte da identificação dos erros, deixando a tarefa de limpeza para o utilizador. Por um lado, é um sistema incompleto pois não automatiza a limpeza de dados toda. Por outro, ao deixar a limpeza de dados nas mãos do utilizador, certifica-se que não coloca automaticamente erros nos dados. É na parte da limpeza de dados que os dados vão ser alterados, e qualquer alteração automática e sem vigilância de um analista poderá inserir erros nos dados que levarão a um desempenho inferior ao esperado.

As abordagens clássicas para a limpeza dos dados baseiam-se em estatística ou Aprendizagem Computacional, não dando particular atenção ao contexto dos dados e do problema. Xu Chu et al. [7] afirmam que estas abordagens têm um limite na precisão de limpeza, e são normalmente aperfeiçoadas com recurso a especialistas para resolver ambiguidade. Com base nesta problema, foi proposto, pelos autores do artigo mencionado, um novo sistema de limpeza de dados que envolve humanos para validar e verificar os padrões encontrados no conjunto de dados. Utiliza Bases de Conhecimento (*Knownledge Bases*), tais como YAGO[3] para descobrir padrões e relações entre os diferentes atributos, com base no seu contexto. Seguidamente, verifica quais os dados que possam estar incorrectos. Quando abordagens automáticas não conseguem resolver ambiguidades, envolve humanos para validar os padrões descobertos e corrigir os dados que possam estar incorrectos.

Engenharia de Atributos

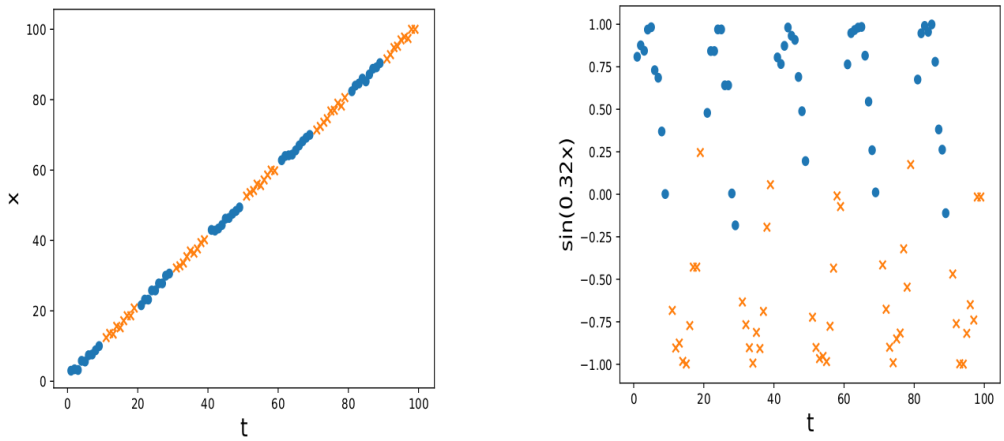
Com os dados limpos e livres de erros, precisamos de criar atributos que consigam fornecer ao modelo informação relevante acerca da tarefa a realizar. Um algoritmo de Aprendizagem Computacional tenta perceber iterativamente os padrões presentes nos dados e a sua representação. Como tal, é fundamental que os padrões estejam bem visíveis e que os dados consigam fornecer informação suficiente para atingirmos a performance desejada.

O processo da alteração da representação dos atributos com o intuito de obter melhores resultados é chamado Engenharia de Atributos. Esta etapa fulcral, transforma a representação e selecciona os dados de modo a facilitar a tarefa ao modelo, reduzindo assim o erro da previsão. Pode ser dividia em duas subetapas: `transformação de atribtuos` e `selecção de atributos`.

Transformação de atributos altera o conjunto de dados, criando novos atributos que se adequam melhor ao modelo. Segundo Udayan Khurana et al. [8], envolve a transformação da representação dos dados, aplicando funções matemáticas aos mesmos, para uma representação que separe melhor os dados e facilite a tarefa de previsão ao modelo. Na

[3] https://github.com/yago-naga/yago3

Figura 2.3, cujos gráficos foram retirados do artigo acima mencionado, é possível verificar que a aplicação da função $\sin 0.32x$ aos dados originais resulta na separação mais visível e explicita das duas classes que estão caracterizadas por cores diferentes.
Com recurso a diversas operações, sejam elas operações que alterem a escala dos dados, como por exemplo normalização, ou operações que criem novos atributos, como por exemplo a soma entre dois atributos, é possível criar um conjunto de dados mais robusto que contribua para que o algoritmo de aprendizagem consiga obter melhores resultados.



(a) Dados Originais

(b) Dados Transformados

Figura 2.3: Ilustração de diferentes opções de representação

Selecção de atributos tem como objectivo descartar os atributos que sejam redundantes ou irrelevantes para a resolução do problema. Na subetapa anterior são criados inúmeros atributos, muitos dos quais não são úteis ou relevantes. Neste sentido, esta subetapa foca-se em seleccionar um subconjunto de atributos a partir do conjunto original, reduzindo assim o tempo de treino e o sobre-ajuste do modelo (*overfitting*). Este último, é um factor importante no que toca à capacidade de generalização do modelo e é frequentemente causado pela 'maldição da dimensionalidade'. Este conceito, diz respeito ao facto de o conjunto de dados ter demasiados atributos causando dificuldades no treino e má performance do modelo. Com esta subetapa, o conjunto de dados é composto apenas pelos atributos que contêm mais informação acerca da variável a prever.

A fase de Engenharia de Atributos é realizada com base na intuição, experiência e conhecimento do domínio do problema. Este último, é bastante importante nesta fase, pois o conhecimento do domínio do problema influencia directamente a escolha dos atributos relevantes. Por exemplo, numa tarefa de prever quantos passageiros vai ter um comboio num determinado dia da semana, podemos ter um atributo no conjunto de treino que nos diz qual é o dia da semana da observação. Para o cientista de dados, este atributo é bastante importante, pois ele sabe que provavelmente à sexta-feira e ao domingo o número de passageiros vai ser mais elevado comparando com os outros dias. Devido à dependência do conhecimento do domínio do problema e da intervenção humana, torna-se um processo de difícil automatização.

A investigação relativa à automatização desta fase foca-se na construção e selecção de atributos capazes de fornecer informação ao modelo. Certas abordagens utilizam o conhecimento do domínio do problema e outras abordagens utilizam técnicas de Aprendizagem

Profunda (*Deep Learning*) para prever que tipos de operações de engenharia de atributos devem ser aplicadas aos dados. De seguida, serão descritas três abordagens.

Investigadores da Universidade da *Califórnia, Berkeley* desenvolveram uma ferramenta, *ExploreKit* ([9]), para automatizar o processo de geração de atributos. Segundo Gilad Katz et al., o objectivo é "gerar um conjunto grande de atributos candidatos, combinando informações dos atributos originais, com o objectivo de maximizar o desempenho de acordo com métricas seleccionadas pelo utilizador". É um processo iterativo, onde, em cada iteração, são efectuados três passos: geração de atributos candidatos, classificação dos atributos candidatos e selecção dos atributos candidatos. Na primeira fase, são aplicados diversos operadores ao conjunto de atributos de modo a gerar novos atributos candidatos. Estes operadores podem ser aplicados a atributos isolados ou a conjuntos de atributos. Posteriormente, um classificador previamente treinado, classifica cada atributo candidato. Por fim, são avaliados todos os atributos candidatos, e aqueles que obterem uma avaliação superior a um limite predefinido são seleccionados para a próxima iteração.

No artigo produzido por Fatemeh Nargesian et al. [10], é apresentada uma abordagem para automatizar a tarefa de Engenharia de Atributos para problemas de classificação. A ideia é recorrer a experiências passadas de Engenharia de Atributos e beneficiar da capacidade de redes neuronais preverem certos valores com base nessas experiências. Para tal, treinaram um conjunto de redes neuronais com vários conjuntos de dados. Estas redes têm como objectivo prever que transformações trazem impacto na performance de classificação. As redes neuronais, previamente treinadas, recebem como *input* o conjunto de dados e prevêem que tipo de transformações devem ser aplicadas ao conjunto de dados. Cada uma das redes neuronais corresponde a uma única transformação. Segundo os autores do artigo referenciado, este método é eficaz, consumindo menos recursos computacionais e menos tempo.

AutoLearn [11], uma investigação realizada por Ambika Kaul et al., é outra abordagem à automatização da construção e selecção de atributos. É um algoritmo orientado a dados e por isso, não requer nenhum conhecimento do domínio do problema. Descrito pelos investigadores como um algoritmo baseado em regressão, é composto por quatro fases: Pré-processamento, Mineração de Atributos Correlacionados, Geração de Atributos e Selecção de Atributos. Primeiro, filtra os atributos originais de modo a obter um conjunto mais pequeno, descartando os atributos com um ganho de informação baixo. De seguida, verifica a correlação entre pares de atributos com o objectivo de eliminar os atributos dependentes. Já com os atributos seleccionados, treinam algoritmos de regressão com pares de atributos, sendo que um atributo é o valor de entrada e o outro é o atributo a prever. Com base nos valores previstos, criam dois tipos de atributos. O primeiro tipo são os valores previstos pelo algoritmo de regressão e o segundo tipo é a diferença entre o valor previsto com o valor real. Por último, selecciona os melhores atributos com base no ganho de informação e na estabilidade dos mesmos.

Por último, no Massachusetts Institute of Technology (MIT) verificaram que a maioria dos dados utilizados em projectos Aprendizagem Computacional são estruturados e relacionais. Constataram que os atributos são construídos a partir de relações entre os dados e muitas vezes são utilizados os mesmos tipos de operações entre diferentes conjuntos de dados e problemas. Notaram ainda que novos atributos são por norma gerados utilizando atributos anteriormente construídos.
Assim sendo, James Max Kanter et al. [12] desenvolveram um algoritmo para gerar atributos automaticamente para conjuntos de dados relacionais. O algoritmo aplica sequen-

cialmente funções matemáticas com base nos relacionamentos entre os dados. Recebe um conjunto de entidades e as tabelas associadas às entidades. De seguida, a partir das entidades, das tabelas e dos relacionamentos, define um conjunto de operações matemáticas que são aplicadas ao nível da entidade e ao nível relacional. Ao nível da entidade, as operações são aplicadas a uma ou mais colunas da mesma tabela, como por exemplo normalização. Ao nível relacional, as operações são aplicadas entre várias tabelas a entidades que tenham relação pai/filho, como por exemplo o máximo de vendas(entidade) por cliente (entidade). Os investigadores decidiram dar o nome de *Deep Feature Synthesis* ao algoritmo, visto que agrega várias operações para criar atributos "profundos".

2.1.1.2 Construção do Modelo e Optimização dos Hiperparâmetros

Depois de preparar os dados e seleccionar os atributos, é necessário de definir o algoritmo de aprendizagem e configurar os seus hiperparâmetros. Do ponto de vista de AutoML, esta tarefa pode ser descrita como a automatização da escolha do modelo e dos respectivos hiperparâmetros.
Tradicionalmente, a escolha entre os diferentes modelos existentes e os respectivos hiperparâmetros é feita por tentativa erro e com base na experiência do cientista de dados, no tipo de dados e no problema a resolver. Existem vários tipos de modelos, e cada um destes modelos tem um conjunto diferente de hiperparâmetros. Não existindo um único modelo que seja melhor que todos os outros em todos os problemas, o cientista de dados tem de testar vários tipos de modelos e várias combinações de hiperparâmetros de modo a encontrar o que obtém melhor performance. Sendo esta tarefa crucial, pois é a ultima tarefa e dá-nos os resultados concretos do problema que temos que resolver, a sua automatização seria uma mais valia.

A escolha do modelo e configuração dos respectivos hiperparâmetros de forma automatizada é muitas vezes definida como Combined Algorithm Selection and Hyper-parameter (CASH), que pode ser visto como um problema de optimização. Radwa Elshawi et al. [2], descrevem o problema do seguinte modo:

Definição 2. *Dado um conjunto de algoritmos de Aprendizagem Computacional e um conjunto de dados dividido em dois subconjuntos: um de treino e outro de validação, o objectivo é encontrar um algoritmo e configurar os hiperparâmetros que obtêm o desempenho mais alto treinando no conjunto de treino e avaliando no conjunto de validação.*

Ou seja, o objectivo é encontrar um algoritmo e os respectivos hiperparâmetros, de entre várias combinações possíveis, que seja eficiente com base numa determinada métrica (por exemplo precisão). Este passo pode ser dividido em dois passos que se influenciam e dependem um do outro:

1. `Escolha do algoritmo:` para a selecção de modelos, Xin He et al. [4] afirmam que existem dois tipos de técnicas: selecção tradicional de modelos e técnicas de geração de arquitecturas de redes neuronais. O primeiro tipo foca-se na selecção de algoritmos tradicionais de aprendizagem computacional, como por exemplo, *Support Vector Machines, Decision Trees e Random Forests.* Neste caso, um espaço de procura, composto por diversos algoritmos já existentes, é percorrido com um determinado método de procura de forma a encontrar o algoritmo mais apropriado. O segundo tipo de técnicas, foca-se na geração de arquitecturas de redes neuronais sem assistência humana. Neste caso, existem dois conceitos importantes: Neural Architecture

Search (NAS) e Meta-Learning. Estes conceitos são muito populares na comunidade científica e, por isso, serão descritos com mais detalhe em seguida.

2. `Optimização dos Hiperparâmetros:` o próximo passo é a configuração dos hiperparâmetros do algoritmo de aprendizagem de modo a optimizar a performance do mesmo. Os hiperparâmetros são os parâmetros que podem ser definidos antes do treino. Em contraste, temos os parâmetros do modelo, que são ajustados pelo modelo durante a aprendizagem. A escolha destes hiperparâmetros pode ser feita por tentativa erro ou com recurso a métodos de optimização que procuram os hiperparâmetros que obtêm melhor performance.

Neural Architecture Search

NAS é o processo que visa automatizar a escolha do modelo, desenhando uma nova arquitectura neuronal sem assistência humana. É um sub conceito fundamental em AutoML que tenta gerar arquitecturas que maximizem a performance.
Com recurso ao conjunto de dados e com a especificação da tarefa, um algoritmo de NAS desenha arquitecturas neuronais compostas por diferentes camadas ligadas por diversas operações. O processo é composto três elementos: espaço de procura, método de procura e avaliação do arquitectura.

O espaço de procura inclui diferentes tipos de estruturas e operações. Este, define quais as estruturas e operações que podem ser utilizadas na construção da arquitectura. A arquitectura gerada vai conter uma combinação das estruturas presentes no espaço de procura ligadas com as operações.
De modo a encontrar a melhor arquitectura precisamos de definir um método de procura. Este método define como é que o espaço que procura é explorado. O espaço de procura é muitas vezes enorme, pelo que testar todas as combinações torna-se inviável. O método de procura une as diferentes estruturas com recursos às operações e trabalha em conjunto com o método de avaliação do modelo de modo a encontrar a melhor arquitectura.
Por fim temos a avaliação do modelo. Um algoritmo de NAS visa encontrar a arquitectura mais eficiente. Para tal, é necessário verificar se a arquitectura desenvolvida obtém boa performance com dados nunca antes vistos. Este passo é essencial para guiar o método de procura, de modo a que este gere arquitecturas cada vez melhores.

Uma abordagem intuitiva para avaliar a arquitectura seria treinar a arquitectura com o conjunto de treino e de seguida avaliar a arquitectura com o conjunto de validação. Contudo, esta abordagem torna-se impraticável devido ao tempo gasto a avaliar uma arquitectura. Dessa forma, foram propostos vários algoritmos para acelerar o processo. Xin He et al. [4] dividem as técnicas em três categorias:

- `Baixa Fidelidade:` quanto maior for a dimensão do conjunto de dados ou o tamanho do modelo, mais tempo irá demorar o treino e consequentemente a avaliação do modelo. Em função disso, esta abordagem caracteriza-se por diminuir a dimensão do conjunto de dados (utilizar um subconjunto ou, no caso de imagens, usar imagens com menos resolução) ou diminuir o tamanho do modelo (por exemplo, treinar com menos filtros por camada);
- `Transfer Learning:` tal como o nome sugere, o objectivo é transferir experiências anteriores de uma ou mais tarefas já realizadas. No início, as redes são treinadas

por longos períodos de tempo, mas com o passar do tempo, as tarefas tornam-se as mesmas e é possível transferir o conhecimento adquirido. A título de exemplo, é possível utilizar parâmetros ou pesos de arquitecturas que foram bem sucedidas;

- `Paragem Antecipada:` é uma abordagem simples já usada em Aprendizagem Computacional clássica. Baseia-se interrupção do processo se a configuração a avaliar obter um desempenho mau com o conjunto de validação ou atingir um limite predefinido em validação ou não apresentar melhorias significativas;

Recentemente, a investigação em torno de NAS tem-se focado no processo de procura e avaliação do modelo. Vários métodos foram propostos, sendo que a maioria aborda este problema com Reinforcement Learning (RL) ou métodos evolucionários.
De forma geral, no caso das abordagens utilizando RL, um agente gera novas arquitecturas com base no *feedback* recebido. As arquitecturas geradas são avaliadas e o desempenho obtido por cada arquitectura é utilizado para actualizar o agente para que este perceba se as estruturas e operações utilizadas favorecem ou não a performance do modelo. Com o tempo, o agente vai percebendo que tipo de combinações funcionam melhor.
No caso das abordagens que utilizam métodos evolucionários são aplicadas diversas operações genéticas aos indivíduos da população. A população é composta por várias arquitecturas que sofrem processos de cruzamento e mutação a fim de explorar novas arquitecturas. No fim de cada geração, são seleccionados um número de indivíduos (arquitecturas) que passam para a geração seguinte.

Estas duas abordagens são as mais utilizadas na literatura para resolver esta questão de NAS. Apesar de utilizarem abordagens semelhantes, os estudos sobre este conceito diferem na maneira de como aplicam RL ou métodos evolucionários. Para perceber as diferentes técnicas utilizadas, serão introduzidas investigações realizadas com base nestas abordagens.

A atenção em torno do conceito de NAS surgiu com o trabalho publicado pela equipa de investigação do *Google Brain*[4] [13] em 2016. Este artigo pioneiro na área, demonstrava uma nova forma de gerar e procurar arquitecturas neuronais. A ideia base e intuitiva é gerar arquitecturas, treiná-las e seleccionar a arquitectura que obtém melhor performance. Na base desta abordagem reside um controlador Recurrent Neural Network (RNN). A escolha deste tipo de rede neuronal traduz-se no facto de as previsões para novas arquitecturas serem condicionadas por previsões anteriores. O controlador RNN é treinado com RL de modo a gerar arquitecturas cada vez melhores com base na performance obtida no conjunto de validação.
O controlador gera uma arquitectura e de seguida treina-a de modo a obter a performance da mesma. Por fim, a performance é fornecida como recompensa ao controlador, que tenta maximizá-la com recurso a RL. Ou seja, a cada iteração, o controlador favorecerá a escolha de arquitecturas que tenham melhor performance, melhorando a pesquisa ao longo do tempo.

A técnica acima descrita, exige muito poder computacional e demora bastante tempo para gerar arquitecturas que obtenham bom desempenho. Como tal, os mesmos investigadores propuseram outra abordagem [14] que acrescenta a partilha de pesos entre arquitecturas diminuindo assim o tempo utilizado para o treino. Os investigadores do artigo mencionado, propõem Efficient Neural Network Architecture Search (ENAS), uma técnica para

[4] `https://ai.google/research/teams/brain/`

encontrar novas arquitecturas baseada em RL, em que uma rede neuronal (controlador) previamente treinada descobre arquitecturas procurando um subgrafo óptimo dentro de um grafo acíclico dirigido. A ideia central desta investigação, é que todos os grafos (arquitecturas) que um algoritmo de NAS gera podem ser vistos como subgrafos de um grafo maior. Assim sendo, o espaço de procura pode ser configurado com recurso a um grafo acíclico dirigido em que os nós representam as computações e os vértices representam o fluxo de informação. Desta forma, o objectivo passa a ser encontrar o melhor subgrafo. O controlador é então treinado para seleccionar um subgrafo que obtenha a melhor precisão no conjunto de validação. Esta investigação melhora a eficiência ao partilhar pesos por todos os nós filhos, evitando assim treinar cada nó do início. Esta técnica de partilha não só diminuiu o tempo de treino como melhorou a performance obtida.

Outra abordagem para procura de arquitecturas é utilizar Optimização Baseada em Gradiente. Um exemplo é o trabalho realizado por Hanxiao Liu et al. [15], que resultou na ferramenta Differentiable Architecture Search (DARTS). Esta abordagem propõe definir um novo espaço de procura. Este passa a ser continuo, permitindo assim a procura eficiente utilizando gradiente descendente. Utilizam a ideia do grafo dirigido acíclico apresentado anteriormente, mas com uma ligeira diferença. No trabalho apresentado anteriormente, o espaço de procura é composto por várias arquitecturas possíveis que diferem pelas camadas, operações e ligações, fazendo assim com que este espaço seja discreto. Neste trabalho, os autores passam o espaço para contínuo ao colocar várias operações candidatas em cada nó e com isso ter várias ligações entre os diferentes nós. Cada ligação, tem um peso associado que representa a probabilidade de um nó com uma determinada operação estar conectado a outro nó com outra operação. O objectivo passa então por encontrar os melhores pesos, que representam no fundo que ligações entre nós vão ocorrer numa determinada arquitectura. Esta procura é feita com recurso ao gradiente descendente.

Tal como vimos, métodos evolucionários também são utilizados para gerar arquitecturas de redes neuronais. Estes métodos são óptimos em resolver problemas de optimização, baseando-se nos mecanismos da evolução biológica. Os autores Jason Zhi Liang et al. [16] propõem a utilização deste tipo de algoritmos para a procura de arquitecturas - Learning Evolutionary AI Framework (LEAF). Primeiro, é criada uma população de complexidade mínima. Cada indivíduo da população é representado por um grafo. A cada geração, ocorre mutação, ao adicionar aleatoriamente um nó ou uma conexão entre nós, e ocorre cruzamento entre dois indivíduos. De seguida, os indivíduos são avaliados e é calculado o seu *fitness*. Por fim, o algoritmo foca-se na optimização dos hiperparâmetros. Com recurso a algoritmos evolucionários é possível gerar e avaliar configurações em que a cada geração, as piores configurações são descartadas e as melhores passam para a geração seguinte.

Algoritmos de NAS pretendem retirar a grande complexidade inerente à criação manual de redes neuronais. Existem vários tipos de operações e de ligações entre operações que servem propósitos diferentes e que são bons em problemas diferentes. Isto, torna a criação de redes neuronais muito complexa e morosa. Com NAS, estes problemas são transferidos para os algoritmos que tentam encontrar que ligações entre operações representam uma boa arquitectura. Contudo, até mesmo para computadores é uma tarefa complexa. O espaço de procura é gigante, levando a que existam milhões de combinações possíveis. Como tal, é necessário ter um algoritmo de procura eficiente e rápido. Neste momento a maioria das investigações foca-se em RL para resolver este problema.
Apesar de ser uma área relativamente recente, tem grande potencial. Já é possível gerar arquitectura eficazes, e agora o maior problema está no tempo em que se demora a gerar essas arquitecturas.

Meta-Learning

Meta-Learning é um conceito já estudado no âmbito de Aprendizagem Computacional clássica, e recentemente tem sido aplicada no domínio de AutoML. Actualmente, diversos estudos utilizam esta técnica para acelerar o processo de procura de arquitecturas neuronais. Por conseguinte, iremos abordar a razão pelo qual *Meta Learning* consegue ajudar na tarefa de automatizar a disciplina de Aprendizagem Computacional.
É um conceito simples, que se baseia na extracção de conhecimento de tarefas já realizadas, e na aplicação do mesmo em problemas futuros. Pode ser descrito como o processo de aprendizagem de experiências anteriores adquiridas durante a aplicação de vários algoritmos de aprendizagem em diferentes tipos de dados.

De notar que *Meta-Learning* e *Transfer Learning*, introduzido anteriormente, são dois conceitos que se sobrepõem. Ambos exploram a experiência adquirida em tarefas já realizadas. Alguns investigadores, dizem que *Transfer Learning* é um caso especial de *Meta-Learning*. No artigo elaborado por Yuqiang Chen et al. [3] os dois conceitos são distinguidos da seguinte forma:

> "Uma abordagem é considerada *Meta-Learning* se extrai conhecimento sobre problemas de aprendizagem e ferramentas (i.e, meta-atributos e performance) a partir de aprendizagem passada (...); caso contrário, é uma abordagem de *Transfer Learning* se usa directamente os resultados finais ou intermediários (i.e, melhor configuração) de aprendizagens passadas." (página 13)

As semelhanças entre diferentes tarefas e ferramentas podem ser avaliadas, o que permite a reutilização e transferência de conhecimento entre diferentes problemas.
O processo é simples e pode ser descrito da seguinte modo: primeiro são extraídas características, denominadas *meta-features*, tais como propriedades estatísticas dos dados ou configuração dos hiperparâmetros. De seguida, o *meta-learner* é treinado com base no conhecimento adquirido em conjunto com o conhecimento acerca do problema. Por fim, o *meta-learner* pode ser aplicado a problemas futuros.

Quanming Yao et al. [3] dividem as diversas técnicas de *Meta-Learning* em três classes: avaliação da configuração, geração de configurações e adaptação dinâmica de configurações.

`Avaliação de configurações` com recurso a *Meta-Learning*, pretende fazer com que o processo de avaliação seja mais rápido. Um dos passos que consome mais tempo em AutoML é a avaliação de uma determinada arquitectura. Para saber se essa arquitectura é promissora, é necessário treinar e avaliar a configuração. *Meta-learners* podem ser treinados para prever a performance ou aplicabilidade de uma certa configuração. Este processo acelera a avaliação de configurações pois é possível, a partir de um conjunto de configurações escolher a que irá obter um melhor desempenho.

`Geração de Configurações` com *Meta-Learning*, visa gerar configurações para o problema em questão com base em configurações bem sucedidas em problemas parecidos. Em vez de começarmos do zero, podemos pegar em configurações conhecidas e aplicar no nosso problema. O *meta-learner* utiliza as características do problema em conjunto com configurações já estudadas, e prevê configurações adequadas.

`Adaptação Dinâmica de Configurações` com *Meta-Learning*, permite que as configurações possam adaptar-se em função da alteração dos dados. Os dados podem mudar com o tempo, o que resulta numa degradação do desempenho do modelo. Grande parte dos modelos assumem uma relação estática entre o *input* e o *output*. O problema é que a distribuição do conjunto de dados pode variar. Então, *Meta-Learning* tenta ultrapassar este problema com a detecção da mudança e a respectiva adaptação à mudança. Quando é detectada uma mudança, a configuração actual pode ser adaptada prevendo configurações mais satisfatórias.

Optimização dos Hiperparâmetros

Ao estabelecermos a arquitectura da rede ou seleccionarmos um algoritmo de aprendizagem computacional, é necessário configurar os hiperparâmetros de modo a optimizar a performance obtida. Tal como nos outros aspectos de Aprendizagem Computacional, a configuração dos hiperparâmetros envolve o conhecimento e experiência do cientista de dados que experimenta diferentes combinações de valores para cada um dos hiperparâmetros. Torna-se uma tarefa árdua, tendo em conta que cada hiperparâmetro tem vários valores possíveis sendo por isso impraticável testar todos os valores.
A escolha dos hiperparâmetros tem de ser cuidadosamente pensada visto que influencia directamente a performance porque os hiperparâmetros interferem na forma como o modelo irá aprender os dados. A título de exemplo, a escolha da taxa de aprendizagem pode ter grandes consequências nos resultados. Este parâmetro controla a taxa a que modelo aprende e que actualiza os seus pesos. Com uma taxa de aprendizagem muito elevada, o modelo pode saltar um ponto óptimo e, por outro lado, com uma taxa de aprendizagem muito pequena, o modelo pode nunca convergir. Deste modo, é necessário encontrar o valor certo que permita que ao modelo aproximar da melhor forma uma função aos dados.

A investigação relativa a este conceito, difere principalmente na abordagem que é utilizada para percorrer o espaço de procura composto pelos hiperparâmetros. As abordagens mais aplicadas são: `Pesquisa em Grelha` (*Grid Search*), `Pesquisa Aleatória, Métodos Evolucionários, RL e Otimização Bayesiana`.

`Pesquisa em Grelha`, em Inglês, *Grid Search*, é um método simples que realiza uma procura exaustiva sobre o espaço de procura. Treina todas as combinações dos hiperparâmetros e verifica qual é que produz resultados melhores.
`Pesquisa Aleatória`, tal como o nome indica, vai treinando o modelo com configurações seleccionadas aleatoriamente com o intuito de seleccionar a melhor configuração. Nesta técnica, é possível limitar o tempo que a pesquisa é feita, ou definir um limite de sucesso. Estas duas abordagens não são eficientes pois são treinadas e avaliadas demasiadas configurações, o que leva à demora na configuração dos hiperparâmetros e à utilização excessiva de recursos computacionais. De notar que, a complexidade e os recursos utilizados aumentam exponencialmente com o número de hiperparâmetros.

`Métodos evolucionários` são muito utilizados em problemas de optimização e baseiam-se no processo de selecção natural. A ideia principal é ter uma população de indivíduos que representem diferentes configurações de hiperparâmetros. A cada iteração, uma nova população é gerada, por meio de processos de cruzamento e mutação, e a aptidão (performance) dos indivíduos é avaliada. Com o cruzamento, que é principalmente aplicado para fazer uma procura global, e a mutação, que é principalmente aplicado para fazer uma procura local, a população de indivíduos deve evoluir para cada vez melhor performance.

As técnicas propostas diferem na forma como são gerados e seleccionados os indivíduos.

`RL` também é muitas vezes aplicado em problemas de optimização. De uma forma geral, é utilizado um controlador que configura os hiperparâmetros e treina o modelo com essa configuração. No fim do treino, o controlador recebe uma resposta (*reward*) de modo a saber se a configuração criada é promissora ou não. Ao longo do tempo, o controlador ficará mais perspicaz e conseguirá perceber quais combinações de hiperparâmetros resultam numa melhor performance, com base nos *rewards* recebidos.

`Optimização Bayesiana` recorre a uma função de aquisição e um modelo probabilístico da função objectivo que se mantém a par dos resultados passados. Estes resultados são utilizados para mapear os diversos hiperparâmetros a uma probabilidade de um certo resultado na função objectivo. É um processo iterativo que a cada iteração o modelo probabilístico é ajustado com todas as observações da função objectivo até ao momento. De seguida, uma função de aquisição, determina importância dos diferentes pontos candidatos. Com base nesta importância, decide o caminho a seguir. A modelação da função objectivo é frequentemente feita com processos gaussianos ou com *Random Forests*.

Nesta subsecção de CASH vimos que este conceito engloba todos os algoritmos que pretendem seleccionar um modelo eficaz e configurar os respectivos hiperparâmetros de forma automatizada. No caso da escolha do algoritmo, esta pode ser feita de forma directa, em que um espaço de procura é composto por diversos algoritmos e o objectivo é encontrar o melhor algoritmo ou pode ser feita com recurso a NAS e *Meta-Learning*. Estes dois últimos são conceitos bastante promissores que presentemente já apresentam resultados bastante positivos. Ao mesmo tempo que se escolhe o algoritmo, temos de configurar os seus hiperparâmetros que influenciam directamente a performance obtida. Isto é um problema de optimização, onde queremos encontrar os hiperparâmetros que melhor se adequam ao problema. Neste contexto, também existem diversas técnicas que diferem na maneira como exploraram o espaço de procura composto pelos diferentes hiperparâmetros.

2.1.2 Estrutura Geral das Abordagens de AutoML

O problema que AutoML pretende resolver é a escolha de uma configuração eficaz, seja ela um conjunto de algoritmos de engenharia de atributos ou uma arquitectura de uma rede neuronal. De modo geral, para resolver este problema, define-se um espaço de procura composto por configurações possíveis com base nos dados e no tipo de problema. As diferentes configurações são seleccionadas ou construídas pelo Optimizador. A configuração construída pelo Optimizador é avaliada pelo Avaliador, e esta avaliação é dada ao Optimizador para que este perceba a relação entre as configurações e o respectivo desempenho.

A Figura 2.4 descreve a estrutura básica das abordagens existentes para resolver o problema de AutoML. Evidentemente, a estrutura pode ser um pouco diferente dependendo do problema a resolver, seja por exemplo limpeza inicial de dados ou selecção do modelo.

As diferentes abordagens diferem na definição do espaço de procura, no Optimizador e no Avaliador. Ou seja, diferem na forma como o espaço de procura é percorrido e na forma como as possíveis configurações são avaliadas.

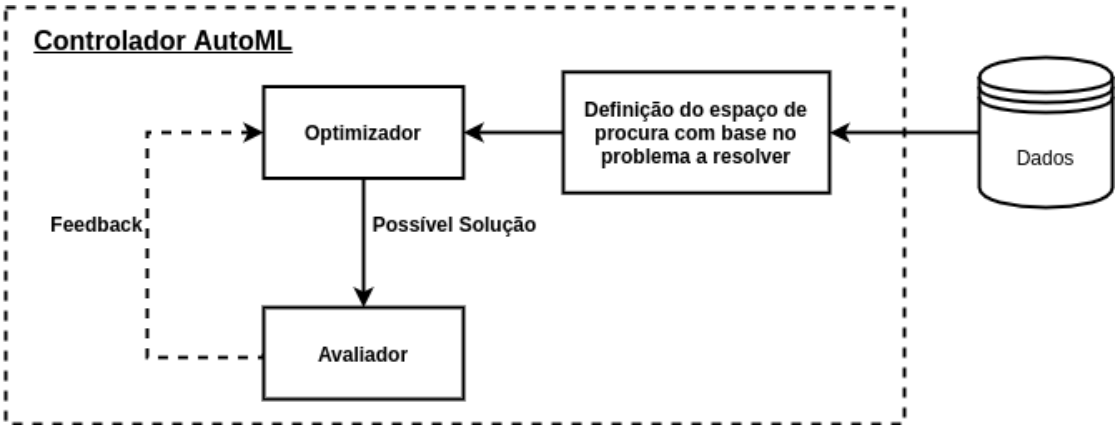


Figura 2.4: Estrutura Geral das Abordagens de AutoML

2.1.3 Ferramentas de AutoML

Com base na necessidade de oferecer um processo de aprendizagem computacional acessível e intuitivo, foram propostas várias plataformas e ferramentas de AutoML. Estas ferramentas permitem que empresas consigam tirar valor dos dados que possuem sem terem de lidar com a complexidade inerente a projectos de Aprendizagem Computacional. As abordagens seguidas pelas diversas ferramentas seguem muitos dos algoritmos apresentados ao longo do capítulo.

As ferramentas propõem automatizar o processo inteiro ou apenas automatizar algumas fases. Embora que o estudo apresentado na Figura 2.2 diga que a componente onde é gasto mais tempo é no tratamento dos dados, grande parte das ferramentas foca-se apenas no problema de CASH e deixa o tratamento dos dados para o utilizador. É compreensível visto que o tratamento dos dados depende do contexto do problema e uma ferramenta automática poderá inserir erros nos dados por não perceber o contexto.

Nesta subsecção serão descritas as ferramentas mais populares, dando uma ideia geral do que tentam automatizar. No fim da subsecção é também apresentada uma tabela (Tabela 2.1) que, de forma sumariada, expõe as principais funcionalidades das ferramentas descritas de seguida.

`Featuretools`[5]

Featuretools é uma biblioteca que pretende automatizar o processo de engenharia de atributos. Na sua base está o algoritmo *Deep Feature Synthesis* [12] que foi apresentado no presente capítulo. Esta ferramenta é intuitiva e de fácil implementação sendo que as únicas tarefas que o utilizador precisa de realizar são a definição das entidades (tabelas) do conjunto de dados e a forma como essas entidades se relacionam (relações pai-filho). Com base nas entidades e nas relações, o algoritmo junta diferentes de operações básicas a que chamam '*Feature Primitive*'. Estas operações podem ser transformações ou agregações. As transformações são aplicadas a uma ou mais colunas da mesma entidade (tabela). E as agregações são operações que são aplicadas em várias tabelas com relações pai-filho.
É uma ferramenta em código aberto que permite a fácil integração no projecto a desenvolver por meio de funções pertencentes a uma biblioteca.

[5] `https://github.com/FeatureLabs/featuretools`

`Auto-sklearn`[6]

Auto-sklearn é uma biblioteca construída com recurso a uma biblioteca de Aprendizagem Computacional já existente - *scikit-learn*. Esta biblioteca foi desenvolvida por investigadores da Universidade de Friburgo e está descrita no artigo produzido por Feurer et al. [17]. Oferece funcionalidades para tratamento de dados e escolha do melhor algoritmo para problemas de classificação e regressão. Utiliza pesquisa bayesiana para optimizar a escolha dos pré-processadores e dos algoritmos utilizados no processo. Várias combinações são treinadas e no fim as melhores podem ser agrupadas num único modelo.
Técnicas de *Meta-Learning* foram utilizadas para configurar os valores iniciais dos hiperparâmetros. A biblioteca foi testada com 140 conjuntos de dados, permitindo assim ter uma base sólida para iniciar o processo de procura em vez de começar do zero.
É composta por 14 algoritmos de engenharia de atributos, 15 algoritmos de classificação e 12 algoritmos de regressão. Todos estes algoritmos pertencem à biblioteca *scikit-learn*.
Esta ferramenta é bastante customizável. É possível especificar o tempo máximo para o algoritmo correr e especificar que algoritmos pertencem ao espaço de procura. Fornece ainda capacidade de computação paralela.

`TPOT`[7]

TPOT automatiza grande parte dos passos do processo de aprendizagem computacional. Tem funcionalidades que permitem a automatização da selecção de atributos, transformação de atributos, selecção do modelo e optimização dos parâmetros. Para optimizar e encontrar os melhores algoritmos utiliza programação genética.
Considera diversos algoritmos de pré-processamento, vários algoritmos de aprendizagem computacional e agrupa todos estes algoritmos de forma a criar um *pipeline* que obtenha bons resultados. Sendo que é necessário escolher um algoritmo de entre vários algoritmos, o processo pode ser bastante demorado. Os investigadores que desenvolveram a plataforma explicam a razão do algoritmo demorar muito tempo. Por exemplo, se tivermos 100 gerações e 100 indivíduos na população, o algoritmo vai avaliar 10000 configurações diferentes. É um número bastante elevado de configurações para treinar e testar.
A grande vantagem é que no fim, o algoritmo retorna o melhor conjunto de modelos treinados e o código base para gerar os modelos. Com o código é possível que cientistas de dados façam as suas alterações de forma a obter uma performance ainda melhor.

`AutoKeras`[8]

AutoKeras foi das primeiras ferramentas implementadas com o objectivo de fornecer capacidades de aprendizagem profunda de forma acessível e intuitiva. Foca-se na automatização da procura de uma rede neuronal profunda para um problema em concreto. Procura ajudar nas tarefas em que é aplicada aprendizagem profunda, recorrendo algoritmos de NAS com optimização bayesiana.
Esta ferramenta foi implementada com base na investigação de Jin et al. [18] que desenvolveu um algoritmo de NAS baseado em optimização bayesiana. Neste artigo foi introduzida a ideia de morfismo da rede que pretende diminuir o tempo de treino. É uma técnica para transformar a arquitectura de uma rede profunda, mas manter sua funcionalidade.
A grande desvantagem é o facto de o algoritmo ter de correr a fase de treino a partir

[6] `https://github.com/automl/auto-sklearn`
[7] `https://github.com/EpistasisLab/tpot`
[8] `https://github.com/keras-team/autokeras`

do zero para todas as arquitecturas geradas levando assim a uma demora na selecção da melhor rede. De forma a solucionar este problema, o sistema pode correr paralelamente na Unidade central de processamento (CPU) e no Unidade de processamento gráfico (GPU) da máquina local, adaptando-se aos diferentes limites de memória das máquinas.

`H2O AutoML`[9]

H20 é uma ferramenta de aprendizagem computacional parecida ao sklearn. Tal como o sklearn, possui diversos algoritmos de aprendizagem e possui ainda um módulo dedicado para automatizar o pré-processamento de dados, encontrar o modelo mais eficiente e optimizar os diversos parâmetros.
Para encontrar os algoritmos mais eficientes, quer para o pré-processamento quer para o algoritmo de aprendizagem, recorre a procura exaustiva. Esta procura pode ser feita com procura aleatória ou com pesquisa em grelha (*grid search*).
É uma ferramenta *open source*, que pode correr localmente numa máquina ou de forma distribuída. A maior desvantagem é o consumo de recursos devido ao tipo de procura utilizado.

`MLBox`[10]

MLBox fornece funcionalidades para limpeza e formatação dos dados, engenharia de atributos, selecção do modelo, optimização dos hiperparâmetros e interpretação do modelo. É uma ferramenta distribuída permitindo assim distribuir a tarefa em várias máquinas. Esta funcionalidade é importante pois a resolução de problemas numa única máquina limita os recursos computacionais disponíveis.
Contêm três sub-pacotes:

- `Pré-Processamento:` faz uma limpeza inicial dos dados e fornece capacidades de seleção dos atributos que contêm mais informação;
- `Otimização:` utiliza a biblioteca; *hyperopt*[11] para optimizar a escolha do modelo e dos hiperparâmetros;
- `Previsão:` fornece capacidades para testar o modelo gerado com o conjunto de teste;

`Ludwig`[12]

Esta biblioteca permite a geração, o treino, o teste e a visualização de modelos a partir da linha de comandos ou com recurso a funções em *Python*. As arquitecturas geradas baseiam-se numa biblioteca de aprendizagem profunda bastante conhecida - *TensorFlow*.
A sua principal vantagem é que fornece a capacidade de treinar redes neuronais profundas sem ser preciso a implementação de código. Se optarmos pela utilização na linha de comandos são apenas precisos dois ficheiros: o ficheiro que contém o conjunto de dados e um ficheiro no formato YAML que especifica o nome dos atributos e a classe a prever. De seguida utiliza-se um comando pré-definido para treinar e encontrar a melhor arquitectura. Possibilita que o treino seja feito localmente ou de maneira distribuída.

[9] `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`
[10] `https://github.com/AxeldeRomblay/MLBox`
[11] `https://github.com/hyperopt/hyperopt`
[12] `https://github.com/uber/ludwig`

`Google AutoML`[13]

Esta tecnologia implementada pela Google permite a integração fácil e rápida de um modelo de Aprendizagem Profunda. Baseia-se em algoritmos de NAS que utilizam RL de forma a encontrar a melhor arquitectura. Possuem um conjunto de arquitecturas pré-treinadas que com recurso a *Meta-Learning* permitem não começar a procura a partir do zero. É composta por uma interface gráfica simples que permite treinar, avaliar e integrar o modelo no mundo real.
Google AutoML oferece vários serviços para vários tipos de problemas:

- `AutoML Vision:` para classificar imagens de acordo com classes predefinidas pelo utilizador;
- `AutoML Video Intelligence:` para classificar sequências ou rastrear objectos em vídeos;
- `AutoML Natural Language:` para classificar documentos ou extrair um conjunto de entidades de textos;
- `AutoML Translation:` para funcionalidades de tradução de textos;
- `AutoML Tables:` para problemas de classificação ou regressão em conjuntos de dados estruturados;

A desvantagem principal é o preço elevado associado a estes serviços.

Ferramenta	**Funcionalidades**	**Interface**	**Código Aberto**
Featuretools	✓Tratamento dos dados	Não	Sim
AutoSklearn	✓Tratamento dos dados ✓Selecção do Modelo ✓Optimização dos hiperparâmetros	Não	Sim
MLBox	✓Tratamento dos dados ✓Selecção do Modelo ✓Optimização dos hiperparâmetros	Não	Sim
TPOT	✓Tratamento dos dados ✓Selecção do Modelo ✓Optimização dos hiperparâmetros	Não	Sim
AutoKeras	✓Neural Architecture Search	Sim	Sim
Google AutoML	✓Tratamento dos dados ✓Neural Architecture Search	Sim	Não
H20	✓Tratamento dos dados ✓Selecção do Modelo ✓Optimização dos hiperparâmetros	Não	Não
Ludwig	✓Tratamento dos dados ✓Neural Architecture Search	Não	Sim

Tabela 2.1: Resumo das Ferramentas de AutoML

[13] `https://cloud.google.com/automl`

2.1.4 Desafios futuros

AutoML tem sido uma disciplina muito estudada nos últimos anos, com cada vez mais investigação de modo a automatizar o processo complexo de Aprendizagem Computacional e fazer face à necessidade de ferramentas que tornam Aprendizagem Computacional mais acessível. Contudo, ainda existem vários desafios para que o processo esteja totalmente automatizado.
O grande objectivo é automatizar todos as etapas, desde tratamento de dados à construção do modelo, e integrá-las num sistema completo. É um objectivo que dará a possibilidade a organizações de tirarem mais valor dos dados para que possam optimizar os seus processos ou oferecer um produto melhor ao cliente. Dará ainda, a oportunidade a pessoas com pouca ou nenhuma experiência tirar benefícios desta disciplina.
Apesar dos progressos recentes, ainda existem diversos desafios para que o domínio de Aprendizagem Computacional esteja ao alcance de todos.

Um dos grandes desafios é a interpretabilidade dos algoritmos de aprendizagem gerados. Por norma, as diversas técnicas desenvolvidas para a geração de modelos, superam os humanos na procura de arquitecturas mais eficazes. No entanto, os modelos gerados são de difícil interpretação e não existem fundamentos ou provas científicas que expliquem o porquê da arquitectura gerada ter mais eficácia do que a desenhada pelo o humano. Se nos focarmos apenas na performance obtida, verificamos que as arquitecturas geradas sobressaem, contudo não existe uma prova matemática para explicar o sucedido.

Reprodutibilidade é um dos desafios já presentes na disciplina de Aprendizagem Computacional e como tal, também presente no campo e AutoML. Grande parte da investigação realizada obtém grandes resultados durante as experiências. Todavia, a reprodução dos mesmos resultados num outro ambiente é improvável.

Outro desafio que vem do domínio de Aprendizagem Computacional, é o facto de nenhum algoritmo conseguir obter uma boa performance em todos os problemas. O teorema "*No Free Lunch*", introduzido por David H. Wolpert et al. [19], constata que não existe nenhum algoritmo que é mais adequado para todos os problemas. Este problema também ocorre na área de AutoML onde o facto dos problemas diferirem no seu domínio e no seu conjunto de dados, dificulta a implementação de um algoritmo geral, que Pedro Domingos define no seu livro [1] como o *algoritmo mestre*.
Outro problema é o tempo que demora até se obter a configuração recomendada. As diferentes técnicas estudas, pretendem encontrar a melhor configuração num determinado espaço de procura. Este espaço, é enorme e é composto por várias combinações possíveis. Por esse motivo, a procura pela melhor configuração é demorada e depende do método de procura utilizado. Neste caso, o utilizador tem de jogar com o tempo que tem disponível. Quanto maior for o tempo disponível, maior será o espaço explorado e maior será a probabilidade de se obter uma configuração melhor. Porém, não é algo assim tão linear. O tempo extra pode ser utilizado a explorar configurações pouco promissoras. Quanto maior for o tempo disponível, maior será o consumo de recursos computacionais e consequentemente o custo associado ao projecto.

Por fim, a capacidade de um sistema de AutoML aprender continuamente ao longo do tempo é também um desafio. Um sistema deve ser capaz de usar conhecimento precedente para resolver novas tarefas e estar constantemente a aprender com novos conjuntos de dados. A capacidade de um sistema se adaptar a novos dados é fulcral para que este esteja continuamente a acrescentar valor às organizações.

Esta página foi intencionalmente deixada em branco.

Esta página foi intencionalmente deixada em branco.

Capítulo 3

Descrição do Projecto

Este capítulo serve o propósito de descrever o módulo AutoML implementado, assim como o assistente virtual criado. São descritas e detalhadas as principais funcionalidades dos sistemas implementados, oferecendo assim uma visão geral para que possam ser introduzidos detalhes da implementação no capítulo seguinte (Capítulo 4).

A primeira secção, Secção 3.1, é dedicada à descrição do módulo AutoML e das suas funcionalidades. A Secção 3.2 descreve o assistente virtual implementado e explica como é que o assistente comunica com o módulo AutoML. Na secção 3.3 são apresentados os requisitos funcionais e os requisitos não funcionais. Por fim, na Secção 3.4, é apresentada a arquitectura do módulo AutoML.

3.1 Descrição do módulo AutoML

O objectivo principal do estágio passou pelo desenvolvimento de módulo capaz de realizar várias tarefas associadas a um projecto de Aprendizagem Computacional de forma automatizada. Neste sentido, foi definido implementar de forma automatizada a limpeza inicial de dados, engenharia de atributos e selecção do melhor modelo.

A decisão relativa a que tarefas o módulo iria automatizar, foi cuidadosamente pensada e teve em conta principalmente 3 factores:

- Visão da Critical Software e dos colegas de equipa para o sistema.O módulo desenvolvido é principalmente para uso interno e como tal, foram avaliadas as necessidades existentes para o desenvolvimento de projectos de Aprendizagem Computacional de forma mais rápida e eficiente. Com o módulo desenvolvido, os engenheiros da Critical Software terão a oportunidade de desenvolver produtos de forma mais rápida e simples, beneficiando das capacidades do módulo para automatizar tarefas que se tornam repetitivas ao longo de diversos projectos.

- As tarefas escolhidas incidem sobre grande parte do processo de Aprendizagem Computacional e são de extrema importância para atingir o melhor desempenho.

- A inclusão da fase de limpeza de dados e da fase de engenharia de atributos deveu-se ao facto de grande parte das ferramentas de AutoML disponíveis não automatizarem

estas fases. Ainda para mais, tendo em conta que, tal como vimos da Secção 2, são das tarefas que consomem mais tempo num projecto de Aprendizagem Computacional.

O módulo AutoML foi dividido em três submódulos, correspondentes a cada uma das etapas: limpeza inicial de dados, engenharia de atributos e selecção do modelo. Esta divisão lógica permite a execução das três etapas sequencialmente assim como a execução de apenas uma etapa. De seguida, iremos abordar o que devem fazer cada uma das etapas e no Capitulo 4 iremos entrar em detalhe nas abordagens e operações referentes a cada submódulo, explicando como é que cada um executa as suas funções.

O submódulo de limpeza de dados automatiza a escolha de operações de limpeza de dados. Recebe o conjunto de dados e, depois de avaliar várias combinações de operações, decide qual é a combinação que consegue tratar os erros e as inconsistências da melhor forma. Após uma análise e pesquisa de quais os erros e inconsistências mais comuns, foi definido que este submódulo iria tratar dos seguintes erros: valores em falta, *outliers*, valores duplicados, valores categóricos inconsistentes, atributos que contêm datas e transformação de atributos categóricos. Este submódulo permite não só a escolha automatizada das operações de limpeza de dados mas também a execução de apenas uma operação em específico.

O submódulo de engenharia de atributos é composto por diversas operações que manipulam e transformam os dados, e automatiza a escolha das operações que melhoram o desempenho preditivo. Este submódulo tem três tipos de operações: transformação de atributos, geração de atributos e selecção de atributos. Estes três tipos divergem no que toca às operações e aos atributos que são transformados/criados/seleccionados. Para cada tipo existem diversas operações, cuja escolha foi sustentada por uma análise das operações mais utilizadas, e o submódulo deve avaliar quais são as operações que resultam numa melhor performance. Da mesma forma que o submódulo anterior, este submódulo permite não só a escolha automatizada das operações de engenharia de atributos mas também a execução de apenas uma operação em específico.

O submódulo de selecção do modelo automatiza a escolha do melhor algoritmo de aprendizagem e optimiza os seus hiperparâmetros. Neste caso, o utilizador pode definir que algoritmos de aprendizagem entram para o espaço de procura. Se o utilizador já tiver um algoritmo de aprendizagem em mente e quiser apenas descobrir quais são os melhores hiperparâmetros, pode apenas executar um algoritmo de aprendizagem específico, e o submódulo encontra a melhor configuração de hiperparâmetros para esse algoritmo de aprendizagem.

Sumariando as funcionalidades do módulo, este permite:

- A execução dos três submódulos sequencialmente - limpeza de dados, seguida de engenharia de atributos e terminando com selecção do modelo
- A execução de apenas um dos submódulos isoladamente. Por exemplo, é possível apenas executar a limpeza de dados automatizada.
- A execução de uma operação específica. Por exemplo, é possível apenas executar a operação de remoção de valores em falta ou apenas um algoritmo de aprendizagem.

Com estas funcionalidades, o módulo AutoML pode ser usado para tirar benefícios de Aprendizagem Computacional Automatizada, sendo que o módulo escolhe quais são as

melhores operações a aplicar, e pode também pode ser utilizado como uma ferramenta de Aprendizagem Computacional porque permite que o utilizador apenas execute métodos específicos.

Por fim, a arquitectura desenhada e implementada teve em conta a versatilidade na inclusão do módulo noutros sistemas. Este módulo deve, no futuro, poder ser integrado noutros sistemas sem que sejam necessárias profundas alterações. Como tal, o módulo AutoML desenvolvido é um *package* de *python*, possibilitando assim a sua utilização noutros projectos e sistemas de forma bastante simples. Foi ainda adicionado um serviço REST que permite a execução das funções do módulo com recurso a pedidos HTTP.

3.2 Descrição do Assistente Virtual

Outro objectivo do projecto era a criação de um Assistente Virtual capaz de realizar tarefas de Aprendizagem Computacional de forma automatizada. O módulo criado iria ser integrado no sistema Lexia que está a ser desenvolvido pela Critical Software. Este sistema permite, ou irá permitir, interacção homem-máquina, baseado na comunicação em linguagem natural. Neste sentido, a integração do módulo AutoML no sistema Lexia iria permitir usufruir das capacidades automatizadas de Aprendizagem Computacional com recurso a linguagem natural.

Contudo, o sistema Lexia, que é externo a este estágio, não ficou desenvolvido a tempo, sendo assim impossível de integrar o módulo no sistema. Existiam então dois caminhos possíveis. O primeiro seria descartar a ideia do assistente virtual e apenas implementar o módulo de AutoML que no fundo é o objectivo principal deste estágio. O segundo, seria implementar um assistente virtual com recurso a ferramentas já existentes que permitem o desenvolvimento rápido deste tipo de sistemas. Foi escolhida a segunda opção.

Desta forma, foi desenvolvido um assistente virtual que utiliza o módulo AutoML e permite que as diferentes tarefas de Aprendizagem Computacional sejam executadas como se o utilizador estivesse a falar por mensagens com um cientista de dados.

Para construir o assistente virtual foi utilizada a biblioteca Rasa[1]. Esta biblioteca foi escolhida porque os colegas da equipa já tinham tido contacto com a mesma e permite o desenvolvimento rápido de assistentes de contexto. Assistentes de contexto, são assistentes que têm em conta a sequência da conversa com o utilizador, guardando o contexto ao longo da conversa. É precisamente isto que precisamos porque se o utilizador diz que quer implementar uma operação de geração de atributos, o assistente deve perguntar que operação é suposto implementar e lembrar-se que a operação é relativa a geração de atributos. No Capítulo 4, será explicado com mais profundidade a implementação do assistente virtual.

O assistente virtual implementado tem a capacidade de: i) executar o passo de limpeza de dados; ii) executar uma operação especifica de limpeza de dados; iii) executar o passo de engenharia de atributos; iv) executar uma operação específica de engenharia de atributos; v) executar o passo de selecção do modelo; vi) executar um modelo em específico; vii) executar os três passos em conjunto. A execução destas funcionalidades tem por base a mensagem do utilizador. O Assistente Virtual deve identificar a intenção do utilizador e

[1] https://rasa.com/

chamar as funções do módulo AutoML de forma a satisfazer o pedido do utilizador.

Para o assistente virtual conseguir servir-se dos métodos do módulo AutoML foi necessário adicionar um serviço REST ao módulo. Este serviço permite que o assistente virtual consiga executar as funções do módulo AutoML por meio de pedidos HTTP.

Na Figura 3.1 é representada a interacção entre o utilizador, o assistente virtual e o módulo AutoML. O utilizador envia uma mensagem ao assistente virtual e este extrai a intenção da mensagem. Com base na intenção, o assistente virtual executa a função desejada do módulo AutoML por meio de um pedido HTTP. Quando a função é executada, o assistente virtual recebe os resultados em formato JSON(JavaScript Object Notation) e retorna os resultados ao utilizador.

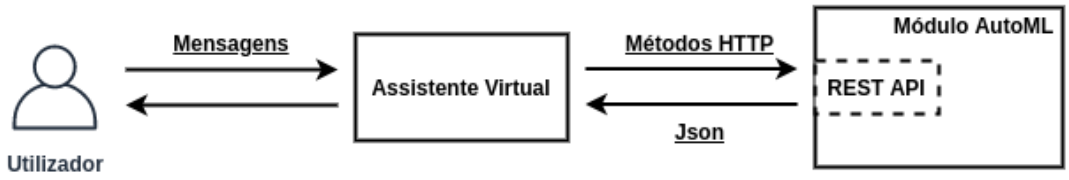


Figura 3.1: Interacção com o Assistente Virtual

3.3 Análise dos Requisitos

Nesta secção são apresentados os requisitos funcionais e os requisitos não-funcionais. Estes dois tipos de requisitos são essenciais em qualquer projecto de desenvolvimento de *software* pois descrevem as funcionalidades de um sistema e descrevem como é que este funciona.

3.3.1 Requisitos Funcionais

Nesta subsecção são apresentados os requisitos funcionais. Estes, são um aspecto importante no desenvolvimento de um produto pois fornecem uma descrição detalhada do que o sistema deve fazer. Permitem que sejam definidas as funcionalidades e o comportamento do sistema. Os requisitos funcionais apresentados foram realizados com base nas funcionalidades abordadas na secções anteriores.
Os requisitos são constituídos por um identificador, pelo nome, por dependências, pelo cenário de sucesso e pela respectiva prioridade.

ID	**RF01**
Nome	Execução automatizada da tarefa de limpeza inicial
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Procurar a melhor combinação de algoritmos de limpeza de dados; 3. Devolver conjunto de dados limpo;
Prioridade	Must Have

Tabela 3.1: Requisito Funcional 01

ID	**RF02**
Nome	Execução de uma operação específica de limpeza de dados
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Executar a operação de limpeza de dados; 3. Devolver conjunto de dados transformado;
Prioridade	Must Have

Tabela 3.2: Requisito Funcional 02

ID	**RF03**
Nome	Execução automatizada da tarefa de engenharia de atributos
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Procurar a melhor combinação de algoritmos de engenharia de atributos 3. Devolver conjunto de dados transformado;
Prioridade	Must Have

Tabela 3.3: Requisito Funcional 03

ID	**RF04**
Nome	Execução de uma operação específica de engenharia de atributos
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Executar a operação de engenharia de atributos; 3. Devolver conjunto de dados transformado;
Prioridade	Must Have

Tabela 3.4: Requisito Funcional 04

ID	**RF05**
Nome	Execução automatizada da tarefa de selecção do modelo
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Seleccionar um algoritmo eficaz; 3. Devolver modelo seleccionado e os respectivos resultados;
Prioridade	Must Have

Tabela 3.5: Requisito Funcional 05

ID	**RF06**
Nome	Execução de um algoritmo de aprendizagem específico
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Procurar os hiperparâmetros do algoritmo que obtêm melhores resultados; 3. Devolver modelo e os respectivos hiperparâmetros;
Prioridade	Must Have

Tabela 3.6: Requisito Funcional 06

ID	**RF07**
Nome	Execução das tarefas em conjunto
Dependências	Conjunto de dados guardado em memória
Cenário de Sucesso	1. Carregar conjunto de dados; 2. Verificar quais os submódulos é suposto executar; 3. Executar os submódulos sequencialmente; 4. Devolver os resultados
Prioridade	Must Have

Tabela 3.7: Requisito Funcional 07

ID	**RF08**
Nome	Assistente Virtual executa operações de Aprendizagem Computacional Automatizada
Dependências	Conjunto de dados guardado em memória Connecção com o módulo AutoML
Cenário de Sucesso	1. O utilizador envia uma mensagem com o seu pedido; 2. O assistente virtual analisa a mensagem e extrai a intenção; 3. O assistente virtual executa uma função do módulo AutoML com base na intenção; 4. O assistente virtual retorna os resultados;
Prioridade	Must Have

Tabela 3.8: Requisito Funcional 08

3.3.2 Requisitos Não-Funcionais

Nesta subsecção são apresentados os requisitos não-funcionais. Estes definem como é que o sistema deve funcionar. São um conjunto de características que avaliam diversos atributos de qualidade do sistema, sendo por isso essenciais para averiguar se o sistema está a realizar o que deve fazer de forma eficaz sem prejudicar a usabilidade do mesmo.
Os requisitos não-funcionais apresentados são constituídos por um identificador, pelo nome e por uma breve descrição.

ID	**RNF01**
Nome	Integrabilidade
Descrição	O módulo implementado deve ser genérico para que seja simples integrar o mesmo noutros projectos

Tabela 3.9: Requisito Não-Funcional 01

ID	**RNF02**
Nome	Performance
Descrição	O módulo implementado deve ser capaz de processar grande quantidades de dados sem que seja afectada a performance do mesmo

Tabela 3.10: Requisito Não-Funcional 02

ID	**RNF03**
Nome	Confiabilidade
Descrição	O sistema deve ser confiável ao ponto de não experienciar falhas críticas 95% das vezes que é utilizado. Deve também, ser capaz de recuperar das falhas sofridas

Tabela 3.11: Requisito Não-Funcional 03

3.4 Arquitectura do Módulo

O módulo implementado encontra-se dividido em cinco grandes classes: *AutoML, Dataset, DataCleaningPipeline, FeatureEngineeringPipeline e ModelSelectionStep*. Esta partição pode ser observada no diagrama exposto na Figura 3.2 que representa o diagrama de classes do *package* de *python* referente ao módulo.

A classe *AutoML* é a classe principal com o qual o utilizador pode interagir. Nesta classe, estão definidas todas as funcionalidades do módulo que são depois executadas com auxílio das classes específicas de cada tarefa: *DataCleaningPipeline, FeatureEngineeringPipeline e ModelSelectionStep*. Esta divisão foi feita pois as tarefas são bastante diferentes e, com esta disposição é possível executar cada tarefa isoladamente e independentemente. Na classe *AutoML* é ainda definido o conjunto de dados, com recurso à classe *Dataset*. Esta classe fornece uma forma de guardar o conjunto de dados que está a ser trabalhado e manipulado.

Na classe *DataCleaningPipeline* estão definidas a função que procura qual é a melhor configuração de algoritmos assim como a função que avalia as diferentes configurações. Esta classe é utilizada quando queremos executar o pipeline inteiro, isto é procurar a melhor configuração de algoritmos, ou quando queremos apenas executar uma operação de limpeza de dados em específico.

A classe *FeatureEngineeringPipeline* fornece todos os métodos necessários para a execução de tarefas relacionadas com engenharia de atributos. Nesta classe, estão definidas funções que permitem procurar a melhor configuração de algoritmos, executar uma operação de engenharia de atributos específica ou ainda treinar os agentes de RL implementados. Esta classe está em constante comunicação com a classe *DeepQAgent* e a classe *DeepQAgentByStep*. Estas, fornecem os métodos relativos à execução e treino dos agentes. Este último é importante pois significa que um utilizador pode treinar os agentes de maneira a fazer face a um caso especial.

Por fim, a classe *ModelSelectionStep* serve o propósito da tarefa de selecção do modelo. Nesta classe, são chamados os métodos da ferramenta *TPOT* de forma a encontrar o melhor algoritmo de aprendizagem.

Esta disposição permite ao utilizador configurar quase todos os aspectos da execução. Isto é, o utilizador consegue ajustar que algoritmos entram no espaço de procura, seja de limpeza de dados, de engenharia de atributos ou de selecção do modelo. Pode treinar os agentes de engenharia de atributos para fazer face às suas próprias necessidades. E, por fim, também é possível a adição de novas operações, sendo apenas necessário criar uma classe que implementa os métodos da super classe já implementada.

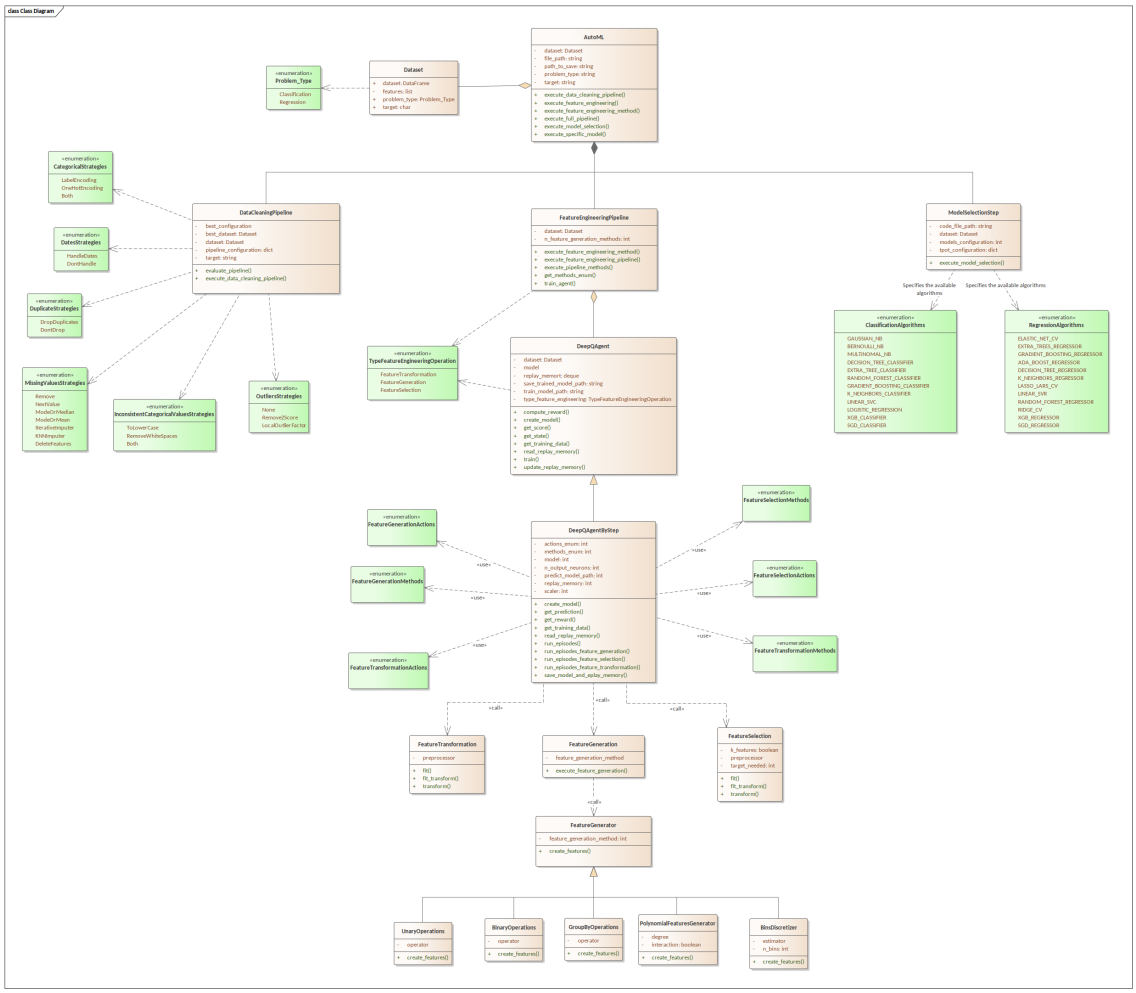


Figura 3.2: Diagrama de classes do módulo AutoML

Esta página foi intencionalmente deixada em branco.

Capítulo 4

Implementação

Neste capítulo, são apresentados os detalhes acerca da implementação dos diversos componentes desenvolvidos ao longo do estágio. Inicialmente, na Secção 4.1, são especificadas as ferramentas exteriores utilizadas ao longo do projecto. O resto do capítulo foca-se na apresentação das abordagens implementadas.
A Secção 4.2 descreve o algoritmo de escolha das operações para a limpeza de dados assim como as operações de limpeza implementadas. A Secção 4.3 é referente à implementação do submódulo de engenharia de atributos. Nesta secção, são introduzidas as operações de engenharia de atributos implementadas assim como a abordagem escolhida para automatizar a selecção de operações de engenharia de atributos. De forma a descrever a abordagem, primeiro introduzimos diversos conceitos e de seguida explicamos a utilização desses conceitos no contexto de engenharia de atributos. A Secção 4.4 descreve como foi utilizada a ferramenta *TPOT* para o submódulo de selecção do modelo. Por fim, a Secção 4.5 apresenta os detalhes referentes à implementação do assistente virtual. Neste secção, é introduzida a ferramenta *Rasa*, assim como os pormenores da implementação do serviço REST.

4.1 Ferramentas

Ao longo do projecto, foram utilizadas várias ferramentas externas ao projecto. Esta secção reúne as mais importantes, apresentando o que são e porque foram utilizadas.

Pandas e numpy
Pandas[1] e *Numpy*[2] são ferramentas que fornecem um conjunto de métodos úteis para trabalhar sobre estruturas de dados. São utilizados ao longo do módulo implementado para manipular dados.

Sklearn
Sklearn (ou scikit-learn)[3] é um módulo de *python* que fornece um conjunto de métodos relacionados com a área de Aprendizagem Computacional. É utilizado sempre que é necessário uma implementação de um algoritmo de Aprendizagem Computacional.

[1] `https://pandas.pydata.org/`
[2] `https://pypi.org/project/numpy/`
[3] `https://pypi.org/project/scikit-learn/`

Flask
Flask[4] é uma ferramenta que permite a escuta de pedidos HTTP. Foi utilizada para que o assistente virtual conseguisse comunicar com o módulo AutoML.

Fuzzywuzzy
Fuzzywuzzy[5] é uma ferramenta que permite calcular a diferença entre *strings*. Foi utilizada no submódulo de limpeza de dados.

Tensorflow
Tensorflow[6] é uma ferramenta que permite o desenvolvimento de redes neuronais. Foi utilizada no submódulo de engenharia de atributos.

PyMFE
PyMFE[7] é uma ferramenta que permite a extração das características (*meta-features*) dos dados. Foi utilizada no submódulo de engenharia de atributos.

4.2 Implementação do submódulo de limpeza de dados

O submódulo de limpeza de dados, é composto por diversas operações que tratam de vários tipos de erros e por uma função de avaliação que avalia as distintas combinações de algoritmos. Neste sentido, é necessário introduzir que operações entram no espaço de procura e como são avaliadas as combinações.

Em relação aos tipos de erros já observámos que o submódulo trata de: valores em falta, valores duplicados, *outliers*, valores categóricos inconsistentes, valores com datas e transformação de valores categóricos. Como existem vários métodos diferentes para tratar cada um deste erros, decidimos implementar as técnicas mais utilizadas. O tratamento de valores em falta tem 7 algoritmos possíveis. O tratamento de valores duplicados tem 2 algoritmos possíveis. O tratamento de *outliers* tem 3 algoritmos possíveis. O tratamento de valores categóricos inconsistentes tem 3 algoritmos possíveis. O tratamento de valores com datas tem 2 algoritmos possíveis. E, por fim, a transformação de valores categóricos tem 3 algoritmos possíveis. Desta forma, existem 756 combinações possíveis.
Estas 756 combinações de algoritmos, a serem executadas, produzem 756 conjuntos de dados diferentes. Dado o poder computacional das máquinas actuais, é possível testar 756 combinações de algoritmos num curto espaço de tempo. Neste sentido, a abordagem para a limpeza de dados é força bruta. Testamos todas as combinações possíveis e averiguamos que combinação traz melhores resultados com recurso à função de avaliação.

No que diz respeito à função de avaliação, foi necessário definir uma abordagem viável que conseguisse avaliar os dados resultantes da aplicação das várias combinações de operações. Desta forma, decidimos recorrer à função *selectKBest*[8] fornecida pela ferramenta *sklearn*.

[4] `https://pypi.org/project/Flask/`
[5] `https://pypi.org/project/fuzzywuzzy/`
[6] `https://pypi.org/project/tensorflow/`
[7] `https://pypi.org/project/pymfe/`
[8] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html`

Esta função é ideal porque recebe o conjunto de dados e avalia os diferentes atributos com base numa função de avaliação. Apesar de ser maioritariamente utilizada para seleccionar os melhores atributos na fase de engenharia de atributos, também pode ser utilizada para avaliar a importância dos atributos pois retorna a pontuação de cada atributo. Esta pontuação descreve o nível de informação que cada atributo tem em relação à variável a prever.
A função de avaliação funciona do seguinte modo: fornece os dados à função *selectKBest* que retorna a avaliação de cada atributo. De seguida, soma as avaliações de todos os atributos de forma a calcular a avaliação do conjunto de dados que está a ser avaliado. Quanto maior for a soma, maior será o nível de informação e consequentemente melhor será o conjunto de dados.

A abordagem à função de avaliação apresentada distancia-se das abordagens mais utilizadas e referidas no Capítulo 2. Grande parte das abordagens, compara os resultados da tarefa de previsão feita com os diversos conjuntos que passaram pelas diferentes combinações de operações, seleccionando o conjunto que obtém melhor performance. Estas abordagens gananciosas, pecam no facto de uma limpeza de dados não ter de necessariamente resultar numa performance melhor. Ao seleccionarmos combinações de operações apenas observando a performance, estamos a descartar combinações que, potencialmente, limpam melhor os dados mas que obtêm menor performance. A nossa abordagem não cai neste erro ganancioso pois ao avaliar o nível de informação que cada atributo limpo tem em relação à variável a prever, selecciona a combinação de operações que gera dados que contêm atributos mais importantes e com mais informação.

O fluxograma do algoritmo implementado para a escolha automatizada de operações, está representado na Figura 4.1, apresentada de seguida.

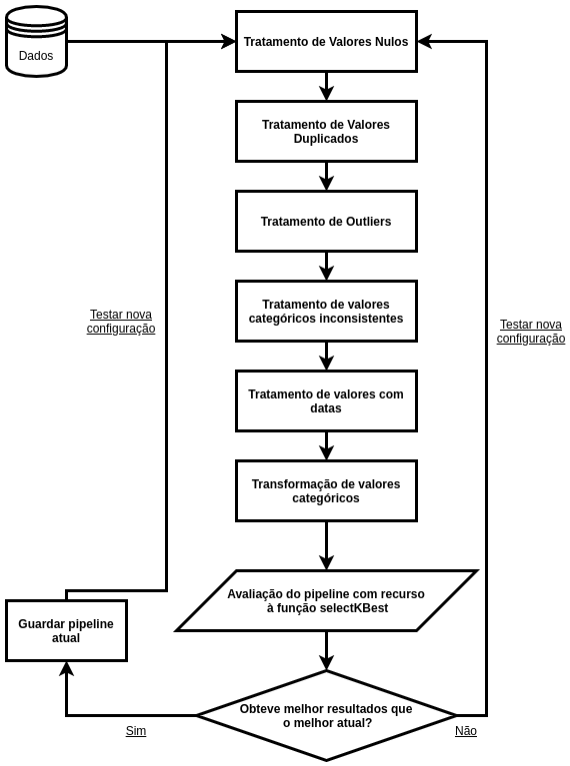


Figura 4.1: Fluxograma do Algoritmo de Limpeza de Dados Automatizada

Tal como podemos observar através da Figura 4.1, a cada iteração, os dados passam pelas operações do *pipeline* que está a ser avaliado. No fim da iteração, a combinação de algoritmos é avaliada, isto é, os dados transformados pela combinação de algoritmos são avaliados. Se esta nova combinação de algoritmos produzir dados de maior qualidade, estes novos dados e a nova combinação são guardados. Iteramos até que todas as combinações de algoritmos sejam executadas.
O algoritmo apresentado procura imitar o processo iterativo subjacente à limpeza de dados feita por um cientista de dados, onde este explora um grande espaço de operações possíveis e iterativamente vai testando que operações limpam melhor os dados.

Nesta fase, iremos dar uma breve descrição dos algoritmos utilizados para tratar de cada tipo de erro.

Tratamento de valores em falta
Aqui temos de encontrar que observações contêm valores em falta. Ao encontrar essas observações podemos seguir dois caminhos: remover essas observações ou substituir o valor em falta com um determinado valor. Não existe resposta certa, mais uma vez, depende do conjunto de dados. Se optarmos por remover as observações podemos incorrer no risco de perder informação relevante que os outros atributos tinham. Se optarmos por substituir o valor em falta, podemos comprometer os dados ao inserir valores errados nos mesmos. Não existindo concordância sobre o que fazer com valores em falta, decidimos implementar vários tipos de operações. Os algoritmos implementados para tratar de valores em falta são:

- `Apagar observações:` simplesmente apagam-se todas as observações que conterem pelo menos um valor em falta
- `Apagar atributos:` apagam-se atributos que contêm mais de 20% de valores em falta.
- `Valor seguinte:` substitui-se o valor em falta pelo valor seguinte. Este método é eficaz quando temos dados temporais
- `Moda ou média:` substitui-se o valor em falta pela média do atributo, no caso de atributos numéricos, ou pela moda do atributo, no caso de atributos categóricos
- `Moda ou mediana:` substitui-se o valor em falta pela mediana do atributo, no caso de atributos numéricos, ou pela moda do atributo, no caso de atributos categóricos
- `KNN Imputer:` recorremos à função KNN Imputer [9] que estima os valores para cada atributo com recurso ao algoritmo *K Nearest Neighbor*
- `Iterative Imputer:` recorremos à função Iterative Imputer [10] que estima os valores para cada atributo com recurso ao algoritmo *Bayesian Ridge*

Tratamento de valores duplicados
Para os valores duplicados também não existe consenso. Mais uma vez, podemos seguir dois caminhos: eliminar os valores duplicados ou não remover os valores duplicados. Alguns investigadores dizem que valores duplicados não adicionam valor aos dados e podem resultar que o algoritmo de previsão demore mais a treinar. Outros afirmam que, valores

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html`
[10] `https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html#sklearn.impute.IterativeImputer`

duplicados podem reflectir o contexto real e como tal devemos deixar os valores duplicados nos dados para que o algoritmo de previsão dê mais peso à classe que tem valores duplicados. Não havendo consenso, foi decidido implementar os dois caminhos e deixar o algoritmo verificar qual é o melhor para cada conjunto de dados. Os algoritmos implementados para tratar de valores duplicados são:

- `Remover valores duplicados:` removemos as observações que se repetem mais do que uma vez no conjunto de dados
- `Não remover valores duplicados:` não removemos os valores duplicados

Tratamento de *outliers*
Outliers podem enviesar a distribuição real dos dados resultando numa previsão errada. Neste caso, a comunidade tem uma opinião consistente. Por norma, é sempre positivo remover dados que se afastem muito dos dados ditos normais. O problema reside no facto de que tentar encontrar *outliers* de forma automatiza sem perceber o contexto dos dados pode levar à detecção de dados que na realidade não são outliers, pois a definição de *outlier* está fortemente dependente do contexto do problema. De forma a minimizar os erros na detecção dos outliers, foi decidido implementar dois algoritmos que detectam *outliers* de forma diferente. E ainda temos uma outra opção que é não detectar/remover *outliers*. O algoritmos implementados para tratar de *outliers* são:

- `Utilizar z-score para detectar e eliminar os dados detectados:` o teste z-score diz-nos o número de desvios padrão que uma observação está da média da amostra. Quanto mais afastado está o valor, maior será o número de desvios padrão. A nossa implementação, em modo *default*, remove todos os valores que estão a 3 (número configurável) desvios padrão acima da média da amostra.
- `Utilizar a função Local Outlier Factor para detectar e eliminar os dados detectados:` Local Outlier Factor [11] é uma função que calcula a densidade de um determinado valor com base nos seus vizinhos. Considera que um valor é *outlier* se o mesmo possuir uma densidade substancialmente menor que a dos seus vizinhos. Aqui, temos de definir o número de vizinhos, que no nosso caso foi definido como 20 (número configurável). Todas as observações que esta função marcar como *outliers* são eliminadas.
- `Não eliminar outliers:` não removemos os *outliers*, minimizando assim o risco de uma detecção errada

Tratamento de valores categóricos inconsistentes
Este passo pretende tratar valores categóricos que possam ter erros ou estar em formatos diferentes levando a que sejam representados como valores distintos mas que na realidade representam o mesmo valor. Por exemplo, um atributo que fornece informação acerca das cidades pode ter o valor '*New York City*' e o valor '*New York*'. Apesar destes valores representarem a mesma cidade para alguém conhecedor do contexto, para as máquinas representam cidades diferentes. Pequenos detalhes levam a que atributos categóricos iguais sejam interpretados como diferentes. Os erros mais comuns no que toca a valores categóricos são inconsistências com letras maiúsculas (*'New York'* terá uma representação diferente de *'new york'*) e espaços em branco no inicio da frase (' Algarve' terá uma representação diferente de 'Algarve'). Estes são tratados de forma bastante simples e directa. Existem

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html`

porém erros onde um caracter pode estar mal inserido (por exemplo 'Algarve' e 'Algarbe'). Estes erros já são mais difíceis de encontrar e por isso recorremos a uma ferramenta externa para realizar tal tarefa.
Os algoritmos implementados para tratar de valores categóricos inconsistentes são:

- `Transformação de letras maiúsculas e letras minúsculas:` todos os valores categóricos passam apenas a conter letras minúsculas. Deste modo resolvemos as incongruências do género 'Algarve' e 'algarve'

- `Remoção dos espaços em branco:` os espaços em branco no início dos valores categóricos são removidos. Deste modo, resolvemos as incongruências do género ' Algarve' e 'Algarve'

- `Correspondência difusa (Fuzzy Matching):` *fuzzy matching* é o processo de encontrar palavras que são parecidas com uma palavra alvo, avaliando o número de letras que as palavras têm em comum. Para tal, foram aplicados os métodos da ferramenta *fuzzywuzzy* que fornecem a capacidade para detectar palavras parecidas. Quando esta ferramenta detecta palavras parecidas, a palavra a ser testada é substituída pela palavra alvo. Esta técnica permite resolver os erros onde um caracter pode estar mal inserido, por exemplo, 'Algarve' e 'Algarbe'

Tratamento de valores com datas
Neste componente estamos interessados em encontrar atributos com datas. Para todos os atributos categóricos tentamos inferir se estes são datas com recurso ao módulo *datetime* do *python*. Se encontrarmos um atributo que é uma data, criamos diversos novos atributos com base nas datas encontradas.
Uma data, à primeira vista, poderá não dar muita informação sendo que é apenas um ponto específico no tempo. Todavia, se a data for separada em por exemplo dia da semana já fornece mais informação. Pegando no exemplo anterior, se tivermos um conjunto de dados em que o objectivo é prever o número de pessoas que assistiu a um determinado evento em diversos dias de um ano, e criarmos o atributo dia da semana, concluiremos que provavelmente na sexta e no sábado o número de pessoas vai ser mais elevado. Nesta etapa, convertemos uma data, que à partida não fornece muita informação, num atributo que irá dá melhores intuições ao modelo.
Cada vez que é encontrado um atributo que contém datas, são criados os seguintes atributos:

- `Dia da semana:` transforma-se a data em dia da semana, de segunda a domingo.

- `Mês:` cria-se um atributo com informação relativa ao mês. Este atributo é importante em situações sazonais. Por exemplo, as vendas de brinquedos no mês 11 (novembro) e 12 (dezembro) aumentam devido ao Natal.

- `Ano:` cria-se um atributo com informação relativa ao ano. É interessante quando temos dados temporais que se estendem ao longo de vários anos

- `Dia da semana ou fim de semana:` atributo binário que indica se é dia da semana ou fim de semana. Útil para as situações onde temos eventos que ocorrem com mais regularidade no fim de semana

- `Parte do dia:` atributo que dá informação se o evento ocorreu de manhã ou de tarde.

Estas operações poderão ser consideradas de geração de atributos pois criam-se novos atributos a partir de um existente. Contudo, tiveram de ser colocadas na fase de limpeza de dados porque os atributos com datas são transformados no passo seguinte que transforma os valores categóricos em valores numéricos. Com isto, iríamos perder a informação relativa às datas e na fase de engenharia de atributos já não seriamos capazes de perceber que atributos contêm datas.

Transformação de valores categóricos
Por fim, é necessário apenas de transformar os valores categóricos em valores numéricos porque grande parte dos algoritmos de aprendizagem computacional não conseguem lidar com valores categóricos. Mais uma vez, não existe consenso sobre os algoritmos a utilizar para codificar os valores categóricos, apesar de existirem dois que são frequentemente utilizados. Cada um tem as suas vantagens e é adequado para certo tipo de problemas. Não havendo concordância, decidimos implementar os dois.
Os algoritmos implementados para codificar os valores categóricos são:

- `Label Encoding:` codifica os valores únicos de um atributo com um valor entre 0 e número de valores únicos. Cria apenas um atributo com toda a informação, onde a cada valor único é fornecido um valor único numérico.
- `One-Hot Encoding:` para cada valor único de um atributo é criado um novo atributo binário. Este atributo binário indica se o valor único em questão está ou não presente na observação.

Resumindo, o submódulo implementado testa todas as combinações dos algoritmos apresentados, para além de algumas variações dos mesmos e, com recurso à função de avaliação verifica qual é que é a combinação que gera melhores dados.

4.3 Implementação do submódulo de engenharia de atributos

A implementação deste submódulo passou primeiro pela análise da forma como poderíamos seleccionar as melhores operações de engenharia de atributos. Divergindo da abordagem de limpeza de dados, preferimos optar por uma abordagem cujas as escolhas de operações não fossem independentes das características dos dados. Numa fase inicial do estágio, definiu-se que existiam três abordagens possíveis: utilizar *Meta-Learning*, utilizar algoritmos evolucionários ou utilizar Reinforcement Learning (RL). Depois de uma análise cuidada das vantagens de cada abordagem, a escolha passou pela implementação de técnicas de RL de modo a treinar agentes que tenham a habilidade de perceber que operações devem ser aplicadas. Na área de RL existem diversas técnicas e, para automatizar esta tarefa, decidimos recorrer à técnica de *Deep Q Learning* que será introduzida mais adiante.

Esta escolha deveu-se principalmente a duas razões:

- RL é uma área bastante popular que tem obtido bons resultados em diversos problemas. Neste sentido, o aluno tinha interesse em aprender mais sobre a mesma.
- RL mostra-se muito eficaz a replicar a forma como nós aprendemos por tentativa erro. Desta forma, ao aplicar RL no contexto de engenharia de atributos, conseguimos simular a maneira como um cientista de dados aprende ao longo da carreira onde,

por experimentação sucessiva, vai percebendo e aprendendo que operações funcionam melhor.

Nesta secção, iremos primeiro introduzir todas as operações de engenharia de atributos implementadas. De seguida, serão descritos os conceitos necessários para a compreensão do algoritmo implementado, nomeadamente RL, *Q Learning* e *Deep Q Learning*. Por fim, será apresentado o algoritmo implementado que escolhe as operações de engenharia de atributos que devem ser aplicadas aos dados.

No que diz respeito às operações implementadas, dividimos as mesmas em três tipos: operações de transformação de atributos, operações de geração de atributos e operações de selecção de atributos. Esta divisão pareceu lógica porque os três tipos manipulam os dados com objectivos diferentes. A transformação de atributos coloca todos os atributos na mesma escala. A geração de atributos cria novos atributos com base nos atributos originais. E, por fim, a selecção de atributos selecciona os melhores atributos. Esta divisão é feita na maioria dos projectos de Aprendizagem Computacional, onde cada tipo de operações é aplicado sequencialmente. Ou seja, primeiro aplicamos transformação de atributos, de seguida geramos atributos e por fim seleccionamos os melhores atributos.

Para cada um dos tipos existem diversas operações possíveis que têm as suas vantagens e desvantagens dependendo no tipo de dados. Após uma análise das operações possíveis, decidimos implementar as operações que costumam mais vezes ser aplicadas em projectos de Aprendizagem Computacional.

Em termos de operações de transformação de atributos decidimos recorrer às implementações da biblioteca *sklearn*. Temos quatro operações: *Standard Scaler*[12], *Normalizer*[13], *Min Max Scaler*[14] e *Robust Scaler*[15]

No caso de geração de atributos existem três tipos de operações: operações unárias, operações binárias e operações que são aplicas com base em todos os atributos.
As operações unárias são aplicadas ao nível do atributo isolado. Utilizam um único atributo para gerar um novo atributo. Estas operações são o logaritmo, a raiz quadrada e *KBinsDiscretizer*[16]. No caso da raiz quadrada e do logaritmo criamos atributos novos que são as raízes quadradas e os logaritmos dos atributos originais, respectivamente. E no caso da implementação do *sklearn* de *KBinsDiscretizer*, um atributo contínuo é divido, usando uma distribuição uniforme, em intervalos com tamanho $k \in [5, 10, 15]$.

As operações binárias são funções que recebem dois atributos e criam um novo atributo. Estas operações são a soma, a diferença, a multiplicação, a divisão e agrupamento. As primeiras quatro são simples, para cada par de atributos é aplicada a respectiva operação

[12] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[13] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html`

[14] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

[15] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

[16] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html`

aritmética. As operações de agrupamento (*group-by*) utilizam uma variável alvo e uma variável para agrupar. Agrupam os dados com base na variável para agrupar, e depois realizam um determinado cálculo à variável alvo a partir dos dados agrupados. Os cálculos são: mínimo, máximo, média ou contagem. Na Tabela 4.1 está um exemplo de como as operações de agrupamento funcionam.

Variável Alvo	**Variável Agrupar**	**Agrupar por Max**	**Agrupar por Média**
2	Classe 1	5	2
5	Classe 1	5	2
-1	Classe 1	5	2
6	Classe 2	20	13.6
15	Classe 2	20	13.6
20	Classe 2	20	13.6

Tabela 4.1: Exemplos de variáveis criadas pela operação de agrupamento

Por fim, foi também implementada uma operação que é aplicada com base em todos os atributos pertencentes aos dados. Esta operação é a *Polynomial Features*[17] do *sklearn* que, tal como o nome indica, cria novos atributos a partir das combinações polinomiais entre atributos.

Para as operações de selecção de atributos decidimos, mais uma vez, recorrer às implementações da biblioteca *sklearn*. Estas operações são: *SelectKBest*[18], *Recursive Feature Elimination*[19], *Principal Component Analysis (PCA)*[20], *Truncated SVD*[21], *Kernal PCA*[22] e *Fast Independent Component Analysis* [23].

No total, o submódulo é composto por 4 operações de transformação de atributos, 14 operações de geração de atributos e 6 operações de selecção de atributos. No modo *default*, a implementação desta fase executa de modo sequencial uma operação de transformação de atributos, seguida de três operações de geração de atributos e por fim uma operação de selecção de atributos. Desta forma, se tentássemos uma abordagem força bruta (tal como fizemos na fase de limpeza de dados) teríamos de avaliar 65856 configurações diferentes. Este método de força bruta não seria de todo uma boa escolha e por isso decidimos recorrer a redes neuronais para prever que operações devem ser aplicadas com base no tipo de dados que temos e nas suas características. Para treinar as redes aplicámos *Deep Q Learning*, conceito que será introduzido de seguida.

Tal como foi enunciado no Capítulo 3 e no inicio desta secção, resolvemos utilizar *Deep Q Learning* (ou *Deep Reinforcement Learning*). Esta técnica de Reinforcement Learning (RL) foi apresentada por Volodymyr Mnih et al. [20] em 2013. Junta redes neuronais a uma técnica de RL muito utilizada chamada *Q-Learning*. Para compreender o conceito de

[17] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html`

[18] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html`

[19] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html`

[20] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html`

[21] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html`

[22] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html`

[23] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html`

Deep Q Learning temos primeiro de introduzir RL e *Q-Learning*.

Reinforcement Learning é uma área de Aprendizagem Computacional composta por um conjunto de técnicas que permitem que um agente aprenda a interagir com o ambiente onde está inserido com o objectivo de maximizar a recompensa recebida. Estas técnicas focam-se em desenvolver agentes que executam acções, com base no estado actual e no ambiente. Num problema de RL existe sempre um agente, um ambiente, um conjunto de acções, um conjunto de estados e recompensas. É um processo iterativo, composto por uma série de episódios, em que o agente ao agir (i.e., executar acções) sobre o ambiente, muda o seu estado e essa alteração retorna uma recompensa que pode ser positiva ou negativa. O objectivo do agente é perceber que conjunto acções maximizam a recompensa recebida. Existem ainda dois conceitos relevantes relativos a RL: política (*policy*) e valor. *Policy* pode ser vista como a estratégia de um agente para chegar a um determinado objectivo. Valor é a recompensa total esperada se o agente seguir uma determinada *policy*. Sendo assim, o objectivo do agente é encontrar a melhor *policy*, ou seja, o conjunto de acções que maximize o valor.

Q-Learning é uma técnica de RL que tenta encontrar a *policy* óptima sem que seja necessário o agente conhecer o ambiente onde está inserido. O agente apenas precisa de saber o conjunto de acções que existem para começar a aprender. Na base desta técnica está a função *Q-value*, definida pela Equação 4.1 e a tabela Q (*Q-Table*), apresentada na Tabela 4.2.
A Equação 4.1 diz respeito ao calculo de valor Q, que é a recompensa total esperada, incluindo acções futuras, estando no estado s e executando a acção a. Este valor é calculado com a recompensa imediata r(s,a), resultante de executar a acção a, somando com o maior valor Q possível a partir do próximo estado *s'*. O gama γ é o factor de desconto que controla a influência de recompensas futuras. Os valores Q(s,a) são guardados na tabela Q, apresentada como exemplo na Tabela 4.2.
O algoritmo segue a seguinte estrutura: a tabela Q é inicializada aleatoriamente. De seguida, o agente começa a interagir com o ambiente e a realizar acções. A cada acção executada, o agente observa a recompensa r(s,a) e transita para um novo estado s'. Com o novo estado, o agente calcula o $\max_a Q(s', a)$, que significa calcular qual é a acção que irá retornar a maior recompensa estando no estado s'. Por fim, soma estes dois valores obtendo assim o valor de Q(s,a) e actualiza a tabela Q com esse valor. Assim sendo, a tabela Q irá ter as recompensas totais esperadas (valores Q), incluindo acções futuras, para cada par estado-acção. Esta tabela é uma tabela de referência para o agente seleccionar a melhor acção com base no valor Q.

$$Q(s, a) = r(s, a) + \gamma \times \max_a Q(s', a) \tag{4.1}$$

A apresentação do exemplo que se segue, torna mais fácil a compreensão da técnica. Na Tabela 4.2 está um exemplo de uma tabela Q para um contexto onde temos 3 estados e 2 acções. Imaginemos que o agente já realizou diversas acções, completando a tabela e, está no estado 2. Escolhe a acção 2, visto que é a que apresenta maior recompensa segundo a tabela (valor de 1.7), e transita para o estado 1. Ao executar a acção 2 recebe a recompensa r(estado 2, acção 2) e calcula o valor $\max_a Q(estado1, a)$ que neste caso é 3.3, visto que no estado 1 a acção 1 é a que tem maior recompensa. Então, o agente soma estes dois valores, obtendo assim o Q(Estado 2, Acção 2), e actualiza a tabela com o resultado.

Resumidamente, o agente, ao longo da execução, vai guiando-se pela tabela para perceber qual é a melhor acção para realizar num determinado estado. À medida que vai executando acções calcula o valor Q (Q(s,a)) e actualiza a tabela.

	Estado 1	**Estado 2**	**Estado 3**
Acção 1	3.3	-4.5	1.2
Acção 2	-2.6	1.7	2.6

Tabela 4.2: Q Table

Esta é a ideia base do algoritmo de *Q-Learning*. Apesar de ser um bom algoritmo para situações em que temos um conjunto de estados e um conjunto de acções possíveis pequeno, torna-se impraticável para situações onde estes conjuntos são enormes ou infinitos. Imaginemos que temos um problema com 1 milhão de estados possíveis e 3 milhões de acções possíveis. Este problema daria uma tabela composta por 1 milhão de colunas e 3 milhões de linhas. Neste caso, seria impraticável utilizar *Q-Learning* porque teríamos de testar todas as acções em todos os estados possíveis, trazendo problemas tanto de memória (necessidade de guardar uma tabela enorme) assim como de tempo (tempo necessário para explorar cada estado). No caso de termos um número de estados infinito ou um número de acções infinito, nem seria possível criar uma tabela Q completa. No fundo, o grande problema está associado com a impossibilidade de calcular o $\max_a Q(s', a)$, visto que é impossível calcular os valores Q para todas as acções num determinado estado.

Dado este problema com *Q-Learning*, investigadores da *DeepMind*, empresa mundialmente conhecida por utilizar diversas técnicas de RL, desenvolveram uma forma de inferir a recompensa de uma acção num determinado estado. Esta nova estratégia chama-se *Deep Q Learning* e junta redes neuronais à técnica *Q-Learning*. Esta técnica recorre a redes neuronais para fazer uma aproximação da função Q, possibilitando assim utilizar as redes para estimar a recompensa que uma acção tem num determinado estado.

O algoritmo base é semelhante ao algoritmo de *Q Learning*. A grande diferença nos algoritmos é que em vez do agente agente olhar para a tabela Q de forma a calcular o $\max_a Q(s', a)$, utiliza a rede neuronal para modelar a função Q e prever os valores. Ou seja, da Equação 4.1, a rede neuronal substitui o cálculo $\max_a Q(s', a)$. E, a cada iteração, o agente em vez de actualizar a tabela Q, retreina a rede neuronal com os novos valores.

O Algoritmo 1 apresenta o pseudocódigo do algoritmo base de *Deep Q Learning*. Este algoritmo, que será explicado de seguida, é utilizado para treinar a rede neuronal para que esta consiga fazer previsões acertadas acerca das recompensas associadas com as diversas acções.
No inicio do algoritmo de treino da rede, é necessário criar uma variável que damos o nome de *replay memory*, criar a rede neuronal e definir um ε e o conjunto de acções disponíveis. A variável *replay memory* guarda os pares estado-acção e a respectiva recompensa e, serve como dados de treino para treinar a rede neuronal.
Iterativamente, o agente tem a hipótese de escolher uma acção aleatória ou recorrer à rede neuronal para escolher a melhor acção. Ao escolher a acção, executa a mesma e observa a recompensa. Adiciona toda a nova informação à variável *replay memory* e a rede neuronal é de novo treinada. Com o tempo, a variável *replay memory* vai conter mais informação acerca dos estados, acções e recompensas, dando assim mais informação à rede neuronal para que esta consiga fazer previsões cada vez mais acertadas.

O ε é fundamental pois permite que novos espaços sejam explorados ao seleccionar operações aleatoriamente. No início do treino, esta variável deve ter um valor elevado de forma a darmos prioridade à escolha de operações aleatórias e explorarmos eficazmente o conjunto de acções e de estados. Ao longo do tempo, deve diminuir de maneira a darmos prioridade às decisões da rede.

Algoritmo 1: Algoritmo Base de Deep Q Learning

```
nicializar replay memory;
inicializar rede neuronal;
definir ε;
definir conjunto de ações ;
for episodio ← 1 to M do
    for passo ← 1 to T do
        Com probabilidade ε seleccionar uma acção aleatoriamente caso contrário
          utilizar a rede neuronal para seleccionar a melhor acção
        Executar a acção e observar a recompensa
        Guardar a transição (estado, acção, recompensa) na variável replay memory
        Treinar a rede neuronal com a replay memory
    end
    diminuir ε
end
```

Para finalizar, é importante notar que existem dois tipos de redes que podemos treinar. Na figura 4.2 estão presentes estes dois tipos. Diferem no que recebem como *input* e no que retornam. A rede do lado esquerdo, recebe o par (estado, acção) e é treinada para retornar a recompensa associada a esse par. A rede do lado direito, recebe apenas o estado actual e é treinada para retornar a recompensa associada a cada acção possível.
No primeiro caso, cada vez que o agente desejar saber qual é a melhor acção, o mesmo terá de utilizar a rede para todas as acções disponíveis de forma a saber qual é a acção que tem a maior recompensa. No segundo caso, o agente apenas precisa de utilizar a rede neuronal uma vez porque a rede já retorna a recompensa esperada para todas as acções.

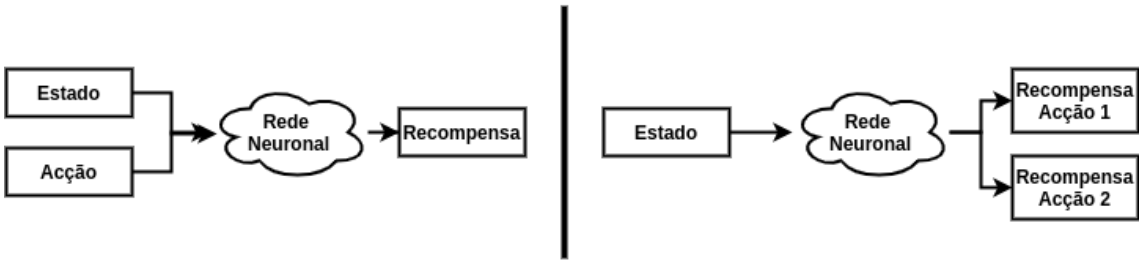


Figura 4.2: Tipos de redes neuronais

O objectivo nesta fase é então utilizar agentes treinados com *Deep Q Learning* para seleccionar as melhores operações de engenharia de atributos a aplicar aos dados.
Tal como vimos, este submódulo é composto por três tipos de operações de engenharia de atributos: transformação de atributos, geração de atributos e selecção de atributos. Estes três tipos diferem bastante no que toca às operações e aos atributos que são transformados/criados/seleccionados, e por essa razão foi determinado criar um agente para cada tipo de operação. Desta forma, treinámos 3 agentes para problemas de classificação e 3 agentes para problemas de regressão. Foi necessário separar os problemas de classificação e de regressão porque a recompensa tem proporções diferentes. Em classificação utilizamos a *accuracy* e em regressão utilizamos o erro absoluto médio para averiguar se uma acção

(escolha de uma operação de engenharia de atributos) foi bem sucedida.

Formulando o problema de RL, temos:

- `Estado:` um cientista de dados analisa as características dos dados e avalia quais são as melhores operações a aplicar. De forma a replicarmos esse método, determinámos que o estado é composto pelas características (*meta-features*) dos dados. Assim, o modelo faz previsões com base nos dados em questão. Para extrairmos as características utilizamos métodos da ferramenta *pymfe*[24]. O estado criado é composto por 29 medidas estatísticas que descrevem as propriedades dos dados;
- `Conjunto de acções:` o conjunto de acções disponível é igual ao conjunto de operações disponível para cada tipo de operações de engenharia de atributos. No caso dos agentes de transformação de atributos existem 4 acções, no caso dos agentes de geração de atributos existem 14 acções e no caso dos agentes de selecção de atributos existem 6 acções.
- `Recompensa:` o objectivo é fazer com que a performance de previsão obtida aumente com a aplicação de uma determinada operação. Em função disso, a recompensa é calculada com a performance obtida com os dados originais e a performance obtida com os dados transformados. No caso de classificação, a recompensa é a diferença entre a performance dos dados transformados com a performance dos dados originais, pois queremos maximizar o valor de *accuracy*. No caso de regressão, a recompensa é a diferença entre a performance dos dados originais com os dados transformados, pois queremos minimizar o erro médio absoluto. Para calcular a performance dos dados foi utilizado o algoritmo de aprendizagem *Extra-Trees*

Em relação ao tipo de rede neuronal, optámos pelo método que está no lado direito da Figura 4.2. Cada rede neuronal treinada, recebe as características dos dados e retorna a recompensa para cada uma das acções possíveis. Escolhemos este tipo de rede porque para cada tipo de engenharia de atributos temos diversas operações possíveis, e consequentemente seria pouco eficaz utilizar a rede neuronal para prever a recompensa de todos os pares (estado actual, acção). Por exemplo, se estamos num determinado estado e existem 14 operações (acções) possíveis, teríamos de utilizar a rede neuronal 14 vezes de forma a sabermos que acção tem a maior recompensa.

Para treinar o agente, recorremos a vários conjuntos de dados de exemplo. Estes dados servem o propósito de o agente aplicar as várias acções sobre os mesmos e verificar que acções retornam a maior recompensa. Os agentes de classificação foram treinados com 60 conjuntos de dados (30 de classificação binária e 30 de classificação multiclasse) e os agentes de regressão foram treinados com 42 conjuntos de dados.

O treino de cada uma das redes, implementadas com a ferramenta *tensorflow*, teve como base o Algoritmo 1 acima exposto. O algoritmo personalizado para o nosso problema, está descrito na Figura 4.3. Para cada conjunto de dados, o algoritmo foi executado oito vezes, isto é, o agente aplicou operações aos dados e observou a recompensa oito vezes para cada conjunto de dados.
O algoritmo começa por extrair as *meta-features* dos dados para definir os estado actual. De seguida, o agente ou escolhe uma acção aleatória ou recorre à rede para fazer a previsão

[24] `https://pypi.org/project/pymfe/`

da melhor acção. Após determinar a acção, o agente aplica a acção ao conjunto de dados, transformando assim o conjunto de dados, e calcula a recompensa associada. Isto é, calcula o ganho (se houve algum) no desempenho ao aplicar a acção. Por fim, antes do inicio de um novo episódio, toda a informação (estado, acção, recompensa) é adicionada à variável *replay memory* e a rede é treinada com os dados presentes nesta variável.

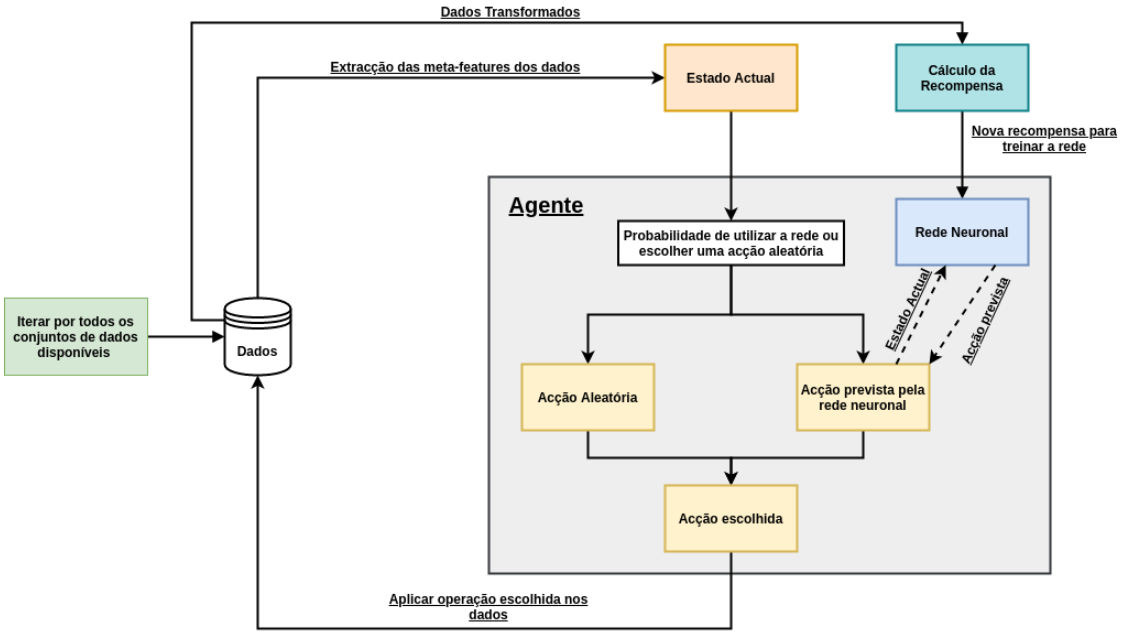


Figura 4.3: Treino de cada agente

Com este algoritmo, recriamos a forma como um cientista de dados aprende com a experiência. Este, ao longo da sua carreira, vai ganhando experiência e intuição ao aplicar operações de engenharia de atributos. E é precisamente isto que o agente faz. Ao longo das iterações, consegue perceber, através da experiência, que acções (operações de engenharia de atributos) são mais eficazes em diferentes tipos de dados.

Neste ponto, apenas falta analisar a forma de como tudo é posto em prática.
Temos então um agente para cada tipo de operação de engenharia de atributos. Cada um destes agentes deve utilizar as redes neuronais para inferir sobre a possível recompensa de cada acção (operação) e depois decidir qual é a operação que deve de aplicar. De forma a não dependermos apenas das previsões de uma rede, para cada agente foram guardadas três redes. Estes redes foram guardadas em fase diferentes do treino e foram treinadas com maior ou menos número de dados. Assim, a decisão do agente não depende apenas de uma rede que pode estar enviesada devido dos dados com que foi treinada.

Na Figura 4.4 estão descritos os passos que um determinado agente faz na altura de averiguar qual será a melhor operação a aplicar aos dados. O agente tanto executa as três acções que têm melhor recompensa estimada pelas redes como também executa uma acção aleatória.
No caso das redes, o agente recebe os dados e constrói o estado (conjunto de *meta-features*). De seguida, fornece o estado a cada uma das redes e recebe as recompensas previstas pelas redes. Para cada uma das redes, averigua qual é a operação que tem maior recompensa e aplica essa acção nos dados. Depois, calcula a recompensa real de cada uma das acções aplicadas. No caso da acção aleatória, apenas escolhe uma acção de forma aleatória e

aplica essa operação nos dados, calculando de seguida a recompensa real.
No fim, o agente tem quatro recompensas reais resultantes de executar as quatro acções (três acções previstas pelas três redes e uma acção aleatória) e, verifica qual é acção que obtém maior recompensa real, escolhendo essa acção.
Desta forma, conseguimos avaliar a recompensa de quatro operações diferentes que são aplicadas nos dados. Utilizamos as escolhas das redes e a escolha aleatória para não ficarmos presos num máximo local e explorarmos diferentes operações.

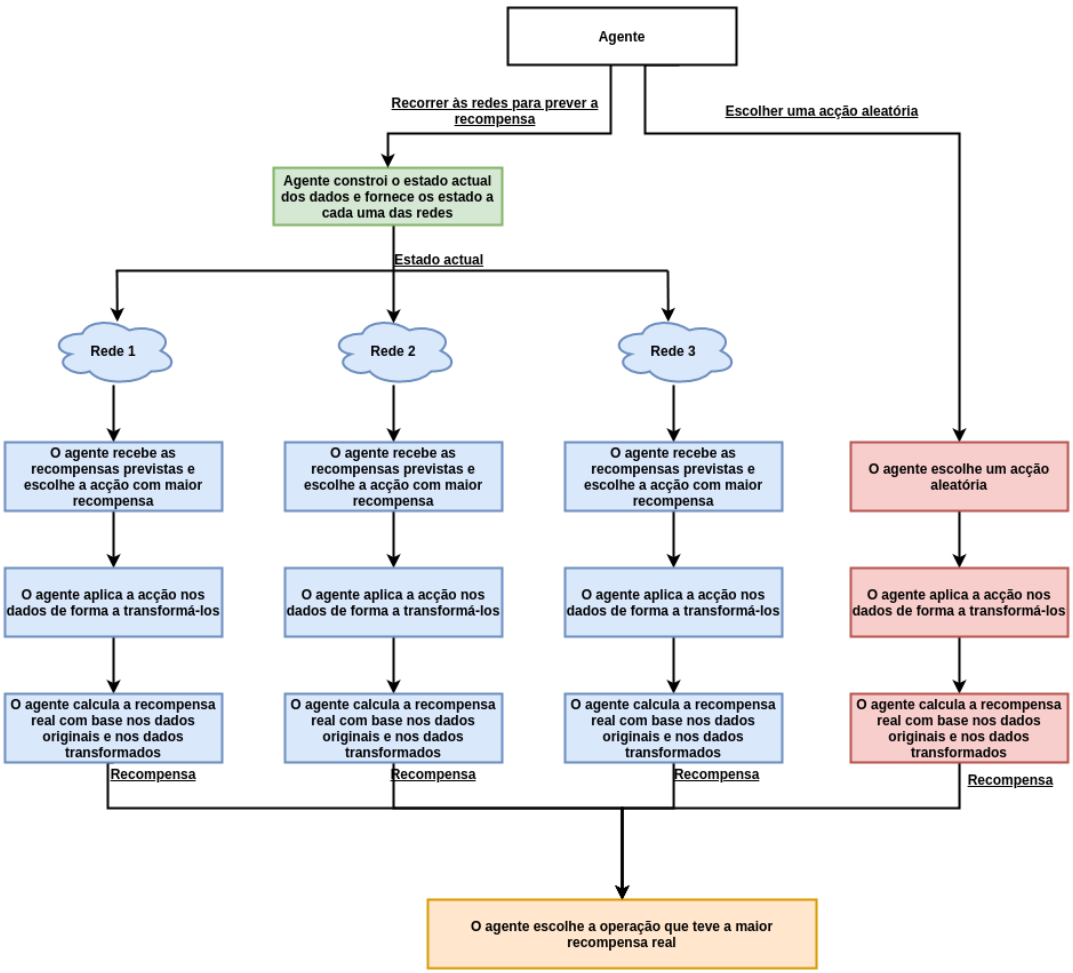


Figura 4.4: Decisão de um agente acerca da melhor operação

Nesta fase apenas falta apresentar todas as partes em conjunto. Na Figura 4.5 estão descritos os passos do algoritmo de engenharia de atributos, caso o utilizador decida executar os três passos. Primeiro, é aplicada a operação de transformação de atributos. De seguida, são aplicadas três (número *default* mas configurável) operações de geração de atributos. E, por fim, são seleccionados os melhores atributos com recurso à operação definida pelo agente de selecção de atributos. Notar que, é possível configurar que tipos de engenharia de atributos entram no algoritmo. Por exemplo, é possível apenas executar transformação de atributos ou transformação de atributos em conjunto com geração de atributos.

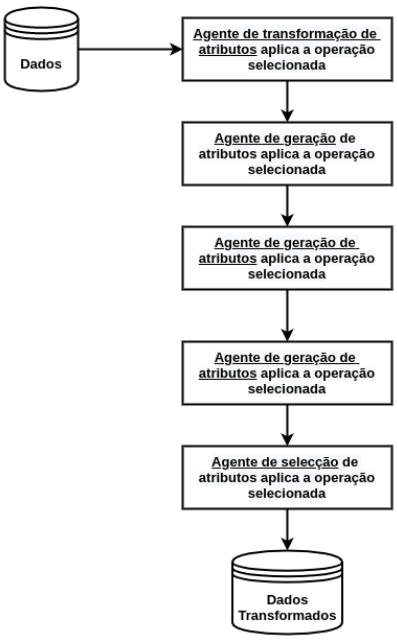


Figura 4.5: Fluxograma do Algoritmo do Submódulo de Engenharia de Atributos

4.4 Implementação do submódulo de selecção do modelo

Neste submódulo pretendemos seleccionar, de forma automatizada, um algoritmo de Aprendizagem Computacional e optimizar os respectivos hiperparâmetros. Como o desenvolvimento de uma ferramenta capaz de encontrar o melhor algoritmo de aprendizagem não era o foco primário deste projecto (dado que era uma tarefa bastante complexa), optou-se por utilizar uma ferramenta exterior que automatize este processo.

Existem várias ferramentas eficazes que fornecem métodos para automatizar a tarefa de selecção do modelo. As principais diferenças entre as ferramentas é o espaço de procura e a forma como o procuram. Inicialmente, a escolha para a ferramenta a utilizar neste submódulo estava entre duas ferramentas: *Auto-sklearn*[25] e *TPOT*[26]. Como tal, foram realizados testes de forma avaliar as duas ferramentas. Em termos de resultados, ambas as ferramentas obtiveram resultados muito semelhantes, sugerindo assim que a diferença entre os resultados de ambas não seria um bom factor de decisão. A grande diferença entre ambas é que o *TPOT* retorna o código *pytho* do *pipeline* encontrado, permitindo assim que o cientista de dados configure o algoritmo à sua maneira de forma a obter resultados ainda melhores. Dada esta grande vantagem, foi escolhido recorrer à ferramenta *TPOT*.

Em relação à implementação deste submódulo, esta foi relativamente simples dado que apenas foi necessário recorrer aos métodos já implementados pela ferramenta. Este submódulo recebe o conjunto de dados e utiliza a ferramenta para encontrar o algoritmo de Aprendizagem Computacional que obtém melhores resultados. No fim da execução, retorna o algoritmo treinado e guarda num ficheiro de *python* o código referente à construção do algoritmo.

[25] `https://automl.github.io/auto-sklearn/master/`
[26] `https://epistasislab.github.io/tpot/`

A ferramenta TPOT utiliza algoritmos evolucionários para testar várias combinações de algoritmos e retornar a melhor combinação. É bastante configurável, permitindo configurar os parâmetros que influenciam os algoritmos evolucionários. Decidimos então, que também seria permitido ao utilizador do módulo implementado configurar os parâmetros do TPOT. Assim sendo, este submódulo recebe não só o conjunto de dados como uma possível configuração dos parâmetros do TPOT. Os parâmetros são os seguintes:

- `Número de gerações:` por *default* 100
- `Tamanho da população:`por *default* 100
- `Número de descendentes:` por *default* 20
- `Taxa de mutação:`por *default* 0.9
- `Taxa cruzamento:`por *default* 0.1

4.5 Implementação do assistente virtual

Tal como observamos no capítulo anterior, a intenção era integrar o módulo no sistema Lexia que estava a ser desenvolvido. Como tal não foi possível, resolvemos implementar um assistente virtual simples como prova de conceito.

O objectivo do assistente virtual é receber mensagens do utilizador, extrair a intenção das mesmas e executar a função desejada do módulo AutoML. Como vimos no Capítulo 3, a comunicação entre o assistente virtual e o módulo AutoML é assegurada por meio de um serviço REST, que permite a execução de funções do módulo com recurso a pedidos HTTP.

No que toca ao assistente virtual, a implementação foi feita com recurso à ferramenta *Rasa*[27]. Esta, permite o desenvolvimento de assistentes virtuais de forma bastante simples e intuitiva, algo importante pois apenas queremos uma prova de conceito.

Esta ferramenta, funciona à base da extracção de intenções presentes nas mensagens, e a partir das intenções extraídas executa acções associadas às intenções. Para criar um assistente virtual apenas precisamos de definir as intenções possíveis, os dados de treino e as acções.
As intenções descrevem o que o utilizador quer realizar. Estas são extraídas das mensagens do utilizador. Por exemplo, no caso da mensagem "Quero limpar os dados", a intenção seria algo relacionado com limpeza de dados. No nosso caso, as intenções definidas dizem respeito aos métodos que o assistente virtual tem a capacidade de executar: executar o passo de limpeza de dados, executar uma operação específica de limpeza de dados, executar o passo de engenharia de atributos, executar uma operação específica de engenharia de atributos, executar o passo de selecção do modelo, executar um modelo em específico e executar os três passos sequencialmente.
Depois de definirmos as intenções criamos os dados de treino. Estes dados, representam exemplos de mensagens que o utilizador poderá dizer. Para cada intenção definida, temos

[27] `https://rasa.com`

de criar vários exemplos - por exemplo, uma frase de exemplo para a intenção de executar limpeza de dados poderia ser "Quero limpar os dados". Os modelos de NLU (Natural Language Understanding) são treinados com estes dados de forma a conseguirem inferir que intenções estão presentes nas mensagens do utilizador.
Por fim, apenas precisamos de definir os métodos que vão ser chamados quando as intenções são extraídas. No contexto do *Rasa*, estes métodos são chamados de acções. As acções são executadas sempre que uma determinada intenção é extraída. No nosso caso, as acções são métodos que utilizam o serviço REST do módulo AutoML para executar as funções do módulo. No fim da execução das funções do módulo, as acções recebem os resultados do módulo e estes são mostrados ao utilizador.

O serviço REST foi implementado com recurso à ferramenta *Flask*. Esta ferramenta é simples e tem um curva de aprendizagem relativamente pequena. Dois factores muito importantes visto que apenas queremos ter uma prova de conceito. Foram então implementados os métodos essenciais para satisfazer as necessidades do assistente virtual. Todos estes métodos são métodos HTTP, mais especificamente GET, que utilizam as funções do módulo AutoML.

Apesar de no início não estar prevista a implementação de um serviço REST, consideramos que este é uma mais valia ao módulo AutoML. Deste modo, este módulo pode ser utilizado directamente num sistema como um pacote de *python* ou pode ser utilizado por meio de pedidos HTTP. Esta nova adição, deixa o módulo mais versátil, permitindo assim a sua utilização sem grandes complexidades de integração.

Capítulo 5

Testes e Resultados

Neste capítulo serão apresentados os testes realizados para validar o módulo AutoML assim como as funcionalidades do assistente virtual.

O capítulo inicia com a Secção 5.1 que detalha todos os pormenores acerca das condições em que os testes foram executados. De seguida, na Secção 5.2, é avaliado o potencial impacto que certas operações de limpezas de dados podem ter nos dados, visto que, a escolha automatizada das mesmas pode levar à adição de ainda mais erros nos dados. As duas secções seguintes dizem respeito ao testes realizados aos submódulos do módulo AutoML implementado. Estas duas secções (Secção 5.3 e Secção 5.4) têm a mesma ordem: resultados de classificação, resultados de regressão e acabam com uma análise geral ao submódulo em questão. A Secção 5.3 é referente aos testes do submódulo de limpeza de dados e respectivos resultados. A Secção 5.4 descreve os testes do submódulo de engenharia de atributos e respectivos resultados. Por fim, a Secção 5.5 serve o propósito de apresentar todas as funcionalidades do assistente virtual.

Todos os resultados estão presentes no apêndice. Os resultados referentes a problemas de classificação estão no Apêndice .2 e os resultados referentes a problemas de regressão estão no Apêndice .3.

5.1 Especificação dos Testes

Esta subsecção serve o propósito de introduzir as condições em que foram feitos os testes. De modo geral, pretendemos verificar e testar o efeito que as operações do módulo AutoML têm nos dados, comparando a performance obtida dos dados originais com a performance obtida com os dados que passaram pelas operações do módulo. A performance diz respeito aos resultados obtidos da execução de vários algoritmos de aprendizagem computacional. Assim sendo, é necessário introduzir os dados, os algoritmos de aprendizagem e as métricas utilizadas.

Em respeito aos conjuntos de dados, decidimos testar o módulo com 14 conjuntos de dados. Metade destes conjuntos são problemas de classificação, binária ou multiclasse, e a outra metade são problemas de regressão. De forma a testar a eficiência do módulo, mais em concreto do submódulo de limpeza de dados, foram seleccionados dados sujos que contêm

Dados	Número de Atributos	Número de Amostras	Número de Classes
Adult	15	48842	2
Airlines	8	30000	2
Car	7	1728	3
CMC	10	1473	3
CRX	15	690	2
Magic04	11	19020	2
Sat	36	4435	7
Baseball	15	1232	-
Bike Sharing	16	731	-
Houses	9	20640	-
Kin8	9	8191	-
Space	7	3107	-
Water	39	527	-
Wind	15	6574	-

Tabela 5.1: Dados Recolhidos

vários valores em falta, valores categóricos, entre outros erros. A Tabela 5.1 descreve as características principais dos dados utilizados.

Para avaliar o efeito das operações aplicadas aos dados, foram utilizados os seguintes algoritmos de aprendizagem computacional:

- `Algoritmos de Classificação:` *Support Vector Classification, Random Forest Classification* e TPOTClassifier
- `Algoritmos de Regressão:` *Support Vector Regression, Regressor Forest Regressor* e TPOTRegressor

Todos os algoritmos foram implementados com os seus valores *default*, sendo que a única alteração foi limitar o tempo de execução disponível dos algoritmos da ferramenta TPOT para 5 minutos.
Importa referir que, a ferramenta TPOT apenas foi incluída nos testes porque faz parte do submódulo de selecção do modulo e que poderá não ser o melhor objecto de comparação. Ou seja, esta ferramenta, que é utilizada no submódulo de selecção do modelo, tenta encontrar o melhor algoritmo de aprendizagem computacional e, consequentemente, a cada execução poderá encontrar algoritmos diferentes produzindo assim resultados diferentes. Por exemplo, na execução com os dados originais poderá retornar o algoritmo *XGBClassifier* e na execução com os dados transformados poderá retornar o algoritmo *LogisticRegression*. Neste sentido, a comparação seria feita com resultados que foram obtidos com classificadores diferentes. Esta ferramenta foi incluída nos testes porque faz parte do submódulo da selecção do modelo e, deste modo, conseguimos avaliar os resultados da execução deste submódulo em conjunto com os outros submódulos.

Relativamente à avaliação dos resultados, foram calculadas duas métricas para cada tipo de problema. Para problemas de classificação foi calculado a *accuracy* e o *f1-score*. Para problemas de regressão foi calculado o erro médio absoluto (Mean Absolute Error (MAE)) e o erro percentual absoluto médio (Mean Absolute Percentage Error (MAPE)).

Accuracy é a percentagem de previsões correctas que o algoritmo obteve. Calcula-se dividindo o número de predições corretas pelo número total de predições. Quanto maior for, melhor será a avaliação. Nas equações seguintes, as variáveis *TP, TN, FP, FN* referem-se a verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos, respectivamente

$$\texttt{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

F1-Score combina as métricas *recall* e *precision* numa única métrica. *Recall* mede a proporção de verdadeiros positivos que foram correctamente identificados e *precision* diz-nos o quão preciso foi o nosso algoritmo a prever a classe positiva. *F1-Score* é, por sua vez, a média harmónica de *Precision* e *Recall* e, é útil para situações onde temos conjuntos de dados não balanceados.

$$\texttt{F1 Score} = 2 \times \frac{precision \times recall}{precision + recall}$$

Nos problemas de regressão não são atribuídas classes a cada uma das amostras. O valor a prever é um valor contínuo. Como tal, as métricas utilizadas não podem medir se uma determinada previsão pertence a uma classe. Neste caso, as métricas precisam de medir a distância que o valor previsto está do valor real.

MAE mede o média do valor absoluto do erro. É calculado fazendo a média da diferença absoluta entre valor real (Y_i) e o valor previsto ($\hat{Y}_i$). Quanto menor for, melhor será a avaliação.

$$\texttt{MAE} = \frac{1}{n} \sum_{1}^{n} \left| (Y_i - \hat{Y}_i) \right|$$

MAPE é o equivalente de MAE em percentagem. É útil para comparar os erros de diferentes dados, pois o MAE de diferentes dados está em escalas diferentes e não fornece a possibilidade de fazer uma comparação directa. MAPE indica, na forma de percentagem, a que distância as previsões do modelo estão dos resultados reais

$$\texttt{MAPE} = \frac{100}{n} \sum_{1}^{n} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|$$

Para todos os testes presentes neste capítulo, os dados foram divididos em dois: um conjunto de treino e um conjunto de teste. O conjunto de treino inclui 80% dos dados, e serve para treinar os diversos algoritmos de aprendizagem computacional. O conjunto de teste inclui 20% dos dados e serve para testar os algoritmos de aprendizagem computacional treinados e obter os resultados.

Nos testes realizados para o submódulo de limpeza de dados e para o submódulo de engenharia de atributos, que estão descritos nas Secções 5.3 e 5.4, o procedimento acima referido foi feito duas vezes. Ou seja, para cada conjunto de dados foram realizados dois testes, sendo que o conjuntos de treino e de teste foram diferentes para cada teste.

Importa frisar que apenas foram realizados testes para o submódulo de limpeza de dados e para o submódulo de engenharia de atributos. Esta decisão prende-se com o facto de estes dois submódulos terem sido implementados com abordagens novas. Isto é, as técnicas utilizadas para estes dois submódulos foram pensadas e implementadas pelo aluno. Em oposição, o submódulo de selecção do modelo recorre a uma ferramenta externa e por isso não carece de uma avaliação profunda. Não obstante, é possível ver os resultados do submódulo de selecção do modelo porque, na execução dos testes, também foi executado este submódulo.

Todos os testes apresentados neste capítulo foram executados numa máquina com as especificações apresentadas na Tabela 5.2

Sistema Operativo	Ubuntu 18.04.4 LTS
CPU	MD Ryzen Threadripper 2950X 16-Core Processor 3.5 GHZ
GPU	GeForce RTX 2080 ti

Tabela 5.2: Especificações da máquina de testes

5.2 Impacto das operações de limpeza de dados

Nesta secção iremos abordar o impacto que três tipos operações de limpeza de dados poderão ter nos dados. A limpeza inicial de dados é um passo importante e tem de ser efectuada com cuidado de modo a não introduzirmos erros nos dados. Realizar esta tarefa de forma automatizada traz ainda mais riscos pois um algoritmo automático poderá não notar certos aspectos e erros da mesma forma que um cientista de dados nota. Esta secção tem apenas o objectivo de analisar o impacto que certas operações têm nos dados. A avaliação completa dos submódulos implementados será apresentada nas secções seguintes (Secção 5.3 e Secção 5.4).

Os tipos operações escolhidas para avaliar o impacto foram: tratamento de valores em falta, tratamento de *outliers* e tratamento de valores categóricos. Estes tipos de operações são compostos por as operações que mais alteram os dados e a sua distribuição e, desta forma, são operações de risco elevado. A escolha automatizada errada de uma destas operações pode alterar o problema em si, originando resultados não esperados.

Primeiro iremos observar o impacto das operações de tratamento de valores em falta. De seguida, iremos verificar as operações de tratamento de *outliers* e por fim, falaremos das operações que tratam dos valores categóricos.

5.2.1 Valores em Falta

Valores em falta são um grande problema em conjuntos de dados. O valor pode estar em falta porque o medidor avariou ou simplesmente porque naquela observação não existia valor para ser medido. O problema é que a maioria dos algoritmos de Aprendizagem Computacional não conseguem trabalhar com valores em falta. Neste sentido, é necessário remover os valores em falta ou de arranjar forma para imputar esses valores.
As soluções implementadas no módulo, pretendem abranger os algoritmos mais utilizados para tratar deste problema. Cada um tem as suas vantagens e desvantagens, não havendo assim um único que seja o ideal. Resumidamente, podemos observar que:

- `Remover observações:` ao remover uma determinada observação só porque essa observação contém um atributo com um valor nulo, podemos estar a perder informação importante que estava presente nos outros atributos
- `Imputar com recurso a média ou mediana:` tem a vantagem que é rápido e simples de implementar mas, sendo uma operação ao nível do atributo, não tem em conta as relações entre o atributo a ser imputado e os outros atributos. Além disso, reduz a variância do atributo, não adicionando nova informação ao mesmo.
- `Valor seguinte:` em situações onde as observações são tiradas em curtos intervalos de tempo, o valor seguinte pode ser muito parecido ao valor que temos nulo, sendo por isso uma boa técnica. Contudo, em situações onde não existe propriamente uma ordem temporal nas amostras, o uso desta técnica pode introduzir grandes erros nos dados.
- `Remover atributos:` esta técnica é por norma consensual pois estar a imputar os valores nulos de um atributo que contém uma grande quantidade de valores nulos é uma tarefa complexa que induz em erros. No nosso caso, removemos atributos com mais de 20% de valores nulos. Contudo, o valor nulo pode conter algum significado no problema em questão e, ao apagarmos o atributo perdemos informação potencialmente relevante.
- `Iterative Imputer e KNN Imputer:` estas técnicas utilizam o conjunto de dados inteiro para estimar os valores em falta. Isto traz uma grande vantagem pois não trabalham só ao nível do atributo, analisando assim as relações entre os diferentes atributos para estimar os valores em falta.

De forma a analisarmos as diferenças em termos de desempenho das várias técnicas, treinámos e testámos um algoritmo de aprendizagem computacional com os diferentes conjuntos de dados que passaram por cada uma das técnicas. O conjunto de dados original [1] não tem nenhum valor nulo permitindo assim que seja uma comparação justa. Deste conjunto original, foram aleatoriamente removidas observações de três atributos, dois numéricos e um categórico. O conjunto de dados com valores nulos, passou por cada uma das técnicas e os conjuntos de dados resultantes foram utilizados para treinar e testar com recurso ao algoritmo *Random Forest Regressor*. Na Figura 5.1 estão presentes o erro médio absoluto obtido com o conjunto de teste para cada uma das técnicas.
Observamos que a técnica Iterative Imputer obteve claramente resultados melhores, seguida da técnica KNN Imputer. Estes resultados confirmam que estas técnicas têm vantagem no facto de estimarem o valor em falta com base nos outros atributos, analisando as relações entre os diferentes atributos.

[1] `https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume`

O resto das técnicas tem um impacto negativo, aumentando o erro médio absoluto em comparação com os dados originais. Porém, não podemos concluir que estas técnicas irão ter sempre um impacto negativo porque estes testes apenas são relativos a um conjunto de dados. Todas as técnicas apresentadas têm as suas vantagens e desvantagens, e estas dependem muito do conjunto de dados e do problema a resolver.

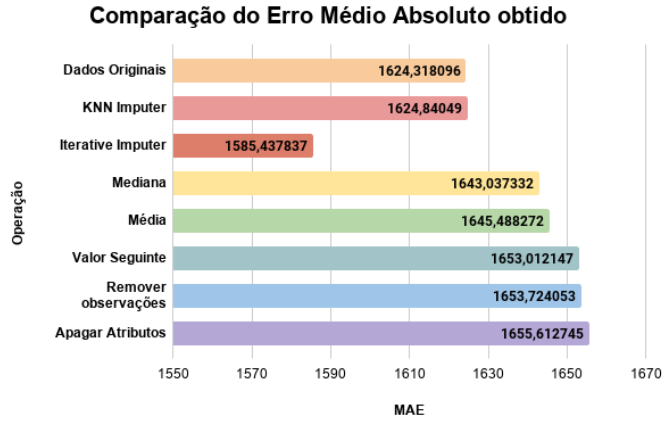


Figura 5.1: Comparação do desempenho obtido pelas diferentes técnicas

Na Figura 5.2 está exposto o gráfico de densidade que mostra a densidade do atributo numérico *clouds_all*. A linha azul, que está quase ofuscada pela linha vermelha, representa a densidade dos valores deste atributo nos dados originais. A linha que mais se aproxima da linha original, é referente à técnica do Valor Seguinte. É natural que, neste caso, a substituição pelo valor seguinte aproxime-se muito ao original porque estamos a falar de um conjunto de dados temporal, com observações que foram tiradas com intervalos de uma hora. Reparamos ainda que as técnicas Iterative Imputer e KNN Imputer aproximam bastante bem, e as técnicas Média e Mediana criam uma grande concentração de valores à volta da média e mediana, tal como seria de esperar.
A Figura 5.3 representa o erro médio absoluto entre o valores reais do atributo *clouds_all* e os valores estimados. Este gráfico tem bastante parecenças com o gráfico da Figura 5.1, no sentido em que as técnicas com o menor erro médio absoluto, isto é, as técnicas que estimam melhor o valor, são as técnicas que obtêm os melhores resultados de performance. Contudo, estes erros ainda são consideráveis. Esta variável está compreendida entre 0 e 100, e um erro médio de 32.39 ou até de 19.25 pode influenciar bastante a tarefa de previsão.

Concluindo, é bastante importante notar que não existe nenhuma técnica que seja boa para todos os tipos de problema. Por essa razão, testamos todas as técnicas e averiguamos qual é a melhor. Ao fazer isto, conseguimos adequar a escolha de operações ao conjunto de dados particular a ser testado.

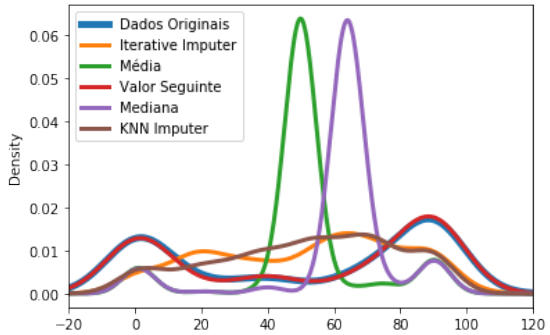


Figura 5.2: Gráfico de densidade do atributo clouds_all

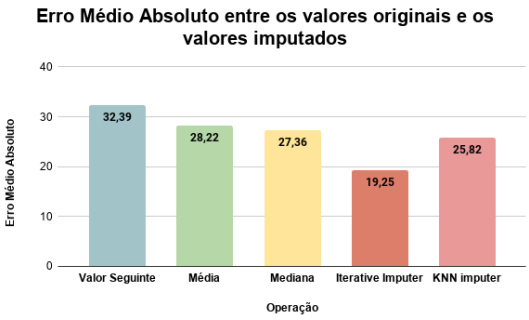


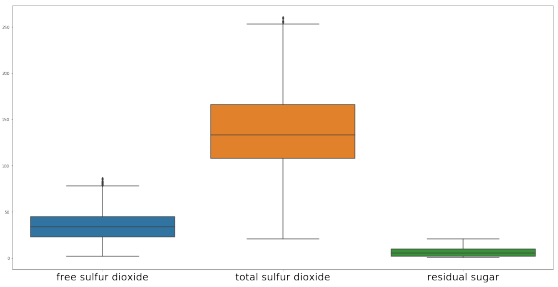
Figura 5.3: Erro médio absoluto entre os valores reais e os valores estimados

5.2.2 Outliers

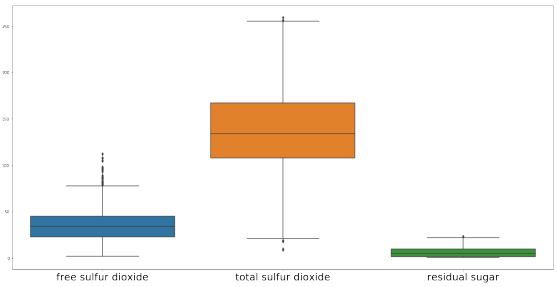
O tratamento de *outliers* é por norma consensual, sendo que a opinião generalizada é que se devem remover estes valores absurdos. O problema aqui reside em definir o que são outliers num determinado atributo. Esta definição é diferente de problema para problema, e de conjunto de dados para conjunto de dados. Alguns valores podem parecer, à primeira vista, *outliers* mas com uma análise mais profunda dos dados e do contexto percebemos que não são *outliers*. Realizar esta operação de detecção de *outliers* de forma automatizada poderá significar a detecção de valores errados. Por exemplo, o salário de o patrão de uma empresa que é 10 vezes maior que o salário médio da empresa, poderá parecer *outlier* para uma máquina que usa métodos estatísticos para detectar, mas para um cientista de dados que conhece o contexto o mesmo valor não é *outlier*.
Existem vários métodos que detectam valores absurdos e nós decidimos implementar dois: *local outlier factor* e outro com recurso ao z-score. Estes dois métodos detectam *outliers* de forma diferente e por isso a definição de *outlier* vai ser diferente. Assim, temos menos hipótese de fazer detecções erradas. Temos ainda a opção de não remover os outliers caso seja essa a melhor alternativa.

Para exemplificar o trabalho das duas técnicas, aplicámos ambas ao conjunto de dados *Wine Quality*[2]. Uma boa forma de visualizar *outliers* é com recurso aos diagramas de caixa representados nas figuras seguintes. Na Figura 5.4c estão representados 3 atributos dos dados originais. E nas Figuras 5.4b e 5.4a, estão representados esses mesmos atributos depois de passarem pela operação *Local Outlier Factor* e pela operação *Z-Score*, respectivamente.
Analisando os dados transformados, verificamos que a operação *Local Outlier Factor* removeu 59 observações e a operação *Z-Score* removeu 411 observações. Através destes resultados e da observação dos gráficos das figuras seguintes, é possível confirmar que as operações definiram o que é um *outlier* de forma diferente. Era exactamente isto que queríamos ao implementar duas técnicas, porque desta forma temos menos probabilidades de cair no erro de detectar erradamente *outliers*.

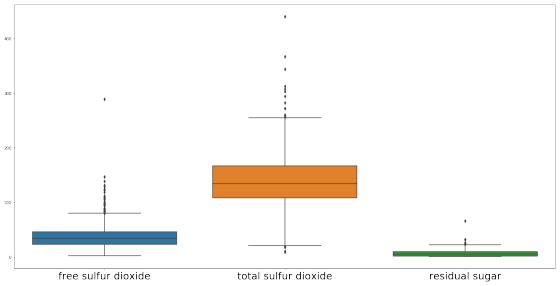
[2] `https://archive.ics.uci.edu/ml/datasets/wine+quality`



(a) Diagrama de caixa dos dados a seguir à operação z-score



(b) Diagrama de caixa dos dados a seguir à operação Local Outlier Factor



(c) Diagrama de caixa dos dados originais

Figura 5.4: Distribuição de três atributos

Para comparar o impacto directo de cada uma das opções (não remover, remover com *Local Outlier Factor* ou remover com *Z-score*) no desempenho de um algoritmo de Aprendizagem Computacional, recorremos novamente ao algoritmo *Random Forest*. Neste caso, o problema era de classificação e por isso os resultados dizem respeito à *accuracy* da previsão no conjunto de teste. Na Figura 5.5 estão representados os resultados obtidos.
A partir da mesma, verificamos que ambas as técnicas melhoraram os resultados obtidos ao removerem valores absurdos que levam o algoritmo a entender erradamente os dados e a fazer previsões incorrectas. Apesar da diferença grande entre o número observações removidas pelo *Local Outlier Factor* e o número observações removidas pelo *Z-Score*, os resultados obtidos são bastante semelhantes.

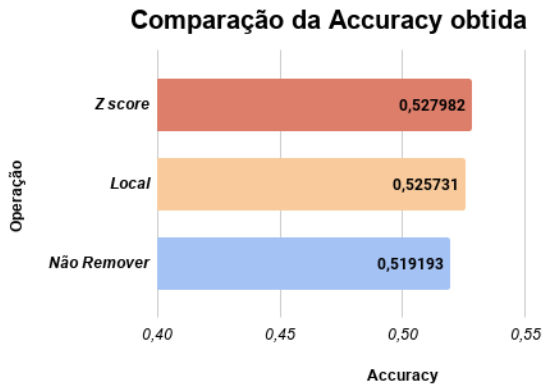


Figura 5.5: Comparação do desempenho obtido pelas diferentes técnicas

5.2.3 Transformação de Atributos Categóricos

No que toca à transformação de atributos categóricos existem duas técnicas que são muito utilizadas: *Ordinal Encoding* e *One Hot Encoding*.
Na primeira, os valores categóricos são transformados para valores numéricos entre 0 e o número de valores únicos pertencentes a um determinado atributo. Por exemplo, um atributo com os valores ['cão','gato', 'cão','leão','gato'] irá ser transformado em [0,1,0,2,1]. Esta codificação tem as suas desvantagens. Em concreto, recorrendo ao exemplo anterior, um algoritmo de aprendizagem poderá inferir que $0 < 1 < 2$, o que na realidade significa que cão $<$ gato $<$ leão. Poderá ainda deduzir que a média entre um cão e um leão é um gato. Este tipo de deduções fazem com que o algoritmo de aprendizagem computacional aprenda incorrectamente o domínio dos dados e do problema. Ainda assim, esta técnica é eficaz em situações que de facto verifica-se uma ordem nos valores únicos (por exemplo, ['primeiro','segundo', 'primeiro', 'terceiro']).
Na segunda técnica, é criado um atributo para cada valor único. Estes atributos criados são binários, sendo que contêm 1 caso a observação pertence ao valor único em questão e contêm 0 caso contrário. A grande vantagem desta técnica é que cria atributos binários em vez de ordinais. Contudo, a desvantagem é que para atributos que tenham muitos valores únicos, o número de atributos novos criados é enorme, levando à 'maldição da dimensionalidade'.

De forma a contornar as desvantagens de cada um foram implementadas no módulo 3 opções: só *Ordinal Encoding*, só *One Hot Encoding* ou os dois juntos. Este último, recorre a *Ordinal Encoding* para atributos que tenham mais do que 15 valores únicos e recorre a *One Hot Encoding* caso contrário. Assim, conseguimos evitar a criação de muito atributos com *One Hot Encoding* caso o atributo tenha muitos valores categóricos.

Para comparar o impacto directo de cada uma das opções no desempenho de um algoritmo de Aprendizagem Computacional, recorremos novamente ao algoritmo *Random Forest*. O conjunto de dados utilizado foi o *Adult*[3] que contém 13 atributos categóricos. Analisando os resultados obtidos, representados na Figura 5.6, verificamos que não existem grande diferenças entre as três opções para este conjunto de dados em particular. Neste caso, *One Hot Encoding* tem uma ligeira desvantagem, algo que pode ser explicado pelo o número elevado de valores únicos que cada atributo tem, levando assim à 'maldição da dimensionalidade'.

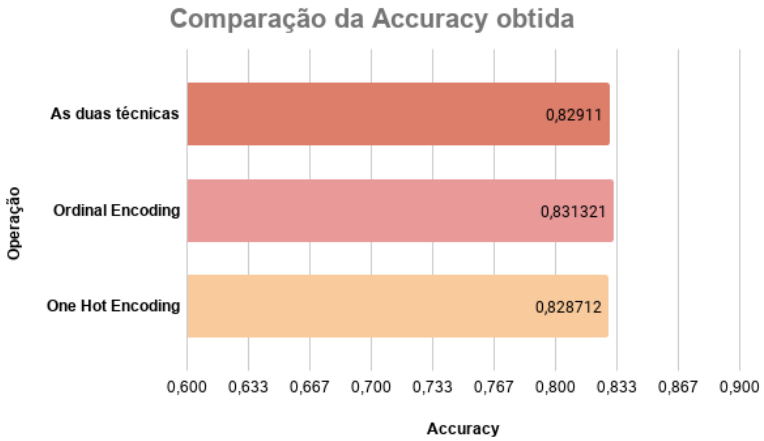


Figura 5.6: Comparação do desempenho obtido pelas diferentes técnicas

[3] `http://archive.ics.uci.edu/ml/datasets/Adult`

Esta secção serviu para mostrar o impacto potencial que estas operações podem ter nos dados. Algo importante, especialmente quando queremos seleccionar, de forma automatizada, operações que alterem os dados.
O impacto de cada operação tem nos dados depende muito dos dados, não havendo por isso uma única operação ideal que possa ser aplicada a todos os dados. Por não atendermos ao contexto dos dados, podemos cair no erro de adicionar ainda mais defeitos aos dados. De forma a minimizarmos este problema, testamos todas as combinações de algoritmos e verificamos a melhor. Com um espaço de procura bastante grande (756 combinações possíveis), conseguimos encontrar as possíveis operações que melhor se adequam aos dados, tornando o submódulo genérico apesar de não considerar o contexto dos dados.

5.3 Testes submódulo de limpeza de dados

Nesta secção iremos apresentar e analisar os testes do submódulo de limpeza de dados. Com estes testes, pretendemos avaliar a competência deste submódulo no que toca à limpeza e transformação dos dados. Para tal, a performance obtida ao executar diferentes algoritmos de aprendizagem computacional com os dados originais é comparada com a performance obtida com os dados transformados pelo submódulo.

Para obter os resultados dos dados originais, isto é, executar algoritmos de aprendizagem computacional com os dados originais, foi necessário realizar uma limpeza inicial mínima, visto que os algoritmos de aprendizagem não conseguem lidar com dados em falta e valores categóricos. A limpeza inicial mínima removeu todos os valores em falta e transformou os valores categóricos com recurso a *Ordinal Encoding*. Compreendemos que estas alterações podem influenciar a comparação, todavia eram necessárias. Esta interferência será abordada mais à frente.
Notar que estas alterações apenas afectaram os dados que foram utilizados para obter os resultados originais. O submódulo de limpeza inicial de dados recebeu os diversos conjuntos de dados com valores em falta e valores categóricos, ou seja, os dados originais sem a limpeza mínima.

A secção segue a seguinte estrutura: primeiro são apresentados os resultados de classificação, depois os resultados de regressão e concluímos a secção com uma análise geral dos resultados e do submódulo implementado.

5.3.1 Resultados Classificação

Nesta subsecção irão ser apresentadas três tabelas com os resultados para cada classificador e, no fim, serão apresentados os resultados das médias de todos os classificadores. Tal como foi descrito anteriormente, para classificação testámos os dados com três algoritmos de classificação, sendo que um deles é a ferramenta TPOT que procura pelo melhor algoritmo e, por isso, o algoritmo encontrado pode ser diferente em cada execução.
Cada conjunto de dados, foi testado duas vezes e, em função disso, os resultados apresentados são a média desses dois testes para cada classificador.

Na Tabela 5.3 estão os resultados de *accuracy* para o classificador *Support Vector Classifier*. Dos 7 conjuntos de dados, 4 obtiveram uma melhoria na performance depois de passarem pelo submódulo de limpeza de dados. Em média, os dados transformados resultaram num aumento de *accuracy* de cerca de 0.0097, com desvio padrão de 0.0278.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,845	0,651	0,875	0,508	0,885	0,868	0,887
Dados Transformados	0,853	0,677	0,944	0,51	0,876	0,869	0,864
Diferença	**0.008**	**0.026**	**0.069**	**0.002**	-0.009	-0.001	-0.023

Tabela 5.3: Resultados (accuracy) obtidos com Support Vector Classifier

A Tabela 5.4 diz respeito aos resultados obtidos com o classificador *Random Forest Classifier*. Dos 7 conjuntos de dados, apenas 2 obtiveram melhor *accuracy* que os dados originais, traduzindo-se assim em resultados menos positivos. Em média, os dados transformados, resultaram num decréscimo de performance de cerca de 0.007, com desvio padrão de 0.0152.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,830	0,69	0,982	0,513	0,901	0,881	0,911
Dados Transformados	0,834	0,688	0,938	0,512	0,897	0,881	0,906
Diferença	**0.004**	-0.002	-0.044	-0.001	-0.004	**0.0**	-0.005

Tabela 5.4: Resultados (accuracy) obtidos com Random Forest Classifier

Por fim, a Tabela 5.5 descreve os resultados obtidos com a ferramenta TPOT. Dos 7 conjuntos de dados, 6 atingiram uma performance mais elevada quando comparada com os dados originais. Em média, os dados transformados, resultaram num decréscimo de performance de cerca de 0.006, com desvio padrão de 0.023. O facto da média ser negativa, apesar da maior parte dos testes resultarem num ganho de performance, tem como principal factor os resultados obtidos com os dados `Car`, cuja ferramenta TPOT não conseguiu encontrar o algoritmo ideal, traduzindo-se numa diferença negativa grande de resultados.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,852	0,702	0,989	0,498	0,885	0,876	0,905
Dados Transformados	0,859	0,704	0,935	0,525	0,894	0,880	0,910
Diferença	**0.007**	**0.002**	-0.054	**0.027**	**0.009**	**0.004**	**0.005**

Tabela 5.5: Resultados (accuracy) obtidos com TPOT

No gráfico da Figura 5.7 estão presentes a média dos resultados obtidos com cada classificador para cada conjunto de dados. Este gráfico possibilita a comparação directa entre a performance dos dados originais e a performance dos dados transformados.
Este gráfico oferece uma comparação directa entre a performance dos dois tipos de dados. Numa primeira análise, tanto do gráfico como das tabelas, reparamos que a diferença de

performance entre os dados originais e os dados transformados não é muito significativa. Isto acontece porque os dados originais tiveram de passar pela limpeza mínima, onde foram removidos os valores em falta e foram transformados os valores categóricos, operações que o submódulo de limpeza de dados também executa. Por esse motivo, os dados originais acabam por não ser muito diferentes dos dados transformados, levando a que os resultados dos dados originais e os resultados dos dados transformados sejam muito idênticos. Este ponto e outros serão abordados com mais profundidade na subsecção 5.3.3.
A partir do gráfico verificamos que, embora a diferença seja mínima, 4 dos 7 conjuntos beneficiaram das operações aplicadas pelo submódulo. A média da diferença de *accuracy* é de 0.00047 com desvio padrão de 0.007. Importante constatar que o processo de limpeza de dados não tem necessariamente de resultar na melhoria da performance. Este, será outro ponto abordado na subsecção 5.3.3.

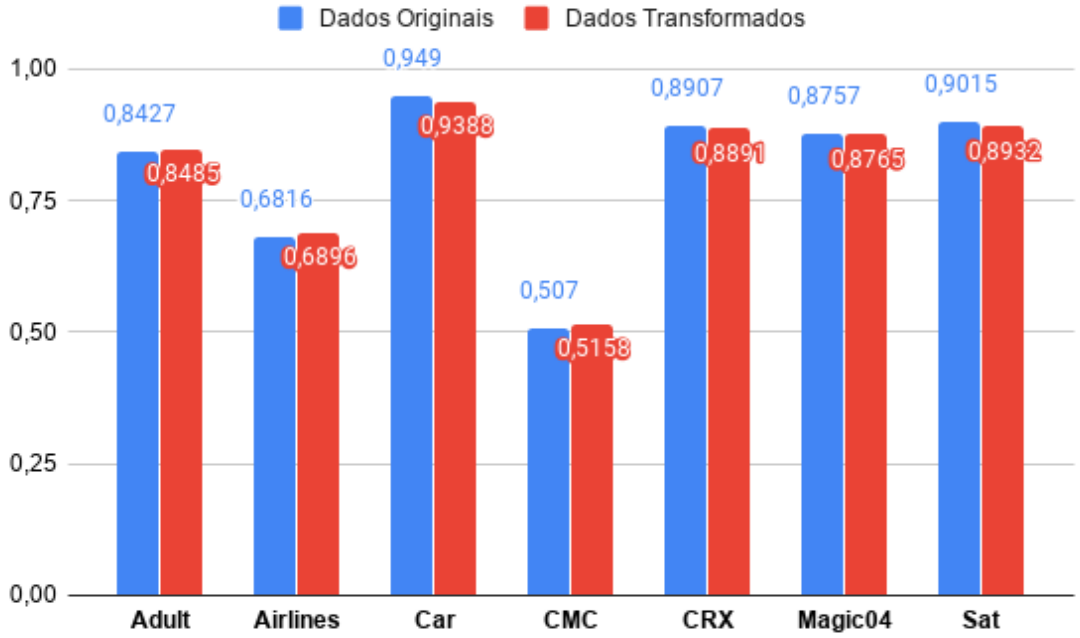


Figura 5.7: Gráfico de barras com as médias de accuracy dos classificadores por conjunto de dados

5.3.2 Resultados Regressão

Para avaliar a eficácia do submódulo em problemas de regressão, foram também testados 7 conjuntos de dados com três algoritmos de regressão. Da mesma forma que em classificação, um destes algoritmos é um algoritmo encontrado pela ferramenta TPOT que pode ser diferente em cada execução.

Nas tabelas, uma para cada algoritmo de regressão, é apresentado o erro médio absoluto (MAE) para cada conjunto de dados. No fim, de modo a apresentar os resultados todos num gráfico, recorremos à métrica erro percentual absoluto médio (MAPE) que fornece o erro médio na forma de percentagem e, como tal, permite a comparação dos erros de diferentes dados. Não foi utilizado o erro médio absoluto para o gráfico, porque este está em escalas diferentes, dependendo do conjunto de dados e, desta forma, não oferece a possibilidade de uma comparação directa.

Ambas as métricas são relativas aos erros e, por isso, de forma contrária à *accuracy*, quanto menor for o resultado melhor.

Na Tabela 5.6 são apresentados os resultados referentes ao algoritmo de regressão *Support Vector Regressor*. Dos 7 conjuntos de dados, 3 obtiveram um erro médio absoluto inferior e 4 obtiveram exactamente os mesmos resultados. O facto de 4 conjuntos de dados transformados obterem os mesmos resultados que os dados originais é devido ao problema descrito brevemente acima. Estes 4 conjuntos apenas tinham valores em falta e valores categóricos e, para obter os resultados dos dados originais foram removidos os valores em falta e foram transformados os valores categóricos com *Ordinal Encoding*. Acontece que, o submódulo de limpeza de dados, depois de avaliar várias combinações, concluiu que remover os valores em falta e transformar os valores categóricos com *Ordinal Encoding* seria a melhor combinação. Como tal, os dados originais e os dados transformados são exactamente iguais e por isso dão resultados iguais.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	65,82	1553,6	88418,6	0,06	0,08	0,46	2,39
Dados Transformados	59,15	1512.4	88418,6	0,06	0,08	0,5	2,39
Diferença	**-6,67**	**-28,43**	0	0	0	**-0.04**	0

Tabela 5.6: Resultados (MAE) obtidos com Support Vector Regression

A Tabela 5.7 apresenta os resultados obtidos com o algoritmo *Random Forest Regressor*. Dos 7 conjuntos de dados, apenas 2 obtiveram um erro médio absoluto menor depois de passarem pelo submódulo de limpeza de dados. Os resultados onde a diferença é zero ou perto de zero acontecem devido ao problema acima descrito que será explicado com mais profundidade na subsecção 5.3.3.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	19,42	70,73	31999,08	0,1	0,07	0,18	2,43
Dados Transformados	18,44	68.48	31999,08	0,1	0,07	0,3	2,42
Diferença	**-0,98**	3.36	0	0	0	0.12	**-0.01**

Tabela 5.7: Resultados (MAE) obtidos com Random Forest Regressor

Na Tabela 5.8 são apresentados os resultados obtidos com a ferramenta TPOT. Verificamos que apenas 2 conjuntos de dados obtiveram um erro médio absoluto menor.
Estes resultados são úteis para mostrar que os resultados da ferramenta TPOT não são bons para comparação pois produzem resultados diferentes dependendo do algoritmo encontrado. Por exemplo, nas tabelas acima verificamos que a diferença dos resultados do conjunto de dados *Houses* é nula. Mas, nos resultados com TPOT, verificamos que o erro médio absoluto diminuiu. Nestes conjunto de dados em particular, os dados originais e os dados transformados são exactamente iguais e por isso os resultados deviam ser muito idênticos. O problema, é que a ferramenta TPOT tenta encontrar o algoritmo de

aprendizagem computacional que obtém melhores resultados, encontrando algoritmos diferentes em execuções diferentes. Ou seja, o algoritmo que o TPOT encontrou com os dados originais foi diferente do algoritmo encontrado com os dados transformados. Desta forma, a comparação entre a performance dos dados originais com a performance dos dados transformados não é justa. TPOT foi apenas executado porque faz parte do módulo AutoML implementado e desta forma conseguimos também testar o submódulo de selecção do modelo.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	16,82	0	33042,77	0,08	0,07	0,2	2,37
Dados Transformados	16,88	0	31597,01	0,08	0,07	0,33	2,36
Diferença	0.06	0	**-1445,76**	0	0	0.13	**-0.01**

Tabela 5.8: Resultados (MAE) obtidos com TPOT

No gráfico da Figura 5.8 estão presentes a média dos resultados obtidos por cada conjunto de dados, oferecendo uma comparação directa entre o erro percentual médio dos dados originais contra os dados transformados. De notar, que aqui foi utilizada a métrica erro percentual absoluto médio porque permite comparar, na forma de percentagem, os erros obtidos pelos diferentes dados.
Dos 7 conjuntos de dados, 5 resultaram num erro percentual absoluto médio menor, apesar da diferença entre os dados originais e os dados transformados ser pouco significante. Igualmente ao sucedido em classificação, isto deve-se ao facto dos dados originais passarem antes por uma limpeza inicial mínima, não permitindo assim uma comparação muito eficaz. Este e outros pontos serão abordados na subsecção seguinte.

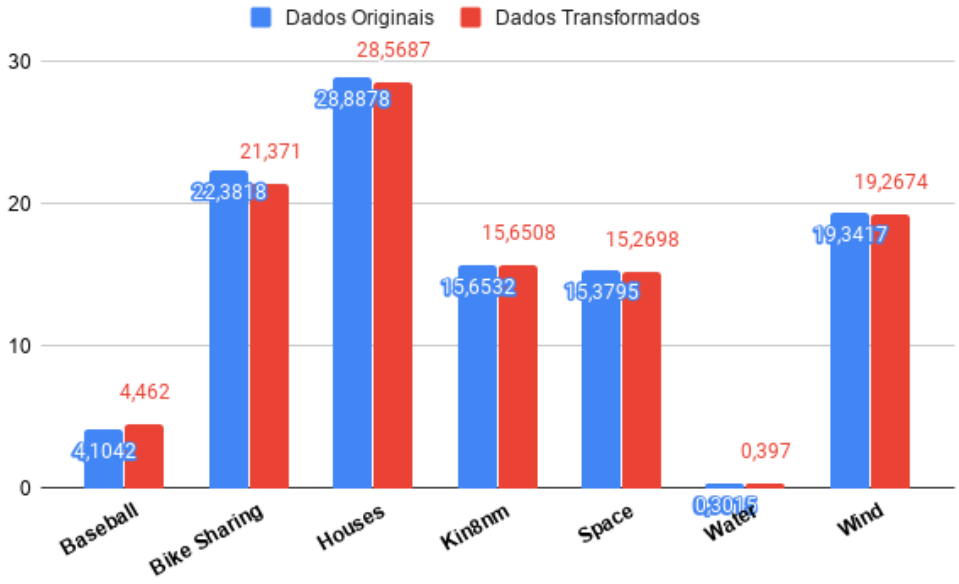


Figura 5.8: Gráfico de barras com as médias de MAPE dos regressores por conjunto de dados

5.3.3 Análise geral do submódulo

Nesta subsecção iremos analisar os resultados obtidos, percebendo o que afectou os resultados e apresentado possíveis soluções para iterações futuras do submódulo de limpeza de dados. Antes de passarmos à análise, é necessário apenas salientar dois pontos importantes no que toca à avaliação de um algoritmo de limpeza de dados.

Primeiro, é importante notar que testar correctamente a eficácia de um módulo de limpeza de dados é complicado e poderá induzir em erros. Isto acontece porque para obter a performance dos dados originais, de forma a termos algo para comparar com os dados transformados, temos de fazer uma limpeza mínima dos dados originais. No nosso caso, removemos os valores em falta e transformámos os valores categóricos. O problema, é que estas duas operações também são utilizadas no submódulo de limpeza de dados e por isso podem ser aplicas de forma a limpar e transformar os dados originais. Ou seja, podemos incorrer no risco de comparar dois conjuntos de dados, os dados originais e os dados transformados, que passaram praticamente pelos mesmos métodos de limpeza.
Mesmo que os dados contenham, por exemplo *outliers*, estes vão ser apenas uma pequena percentagem. Ao serem eliminados não vão alterar significativamente os dados porque, por norma, a percentagem de *outliers* eliminados é bastante pequena quando comparada com os dados todos. Isto, leva a que os resultados, apesar de diferentes, não tenham uma diferença muito significativa.

Segundo, uma limpeza de dados eficaz não tem necessariamente de resultar numa performance mais elevada. Dados sujos, repletos de por exemplo *oultiers*, podem fazer com que um algoritmo de aprendizagem computacional perceba erradamente a distribuição real dos dados e do problema, resultando em talvez bons resultados. O problema, é que estes aparentes bons resultados não são fidedignos porque os dados com que o algoritmo foi treinado não representam bem o problema. Ao remover os *outliers*, a distribuição dos dados é complemente diferente e por isso o problema é diferente o que pode resultar numa diferença significativa, muitas vezes negativa, na performance obtida.
Evidentemente, todas as alterações dos dados têm em vista a melhoria da performance. Contudo, certas operações, especialmente operações de limpeza de dados, podem conduzir à perda de performance. É importante notar que, as operações de limpeza de dados pretendem predominantemente remover erros presentes nos dados. Esse passo deve ser complementado com engenharia de atributos, que já tem como objectivo principal aumentar a performance dos dados.

Com estas duas notas, já será possível analisar os resultados obtidos.
Numa perspectiva global, as médias dos algoritmos de aprendizagem computacional para cada conjunto de dados, indicam que dos 14 conjuntos de dados, 10 conjuntos que passaram pelo submódulo de limpeza de dados resultaram numa melhoria de performance. Contudo, a melhoria verificada é praticamente insignificante principalmente devido ao primeiro ponto falado nesta subsecção. Nos casos de pioria, a diferença também é bastante insignificante.

No geral, os resultados são bastante positivos porque não houve grande perdas de performance quando comparando os dados originais que passaram pela limpeza mínima com os dados transformados que passaram pelo submódulo de limpeza de dados. O principal propósito deste submódulo é limpar os dados e de forma geral esse propósito foi bem conse-

guido. Através da análise dos dados transformados, é possível verificar os dados testados, foram bem limpos e foram removidos quaisquer erros que o módulo consegue detectar. Contudo, existem certos aspectos do submódulo que são problemas e que podem vir a ser melhorados.

O principal problema com o submódulo é a função de avaliação. Este submódulo avalia no máximo 756 combinações (o número depende do tipo de dados) com recurso à função *SelectKBest*. Tal como já foi descrito, esta função é maioritariamente utilizada para seleccionar os melhores atributos mas também pode ser utilizada para avaliar a importância que cada atributo tem em relação à variável a prever. No contexto do submódulo, para cada combinação, a função de avaliação soma a importância que cada atributo tem. Isto traz um problema, porque quanto mais atributos tiver um conjunto de dados, mais atributos serão avaliados e, consequentemente, maior poderá ser a soma final. Uma combinação onde foram criados atributos (por exemplo com *One Hot Encoding*) tem mais hipóteses de ter uma soma final maior do que uma combinação onde foram removidos atributos (por exemplo com a remoção de atributos com valores em falta).
A título de exemplo, nos pontos seguintes estão duas possíveis avaliações de dois conjuntos de dados resultantes de duas combinações diferentes. O primeiro conjunto de dados tem 3 atributos e a soma final é 2,15. O segundo conjunto de dados tem 11 atributos e a soma final é 2,3.

- `Conjunto de dados 1 (3 atributos):` $0.9 + 0.5 + 0.75 = \mathbf{2.15}$
- `Conjunto de dados 2 (11 atributos):` $0.1 + 0.15 + 0.2 + 0.5 + 0.3 + 0.1 + 0.05 + 0.15 + 0.25 + 0.2 + 0.3 = \mathbf{2.3}$

Através da análise das avaliações de cada atributo, é possível verificar que o conjunto de dados 1 tem atributos bem mais importantes e relevantes que o conjunto de dados 2. Apesar de apenas conter três atributos, estes três atributos têm muita informação acerca da variável a prever. Algo que não acontece com o conjunto de dados 2 onde, não existe nenhum atributo realmente importante ou relevante. Contudo, a combinação de algoritmos que resultou no conjunto de dados 2 iria ser escolhida visto que a soma final é maior.

Apesar de desejarmos seleccionar a melhor combinação de algoritmos, é difícil definir o que é o "melhor" no contexto de limpeza de dados automatizada pois, o melhor depende do domínio dos dados. Contudo, foram pensadas quatro possíveis soluções para o problema da função de avaliação:

- `Atribuição de pesos:` a ideia é atribuir pesos com base na importância que cada atributo tem. Ou seja, os atributos mais importantes iriam ter um peso maior no que toca à soma final. Deste modo, a soma de atributos pouco relevantes iria ser menor porque tinha pesos menores associados
- `Somar apenas os N atributos mais importantes:` nesta possível solução, apenas tínhamos em consideração os N atributos mais importantes, sendo que N dependia do número de atributos que o conjunto de dados original tinha. No exemplo acima, o N poderia ser 3, e desta forma o conjunto de dados 1 ia ser seleccionado.
- `Recorrer a algoritmos de aprendizagem computacional:` esta solução pretende utilizar algoritmos de aprendizagem computacional de forma a avaliar uma combinação. É bastante utilizada por outras abordagens de limpeza de dados, descritas no

Capítulo 2, mas peca no facto de para alguns dados não ser um bom avaliador porque uma limpeza eficaz poderá significar uma performance menor, tal como explicámos no inicio desta subsecção.

- `O utilizador fornece a função de avaliação:` esta solução, implementada por algumas abordagens descritas no Capítulo 2, passa por o utilizador definir a função de avaliação. Desta forma, a função de avaliação será adequada ao domínio dos dados.

O problema com a função de avaliação levou, na maioria dos testes, à escolha da operação *One Hot Encoding* como melhor operação para tratar dos valores categóricos. Esta operação é frequentemente escolhida pelo submódulo, uma vez que os conjuntos de dados resultantes são compostos por muitos atributos, o que por sua vez resulta do facto de esta operação criar um atributo para cada valor categórico único. A escolha constante desta operação pode ter resultado num decréscimo na performance em certos testes, porque estes foram realizados com dados compostos com muitos atributos levando à maldição da dimensionalidade (conceito já referido neste relatório). Por exemplo, no caso do teste dos dados *Airlines*, os dados transformados tinham 286 colunas que foram criadas pelo *One Hot Encoding*. Estes dados transformados resultaram num decréscimo de performance porque os algoritmos de aprendizagem não conseguiram aprender eficazmente devido ao número elevado de atributos.
Não quer este problema dizer que *One Hot Encoding* é mau. Apenas pretende mostrar como os resultados podem ter sido enviesados devido ao grande número de colunas que, que por sua vez, teve como principal razão a função de avaliação. *One Hot Encoding* é bastante útil e oferece bons resultados. Tem é de ser combinado com engenharia de atributos de forma a seleccionar os atributos mais relevantes.

Outro problema identificado ao efectuar a limpeza automatizada diz respeito aos atributos numéricos que são identificadores de uma determinada observação. A grande maioria dos dados, contêm um atributo que identifica uma determinada observação (por exemplo, o *id* do número de sócio). Estes identificadores por norma são valores aleatórios e bastante elevados. O problema, é que podem ser classificados erradamente como *outliers*. Por exemplo, um atributo deste género pode conter o valor 250 (o número de um sócio bastante antigo) e conter o valor 10200000 (o número de um sócio mais recente). Os algoritmos de detecção de *outliers* irão identificar erradamente o segundo número como *outlier*, porque se afasta bastante da distribuição do atributo, e observação em questão irá ser eliminada. Este problema acontece com todos os atributos numéricos cujos números não representam necessariamente algo numérico e aditivo. Uma forma simples de solucionar este problema, seria pedir ao utilizador para identificar todos os atributos que não deveriam entrar no algoritmo de detecção de *outliers* ou especificar dados deste tipo como categóricos.

Estes foram os principais problemas identificados que, com as possíveis soluções identificadas, poderiam de certa forma ser removidos ou pelo menos diminuídos. Apesar destes problemas, o submódulo de limpeza de dados consegue efectivamente limpar os dados de forma automatizada. Sendo bastante difícil de identificar o certo e o errado no que toca às operações aplicadas, consideramos que de forma geral este submódulo é bastante eficaz. Após uma análise dos dados transformados, verificamos que todos os erros e inconsistências são bem identificados e tratados pelo submódulo, existindo no entanto situações específicas de cada domínio que são mais difíceis de tratar.

Para analisar o tempo que cada teste demorou, recorremos à Tabela 5.9 onde estão reportados os tempos, em segundos, que demorou o submódulo de limpeza de dados a encontrar a melhor combinação de algoritmos para cada teste realizado. Em termos médios, o submódulo demorou cerca de 127,44 segundos a testar várias combinações. Este número é bastante positivo, especialmente tendo em conta podem ser testadas até 756 combinações de algoritmos. O tempo que levaria a um cientista de dados implementar todas as funções executadas seria muito superior. Notamos assim, que com recurso ao submódulo de limpeza de dados, o utilizador ganha bastante tempo, sendo este um dos propósitos de AutoML

Dados	**Tempo Teste 1 (s)**	**Tempo Teste 2 (s)**	**Média (s)**	**Nº de Observações**
Adult	158,541	153,298	155,91	48842
Airlines	34,637	32,952	33,794	30000
Car	7,587	7,383	7,485	1728
CMC	38,057	38,12	38,09	1473
CRX	35,509	37,42	36,464	690
Magic04	5,456	5,181	5,318	19020
Sat	8,13	6,905	7,517	4435
Baseball	16,646	16,581	16,613	1232
Bike Sharing	1031,029	995,438	1013,233	731
Houses	8,857	9,609	9,233	20640
Kin8	18,512	18,338	18,425	8191
Space	7,682	8,874	8,278	3107
Water	405,648	413,968	409,808	527
Wind	23,615	24,55	24,0825	6574
Média	128,56	126,308	127,44	10513.57

Tabela 5.9: Tempos de execução dos testes do submódulo de limpeza de dados

O gráfico da Figura 5.9 compara o número de observações dos conjuntos de dados com os respectivos tempos médios de execução. A apresentação deste gráfico serve para destacar dois pontos.
Primeiro, este gráfico mostra que o número de observações não é o único factor que influencia o tempo demorado. Naturalmente, seria de esperar que quanto mais amostras tivesse um determinado conjunto de dados, mais tempo seria utilizado para encontrar a combinação de algoritmos. Contudo, tal nem sempre acontece porque o número de combinações avaliadas depende do conjunto de dados. Isto é, se um determinado conjunto de dados não conter valores em falta, o número de combinações vai ser muito menor pois não vão ser avaliadas as operações do tratamento de valores em falta.
Segundo, o gráfico também serve para mostrar as duas operações que demoram mais tempo: tratamento de valores categóricos inconsistentes e tratamento de atributos com datas. Estas, causaram os dois picos de tempo que se observam no gráfico. O primeiro pico (409,8) deve-se ao facto do tratamento de valores categóricos inconsistentes e o segundo pico (1013,23) deve-se ao facto do tratamento de atributos com datas. Estas duas operações trabalham ao nível da amostra e são feitos cálculos individuais para cada amostra. Como tal, o tempo de execução irá ser sempre muito maior nos casos em que existam muitos valores categóricos inconsistentes ou muitos atributos com datas.

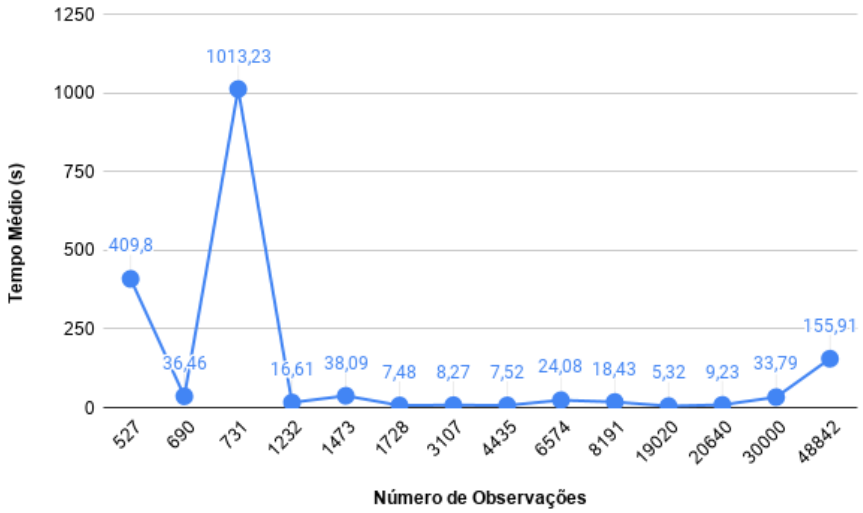


Figura 5.9: Tempo médio por número de operações

Em conclusão, o submódulo implementado cumpre os requisitos e oferece bons mecanismos para a limpeza de dados. Revisitando os objectivos de AutoML e em específico da tarefa de limpeza de dados, consideramos que foram cumpridos. Um objectivo é diminuir o tempo associado a esta tarefa. Este objectivo foi muito conseguido, tendo em conta que a média de execução é de 127,44 segundos. Um outro objectivo é ter uma limpeza inicial dos dados competente e adequada a cada tipo de dados. Este objectivo é mais difícil de medir, pois como vimos uma limpeza eficiente não resulta necessariamente numa performance melhor. Ainda assim, consideramos que ao avaliar várias combinações, conseguimos de certa forma encontrar a combinação mais adequada a cada tipo de dados, fazendo assim com que o submódulo seja bastante versátil e genérico.

5.4 Testes submódulo de engenharia de atributos

Nesta secção serão primeiramente apresentados os resultados dos testes realizados ao submódulo de engenharia de atributos e de seguida será feita uma análise ao mesmo, onde serão abordados os problemas e propostas soluções.

Os testes foram feitos com os dados limpos pelo submódulo de limpeza de dados. Isto é, os dados que foram fornecidos ao submódulo de engenharia de atributos são os dados limpos provenientes dos testes anteriores. Igualmente ao anteriormente feito, irão ser apresentadas várias tabelas com os resultados dos dados originais e os resultados dos dados transformados. Neste caso, os dados originais dizem respeito aos dados limpos pelo submódulo de limpeza de dados, e os dados transformados dizem respeito aos dados que passaram pelo submódulo de engenharia de atributos.

5.4.1 Resultados Classificação

Esta subsecção descreve os resultados obtidos dos problemas de classificação com os dados que foram transformados pelo submódulo de engenharia de atributos. Primeiro, irão ser apresentados os resultados para cada classificador e no fim, serão exibidos os resultados das médias de todos os classificadores.

A Tabela 5.10 é relativa aos resultados de *accuracy* obtidos para o classificador *Support Vector Classifier*. Dos 7 conjuntos de dados, 6 obtiveram uma *accuracy* mais elevada depois de passarem pelo submódulo de limpeza de dados. Em média, os dados transformados resultaram num aumento de *accuracy* de cerca de 0.0078, com desvio padrão de 0.0278.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,852	0,677	0,943	0,51	0,875	0,868	0,864
Dados Transformados	0,864	0,681	0,947	0,485	0,885	0,894	0,890
Diferença	**0.012**	**0.004**	**0.004**	-0.025	**0.001**	**0.026**	**0.026**

Tabela 5.10: Resultados (accuracy) obtidos com Support Vector Classifier

Na Tabela 5.11 estão representados os resultados obtidos com o classificador *Random Forest Classifier*. Neste caso, verificamos uma melhoria de performance em 5 conjuntos de dados. A média da diferença entre os resultados com os dados transformados e os resultados com os dados originais é de 0.0176 e o desvio padrão é 0.021.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,833	0,687	0,937	0,511	0,897	0,881	0,905
Dados Transformados	0,869	0,684	0,942	0,568	0,893	0,906	0,916
Diferença	**0.036**	-0.003	**0.005**	**0.057**	-0.004	**0.025**	**0.011**

Tabela 5.11: Resultados (accuracy) obtidos com Random Forest Classifier

Por fim, temos a Tabela 5.12 que apresenta os resultados obtidos com a ferramenta TPOT. Neste cenário, 6 conjuntos de dados favoreceram das transformações realizadas pelo submódulo de engenharia de atributos. As transformações resultaram, em média, numa melhoria de *accuracy* de cerca de 0.0074 com desvio padrão de 0.0327.

	Adult	**Airlines**	**Car**	**CMC**	**CRX**	**Magic04**	**Sat**
Dados Originais	0,858	0,703	0,934	0,525	0,894	0,879	0,909
Dados Transformados	0,875	0,696	0,951	0,545	0,896	0,905	0,913
Diferença	**0,017**	-0,077	**0,017**	**0,02**	**0,002**	**0,026**	**0,004**

Tabela 5.12: Resultados (accuracy) obtidos com TPOT

O gráfico da Figura 5.10 permite a comparação directa entre os resultados obtidos dos vários conjuntos de dados, oferecendo assim uma visão geral do impacto do submódulo. As barras azuis representam o valor médio de *accuracy* obtido através da execução dos três classificadores com os dados originais e, as barras vermelhas representam o valor médio de *accuracy* obtido com os dados transformados pelas operações seleccionadas pelo submódulo de engenharia de atributos.
A partir da Figura 5.10, verificamos que 6 dos 7 conjuntos favoreceram das operações aplicadas, e que a média da diferença de *accuracy* é de 0.012 com desvio padrão de 0.0092. Existem 2 conjuntos de dados (*Adult e Magic04*) cujas transformações resultaram num aumento de *accuracy* cerca 2%, e os restantes resultados positivos mostram um aumento entre 0.25% e 1.5%. O resultado negativo, obtido com o conjunto de dados *Airlines*, é insignificante, sugerindo que as operações aplicadas ao conjunto não resultaram em grandes alterações no mesmo.

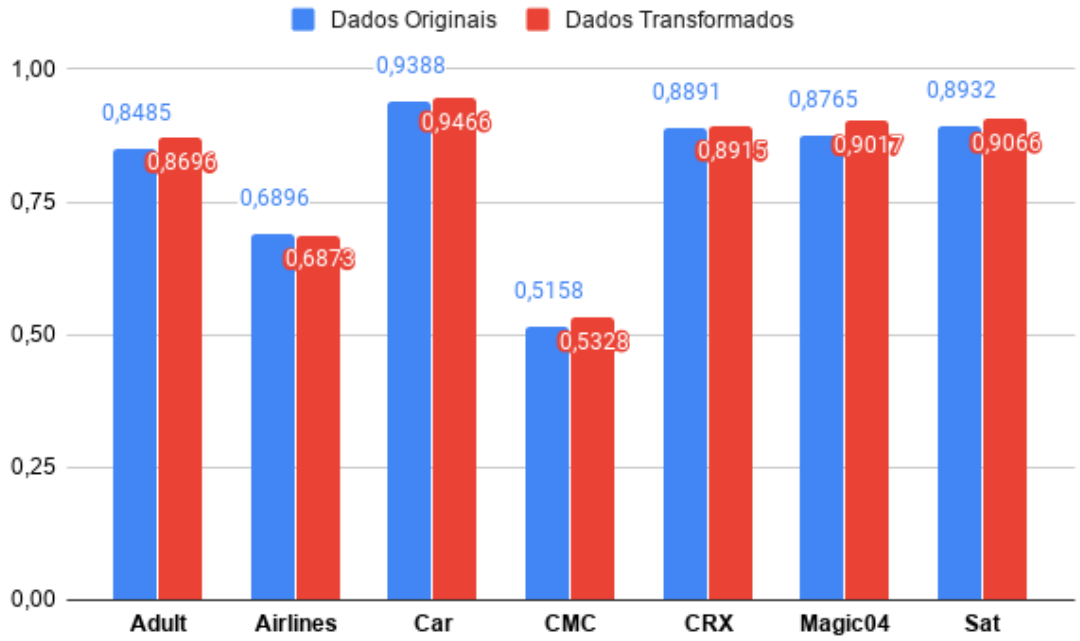


Figura 5.10: Gráfico de barras com as médias de accuracy dos classificadores por conjunto de dados

5.4.2 Resultados Regressão

Nesta subsecção iremos apresentar o impacto que o submódulo de engenharia de atributos teve em problemas de regressão. Da mesma forma que foi feito na subsecção 5.3.2, primeiro irão ser apresentados os erros médios absolutos obtidos para cada algoritmo de regressão e de seguida irão ser apresentados os erros percentuais absolutos médios na forma de um gráfico de barras.

Na Tabela 5.13 estão expostos os erros médios absolutos que cada conjunto de dados obteve antes e depois da execução do submódulo de engenharia de atributos. Apenas um conjunto de dados favoreceu das operações aplicadas pelo submódulo de engenharia de atributos. O resto dos dados obteve resultados muito semelhantes, fazendo aparentar que as operações escolhidas pelo submódulo não resultaram em profundas alterações nos conjuntos de dados. O motivo para o sucedido será abordado na Secção 5.4.3.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	59,15	1512,43	88418,64	0,06	0,08	0,5	2,39
Dados Transformados	32,11	1522,95	88479,34	0,07	0,079	0,37	2,47
Diferença	**-27,04**	10.52	60,7	0,001	**-0,001**	**-0.13**	0,08

Tabela 5.13: Resultados (MAE) obtidos com Support Vector Regressor

Em relação aos resultados com o *Random Forest Regressor*, apresentados na Tabela 5.14, verificamos que apenas 3 conjuntos de dados beneficiaram das operações escolhidas pelo submódulo de engenharia de atributos. Dois destes conjuntos (*Bike Sharing e Water*) obtiveram resultados positivos bastante significativos.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	18,44	68,48	31999,08	0,1	0,07	0,3	2,42
Dados Transformados	18,26	56,85	35239,74	0,09	0,08	0,17	2,39
Diferença	**-0,18**	**-11,63**	3240,66	**-0.01**	0,01	**-0.13**	**-0,03**

Tabela 5.14: Resultados (MAE) obtidos com Random Forest Regressor

Os resultados obtidos com a ferramenta TPOT, expostos na Tabela 5.15, foram igualmente negativos, sendo que apenas as operações aplicadas ao conjunto *Water* resultaram na melhoria de performance. Contudo, grande parte dos resultados negativos são insignificantes mostrando mais uma vez que as operações aplicadas pelo submódulo não resultaram em alterações significativas nos conjuntos de dados.

	Baseball	**Bike Sharing**	**Houses**	**Kin8**	**Space**	**Water**	**Wind**
Dados Originais	16,88	0	31597,01	0,08	0,07	0,33	2,36
Dados Transformados	17,25	27,84	35489,8	0,1	0,08	0,13	2,37
Diferença	0,37	27,84	3892,79	0,02	0,01	**-0,2**	0,01

Tabela 5.15: Resultados (MAE) obtidos com TPOT

No gráfico da Figura 5.11 estão presentes a média dos resultados obtidos por cada conjunto de dados. A métrica utilizada para exprimir os resultados foi o erro percentual absoluto médio.
Dos 7 conjuntos de dados avaliados, 5 conjuntos obtiveram melhores resultados depois de serem aplicadas as operações de engenharia de atributos. Destes 5 conjuntos positivos, 3 (*Baseball, Bike Sharing e Water)* obtiveram resultados positivos bastante significativos. Os restantes conjuntos avaliados, tanto os que tiveram resultados negativos como os que tiveram resultados positivos, não mostraram uma diferença significativa no erro obtido, sugerindo mais uma vez que as operações aplicas não alteraram os dados significativamente.

A excepção poderá ser o conjunto *Houses* cujas alterações significaram num aumento maior do erro. O problema associado com os resultados que não têm uma diferença significativa será abordado na subsecção seguinte.

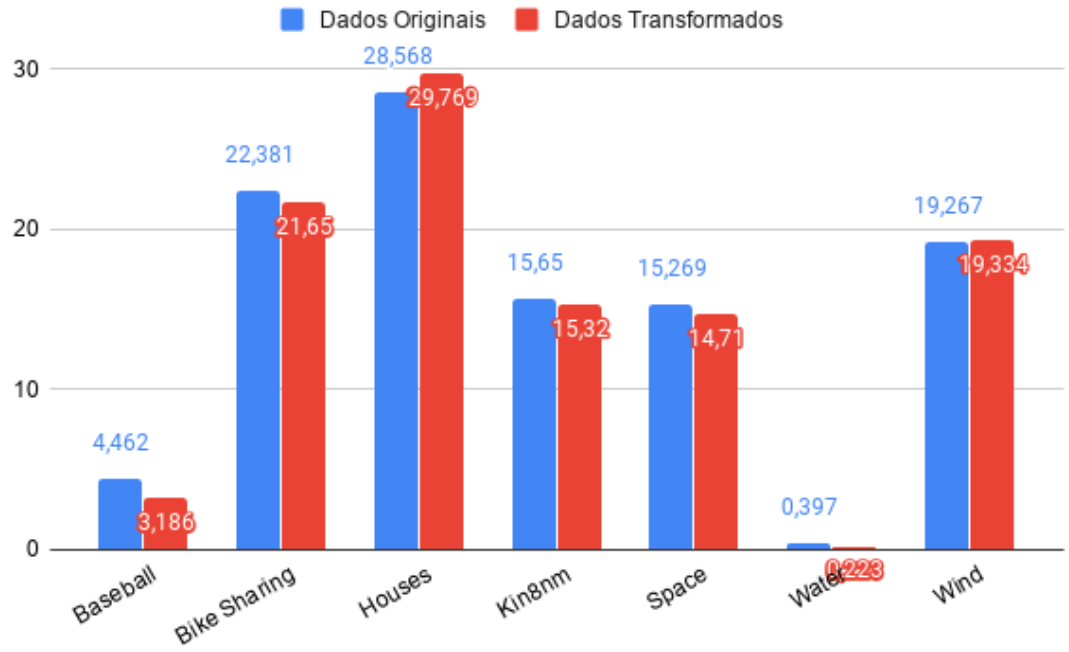


Figura 5.11: Gráfico de barras com as médias de MAPE dos regressores por conjunto de dados

5.4.3 Análise geral do submódulo

Nesta subsecção iremos analisar e perceber que factores influenciaram positivamente e negativamente os resultados obtidos.

Numa visão global, as médias dos algoritmos de aprendizagem computacional para cada conjunto de dados, apontam que as operações aplicadas 11 conjuntos de dados significaram numa melhoria da performance quando comparada com a performance obtida dos dados originais. No geral, os resultados são positivos contudo as diferenças pequenas entre a performance dos dados transformados e a performance dos dados originais, indicam que as operações aplicadas não alteraram significativamente os dados.

Este problema advém principalmente do problema da limpeza de dados, em que é escolhida, na maioria das vezes, a operação *One Hot Encoding* para transformar os atributos categóricos. Esta operação cria diversos atributos binários, ficando assim o conjunto de dados com demasiados atributos com valores 0 ou 1. Esta escolha quase constante de *One Hot Encoding* influenciou negativamente os resultados obtidos no submódulo de engenharia de atributos dado que, as diversas operações deste submódulo não funcionam correctamente com atributos binários, isto é, não alteram profundamente os dados. Especificamente, as operações de geração de atributos não geram atributos muito diferentes dos originais ou até não conseguem gerar atributos de todo. Das 14 operações de geração de atributos implementadas, 12 não conseguem trabalhar correctamente com atributos cujos valores são 1 e 0. Em concreto:

- `Operações K Bins Discretizer`: divide um atributo em intervalos de tamanho k, sendo que o k, no contexto do submódulo, pode ter valor 5, 10 ou 15. No caso de um atributo binário, é impossível dividir o mesmo em intervalos com os valores de k seleccionados, não sendo por isso possível gerar novos atributos com base em atributos binários.

- `Raiz Quadrada`: esta operação é inútil, pois no caso de atributos binários criados pelo *One Hot Encoding*, os atributos gerados vão ser iguais aos atributos originais ($\sqrt{0} = 0$ e $\sqrt{1} = 1$).

- `Logaritmo`: esta operação não chega a ser executada para atributos que contêm valores menor ou igual a zero, que é o caso dos atributos criados pelo *One Hot Encoding*.

- `Operações de Agrupamento`: este tipo de operações é interessante para combinar atributos numéricos e atributos discretos, criando novos atributos discretos. Contudo, se tivermos um conjunto de dados que é composto maioritariamente por atributos discretos, estarmos a criar ainda mais atributos discretos não vai adicionar informação relevante.

- `Operações Polinomiais`: este tipo de operações cria todas as combinações polinomiais entre atributos. É muito útil para atributos contínuos, mas ao ser aplicada apenas a atributos discretos não adiciona informação relevante ao conjunto de dados

Sendo que, grande parte dos conjuntos de dados testados eram compostos maioritariamente por atributos binários, as operações de geração de atributos não conseguiram gerar atributos informativos e diferentes dos originais, existindo operações que nem sequer geraram atributos novos. Consequentemente, os dados transformados eram muito semelhantes aos dados originais, o que resultou numa diferença de performance mínima e insignificante. Este problema resulta do defeito do submódulo de limpeza de dados, e por isso a solução passaria por resolver o defeito no submódulo de limpeza de dados.

A título de exemplo, os conjuntos de dados *CMC* e *Magic04*, beneficiaram das operações escolhidas pelo submódulo de engenharia de atributos, verificando-se uma melhoria na performance obtida. Isto resulta do facto de nestes conjuntos de dados não ter sido aplicada a operação *One Hot Encoding*, permitindo assim que as operações de geração de atributos conseguissem gerar atributos novos e informativos.

Outro problema identificado, diz respeito ao treino dos agentes e é um problema conhecido da abordagem *Deep Q Learning* utilizada. O problema reside no facto de utilizarmos a rede neuronal para prever os valores com que a mesma rede vai ser treinada. Isto é, a cada iteração a fornecemos à rede o estado actual e o target que é composto pelas recompensas de todas as acções. Contudo, em contextos onde existem inúmeras acções, não temos a recompensa real para todos os pares de estado-acção e por isso temos de utilizar a rede para prever essas recompensas e construir o target. Este problema de utilizarmos a rede para prever valores com que a mesma vai ser treinada, gera grande instabilidade nos dados e pode causar *overfitting* da rede neuronal.

Para uma explicação mais clara e simples, apresentamos um exemplo que está exposto na Tabela 5.16. Esta tabela é composta por um possível *target* com que a rede é treinada. Os valores a sublinhado são as recompensas previstas pelas redes, e os valores que não estão sublinhados são as recompensas reais que advêm do agente executar a determinada acção.

Neste contexto, temos cinco acções disponíveis.
Na primeira iteração, o agente escolheu aleatoriamente a acção 1 e executou essa acção. Ao executar a acção recebeu a recompensa real de 1.5. De forma a construirmos o *target* completo para treinarmos a rede neuronal, temos de utilizar a rede para prever os valores das acções dois a cinco dado que não temos recompensas reais para os mesmos. Estas previsões, vão ser aleatórias visto que a rede não sabe nada acerca do contexto. Depois de construirmos o *target* da primeira iteração, fornecemos à rede o estado actual e o *target* construído. O problema, é que o *target* construído que a rede usa para aprender, é composto maioritariamente por valores aleatórios, previstos pela rede, não fornecendo assim uma descrição do contexto real.
Este problema continua ao longo das iterações onde, em cada iteração, temos de usar a rede para prever quatro recompensas de forma a construirmos o *target*. Apesar da rede evoluir ao longo do tempo por ser treinada com mais dados, os dados com que vai ser treinada vão ser muito instáveis. Isto acontece porque a cada iteração estamos a prever os valores para o *target* com redes diferentes. Isto é, na iteração 1 utilizamos uma rede que não sabe nada acerca dos dados para prever os valores do *target* da iteração 1. Depois, na iteração 2, utilizamos a rede que, foi treinada com os valores da iteração 1, para prever os valores do *target* da iteração 2. Este problema continua ao longo do treino, onde em cada iteração utilizamos a rede que foi treinada na iteração anterior para prever os valores do *target*. Ao utilizarmos redes em diferentes estágios de aprendizagem, os *targets* das iterações vão ser diferentes o que cria uma grande instabilidade nos dados de treino.

	Acção 1	**Acção 2**	**Acção 3**	**Acção 4**	**Acção 5**
Iteração 1	1.5	**-33**	**20**	**66**	**-5**
Iteração 2	**1.3**	**23**	4	**-20**	**15**
Iteração 3	**1.5**	**-17**	**3**	-1	**-27**

Tabela 5.16: Exemplo das recompensas (*target*) de várias acções

Este problema com *Deep Q Learning* é conhecido e foi assumido pelos investigadores do artigo original. Existem duas soluções que mitigam esta adversidade: utilizar uma variável para guardar toda a informação ou utilizar uma segunda rede neuronal.

A primeira solução foi implementada neste submódulo contudo não é uma solução perfeitamente eficaz especialmente para os agentes de geração de atributos. Esta solução passa por guardar toda a informação relativa ao estado, acção e recompensa e, ao construir o target procurar se existe a recompensa real para uma determinada acção na informação guardada. Caso exista, utilizamos a recompensa real em vez de utilizar a rede para prever a recompensa da acção. Mais uma vez, vamos recorrer a um exemplo para tornar a explicação mais clara.
A Tabela 5.17 exemplifica o processo de construir o *target* para cada iteração com base em iterações passadas. Tal como no exemplo anterior, as recompensas sublinhadas referem-se aos valores que a rede previu e as recompensas não sublinhadas referem-se às recompensas reais.
Na iteração 2, o agente escolheu a acção 3 e com a sua execução obteve a recompensa real para essa acção. Na hora de construir o *target* relativo a essa iteração, podemos procurar na informação guardada se existem recompensas reais para as acções 1,2,4 e 5. Neste caso, existe a recompensa real para a acção 1 pois esta acção foi executada na iteração 1. Como tal, utilizamos a recompensa real da acção 1 e, para o resto das recompensas temos utilizar a rede neuronal para prever valores. Na iteração 3, o procedimento é igual e podemos usar

a recompensa real das acções 1 e 3.
Com esta solução, diminuímos o número de vezes que temos de utilizar a rede neuronal para prever valores para construir o *target* com que é treinada. Com o evoluir das iterações, iremos ter mais valores de recompensas reais e por isso conseguimos construir o *target* que represente melhor o contexto.

	Acção 1	**Acção 2**	**Acção 3**	**Acção 4**	**Acção 5**
Iteração 1	1.5	**-33**	**20**	**66**	**-5**
Iteração 2	1.5	**23**	4	**-20**	**15**
Iteração 3	1.5	**-17**	4	-1	**-27**

Tabela 5.17: Exemplo do *target* construído com base em informação passada

É importante observar que, no exemplo anterior presumimos que apenas existe um estado, e por isso a recompensa da acção 1 na iteração 2 é igual à recompensa da acção 1 na iteração 1. Algo que não acontece no nosso contexto, onde temos vários estados. A solução é idêntica, mas em vez de simplesmente procurarmos a recompensa de uma acção, temos de procurar a recompensa de uma acção num determinado estado. Esta condição, de procurar por estado, limita esta solução porque o número de estados possíveis é infinito e o número de acções possíveis é também elevado.No caso do número de acções possíveis ser elevado, o número de vezes que temos de recorrer à rede neuronal para construir o *target* é mais elevado. Nos agentes de transformação de atributos e de selecção de atributos, onde temos 4 e 6 acções possíveis, respectivamente, forçámos os agentes a executarem todas as acções para um determinado estado de forma a termos as recompensas reais e assim conseguimos treinar as redes neuronais com *targets* reais. Contudo, nos agentes de geração de atributos, tal não é possível. Nesta situação, existem 14 acções possíveis e por isso é inviável executar todas as acções possíveis para um determinado estado. Assim, os dados com que os agentes de geração de atributos foram treinados são constituídos por recompensas reais e por recompensas previstas pela rede. Estas últimas, levam a que as redes sejam treinadas com recompensas que não representam correctamente o contexto, afectando assim a performance das mesmas.

Resumindo, esta solução implementada, apesar de ser uma melhoria, não soluciona o problema por completo porque existem situações onde temos de recorrer à rede neuronal para prever os dados com que a mesma vai ser treinada.

A segunda solução passa por utilizar duas redes neuronais durante o treino do agente de forma a mitigar o problema da instabilidade nos dados de treino, que acontece devido ao facto de utilizarmos redes diferentes (redes em diferentes fases de treino) para prever os dados de treino. Uma rede, a rede principal, é constantemente actualizada, enquanto a outra rede, chamada *Target Q-Network* é sincronizada com a primeira rede em intervalos regulares. Ambas as redes têm a exactamente a mesma arquitectura. A diferença, é que a *Target Q-Network* é apenas utilizada para estimar as recompensas que precisamos para construir o *target* que vai ser utilizado para treinar a rede principal. A *Target Q-Network* é actualizada ao fim de intervalos regulares, para que possa evoluir ao longo do treino. Desta forma, não utilizamos a rede principal para prever os valores com que a rede neuronal principal vai ser treinada e os valores previstos vão ser mais estáveis porque utilizamos a mesma rede a cada iteração e não uma rede neuronal que está em constante mudança.

Ambas as soluções visam solucionar este problema conhecido de *Deep Q Network*, sendo

que a segunda solução é a mais eficaz nos contextos em que o número de acções é elevado. Contudo, apenas foi implementada a primeira solução. Em iterações futuras do submódulo, seria benéfico implementar a segunda solução e treinar de novo os agentes.

A Tabela 5.18 inclui o número de vezes que cada operação foi seleccionada pelos diferentes agentes em todos os testes, tanto em problemas de classificação como em problemas de regressão. As linhas tracejadas separam os três tipos de operações: transformação de atributos, geração de atributos e selecção de atributos.
Analisando as operações de geração de atributos, verificamos que para problemas de classificação foram mais vezes escolhidas operações que agrupam os dados em classes (*K Bins Discretizer* e operações de agrupamento), e para problemas de regressão foram mais vezes escolhidas operações que utilizam uma operação aritmética com duas variáveis continuas (Divisão, Soma, Logaritmo e *Polynomial Features*). Estas escolhas indicam que as redes são capazes de avaliar correctamente as necessidades de cada tipo de problema e que a abordagem inovadora é viável. Nos problemas de classificação é útil agrupar os dados em classes pois a variável a prever é categórica e nos problemas de regressão é benéfico executar operações aritméticas criando atributos contínuos pois a variável a prever é contínua. Existe alguma aleatoriedade devido ao problema acima mencionado, mas estes resultados revelam que as redes conseguiram aprender correctamente os diferentes tipos de problemas e as suas necessidades, demonstrando assim que *Deep Q Learning* é uma boa abordagem, e que com mais dados de treino dos agentes a aleatoriedade acabaria por diminuir.
Em relação às operações de selecção de atributos, verificamos que na maioria dos testes, o agente decidiu que não deveria ser aplicada nenhuma operação de selecção de atributos. O facto de não seleccionarmos os melhores atributos com recurso a uma operação de selecção de atributos, pode levar a que os resultados sejam piores devido à "maldição da dimensionalidade", especialmente nos dados onde são criados centenas a milhares de atributos devido à fase de geração de atributos. O problema do agente escolher mais vezes não aplicar nenhuma operação de selecção de atributos reside no facto de os dados com que o agente foi treinado terem mais dados e recompensas reais da acção 'sem selecção de atributos' porque para obtermos a recompensa desta acção não necessitávamos de executar nenhuma operação. Com isto, os agentes de selecção de atributos atribuíram mais peso a esta acção levando a que esta acção seja escolhida mais vezes. Este problema seria solucionado com um conjunto de dados de treino mais balanceado.

Outro problema notado ao analisar as operações escolhidas pelos agentes, é que frequentemente foram seleccionadas as mesmas operações de geração de atributos duas ou três vezes seguidas. Ou seja, a fase de geração de atributos é composta por três execuções de operações seguidas, e o que se verificou é que as três operações escolhidas foram muitas vezes repetidas. Isto aconteceu em consequência do problema do *One Hot Encoding*, enunciado anteriormente. Por exemplo, para a primeira operação, o agente escolhia o Logaritmo com base nas características dos dados. A aplicação desta operação não iria resultar em profundas alterações porque a maioria dos atributos era composto por zeros e uns. Como não alterava os dados, as características dos dados que o agente utiliza para prever as operações, acabavam por ser muito semelhantes às características dos dados antes da primeira operação levando a que o agente escolhesse novamente a operação Logaritmo. Esta operação, novamente escolhida, não iria surtir em grandes alterações nos dados. Este problema de escolher repetidamente as mesmas operações, levou a que os resultados dos dados originais e os resultados dos dados transformados fossem muito semelhantes pois as operações aplicadas aos dados originais não resultaram em grandes transformações nos dados transformados.

Operação	**Resultados de Classificação**	**Resultados de Regressão**	**Soma**
Standard Scaler	2	8	10
Normalizer	4	4	8
Min Max Scaler	5	1	6
Robust Scaler	3	1	4
Soma	0	5	5
Diferença	2	11	13
Divisão	6	3	9
Raiz Quadrada	10	4	14
Logaritmo	5	6	11
5 Bins Discretizer	0	1	1
10 Bins Discretizer	1	2	3
15 Bins Discretizer	5	1	6
Agrupar por MAX	2	0	2
Agrupar por MIN	6	0	6
Agrupar por Média	0	1	1
Agrupar por Conta	3	3	6
Polynomial Features No Interaction	0	1	1
Polynomial Features Interaction Only	2	4	6
Sem Selecção de Atributos	8	11	17
PCA	0	0	0
Kernel PCA	2	1	3
Truncated SVD	1	0	1
Fast ICA	0	0	0
Select K Best	1	2	3
RFE	2	0	2

Tabela 5.18: Operações escolhidas pelos agentes de classificação e regressão

O facto dos resultados de problemas de classificação serem mais positivos que os resultados de problemas em regressão revela que a abordagem escolhida é potencialmente eficaz e que se for treinada com mais dados é possível obter resultados melhores. Isto é, os agentes de classificação, que obtiveram melhores resultados, foram treinados com 60 conjuntos de dados enquanto os agentes de regressão foram treinados com 42 conjuntos de dados. Esta diferença significativa, levou a que os agentes de classificação fossem mais capazes de prever operações. Sugere então, que um treino mais robusto e com mais conjuntos de dados leva a que os resultados sejam melhores.

Resumindo, existiram dois grandes problemas associados com o submódulo e com os resultados obtidos. O primeiro problema analisado diz respeito ao facto da maior parte dos conjuntos de dados ser composto por atributos binários, devido à operação de *One Hot Encoding* aplicada na limpeza de dados. E o segundo problema examinado é referente a uma falha na técnica de *Deep Q Learning*. Ambos os problemas têm soluções, sendo que no primeiro a solução passa por solucionar o problema referente ao submódulo de limpeza de dados e, no segundo a solução passa por adicionar uma segunda rede neuronal no treino dos agentes e treinar com mais dados.

Passando agora para a análise dos tempos de execução, apresentamos na Tabela 5.19 os tempos, em segundos, que o submódulo demorou a seleccionar e a aplicar as operações aos dados. Estes tempos incluem, o passo de transformação de atributos, o passo de geração de atributos e o passo de selecção de atributos. Em média, o tempo de execução submódulo foi cerca de 818.38 segundos. Significa que o módulo demora cerca de 14 minutos a seleccionar e a aplicar um conjunto de operações que, potencialmente, aumenta a performance obtida. A parte onde o agente selecciona a melhor operação é relativamente rápida pois é apenas necessário utilizar a rede para prever valores. Por outro lado, a parte onde o agente executa as operações demora mais tempo pois muitas operações trabalham ao nível da amostra e exigem cálculos complexos.
Constatando que um cientista de dados pode passar dias a analisar os dados e a escolher as melhores operações de engenharia de atributos, consideramos que este submódulo, ao demorar apenas 14 minutos, permite reduzir significativamente o tempo despendido nesta fase.

Dados	**Tempo Teste 1 (s)**	**Tempo Teste 2 (s)**	**Média (s)**	**Nº de Observações**
Adult	1749.11	338.45	1043.78	48842
Airlines	3108.12	866.22	1987.17	30000
Car	19.71	18.32	19.015	1728
CMC	93.98	90.05	92.015	1473
CRX	46.85	46.85	46.85	690
Magic04	400.28	400.28	400.28	19020
Sat	1212.86	4966.91	3089.88	4435
Baseball	117.05	286.96	202.0	1232
Bike Sharing	269.49	167.62	218.55	731
Houses	280.432	437.47	358.95	20640
Kin8	4032.78	878.17	2455.47	8191
Space	750.06	616.68	683.36	3107
Water	123.49	101.9	112.695	527
Wind	828.52	666.09	747.31	6574
Média	930.909	705.85	818.38	10513.57

Tabela 5.19: Tempos de execução dos testes do submódulo de engenharia de atributos

Em suma, o submódulo implementado cumpre os requisitos, no sentido em que consegue decidir quais as operações que devem ser aplicadas e desempenha esta tarefa em pouco tempo. Evidentemente, há espaço para melhoria, nomeadamente em relação aos problemas apontados anteriormente.
Apesar dos problemas, esta abordagem nova, demonstrou ser viável e capaz de obter bons resultados. Reparámos que, os agentes de classificação, que foram treinados com mais dados, alcançaram melhores resultados que os agentes de regressão. Este facto, indica que os agentes conseguiram aprender e identificar as relações entre as características dos dados e as operações, e que se forem treinados com mais dados conseguem aprender melhor e obter resultados melhores. Solucionando os problemas acima mencionados e treinando os agentes com mais dados, e ficamos com um submódulo ainda mais robusto. Não obstante, o submódulo já implementado cumpre os requisitos e é uma mais valia para o trabalho de um cientista de dados ou de qualquer outro utilizador.

5.5 Funcionalidades Assistente Virtual

Nesta secção iremos apresentar as conversas tidas com o Assistente Virtual implementado. De relembrar, que este agente foi desenvolvido para conseguir fazer face às seguintes intenções: executar o passo de limpeza de dados, executar uma operação especifica de limpeza de dados, executar o passo de engenharia de atributos, executar uma operação específica de engenharia de atributos, executar o passo de selecção do modelo, executar um modelo em específico e executar os três passos em conjunto. Assim sendo, as figuras seguintes dizem respeito à execução de cada uma destas intenções. Em todas as figuras, o texto a azul representa texto relativo ao assistente, ou seja o que ele disse. E o texto a roxo representa o texto escrito pelo utilizador.

Antes do assistente saber qual é intenção do utilizador, tem de saber três componentes importantes: qual o conjunto de dados a utilizar, o nome do atributo a prever e o tipo de problema. Esta pergunta é sempre feita pelo assistente no início da execução do mesmo, tal como se vê na Figura 5.12. Se por algum motivo, o assistente perder esta informação, irá perguntar de novo sempre que o utilizador digitar uma intenção, tal como se vê na Figura 5.13.

```
Where is the data file located?
Your input ->  test/data/Binary Classification/pima.csv
Which is the name of the target column?
Your input ->  Target
What is the problem type? (classification or regression)
Your input ->  classification
```

Figura 5.12: Primeiras perguntas realizadas pelo assistente

Execução do pipeline inteiro

O pipeline inteiro corresponde a executar os três passos sequencialmente - limpeza de dados, engenharia de atributos e selecção do modelo. Tal pedido pode ser visto na Figura 5.13. No fim da execução do módulo AutoML, o assistente coloca toda a informação legível e mostra ao utilizador. Esta informação é relativa à performance do modelo, às operações de limpeza de dados e engenharia de atributos que foram aplicadas e, em que local foram guardados todos os ficheiros gerados (dados limpos, dados passados por engenharia de atributos e código do modelo).

```
Your input ->  Execute every step
Going to execute full pipeline
Where is the data file located?
Your input ->  test/data/Binary Classification/pima.csv
Which is the name of the target column?
Your input ->  Target
What is the problem type? (classification or regression)
Your input ->  classification
I just finished running the pipeline.
        Best model score: 0.7818181818181819
        These operations were performed in in order to clean the data:
                Handle Missing data:  none
                Handle Outliers: local outlier factor
                Handle duplicated: dont remove
                Inconsistent data: to lower case
                Handle dates: dont handle dates
        Data was saved in test/data/Binary Classification/1591637423.6028647_data_after_data_cleaning.csv
        These feature engineering operations were performed:
                Feature Transformation: min max scaler
                Feature Generation: group by then max
                Feature Selection: select k best
        Data was saved in test/data/Binary Classification/1591637441.8580766_data_after_feature_engineering.csv
        Code with the selected model was saved in: test/data/Binary Classification//1591637504.634316_model.py
```

Figura 5.13: Execução do pipeline inteiro

Execução do passo de limpeza de dados
Em relação à limpeza de dados, podemos executar o pipeline inteiro, tal como está representado na Figura 5.14, ou podemos executar uma operação em específico, representado na Figura 5.15.

Ao pedirmos para executar o pipeline inteiro, o módulo AutoML irá testar todas as combinações de operações e depois retornar a melhor configuração. Essa configuração é depois mostrada ao utilizador. Esta interacção pode ser vista na Figura 5.14.

```
Your input ->  Can you clean my data?
Going to execute data cleaning pipeline
Do you want to select the operations that will be used in the pipeline? If no, the default pipeline will be used
Your input ->  No, you can run the default pipeline
I just finished running data cleaning pipeline.
        Data was saved in test/data/Binary Classification/1591689495.6191344_data_after_data_cleaning.csv
        These operations were performed in in order to clean the data:
                Handle Missing data:  none
                Handle Outliers: local outlier factor
                Handle duplicated: dont remove
                Inconsistent data: to lower case
                Handle dates: dont handle dates
                Categorical Transformation: label encoding
```

Figura 5.14: Execução do passo de limpeza de dados

No cenário de executar uma operação em específico, o assistente pergunta que tipo de método é para executar, e depois, com base no tipo de método, pergunta qual é a operação a executar. Na exemplo da figura seguinte, o utilizador escolheu executar uma operação relacionada com *outliers* (o tipo), mais em concreto a operação *local outlier factor*.

```
Your input ->  I want to run a data cleaning operation
Going to execute data cleaning method. Please choose the method
Which method type would you like? (Missing values, Outliers, Duplicated values, Inconsistent data or dates)
Your input ->  Outliers
Which method would you like?
        - Z score
        - Local outlier factor
Your input ->  Local Outlier factor
I just finished running method: local outlier factor.
        Data was saved in test/data/Binary Classification/1591689839.1118212_data_after_data_cleaning.csv
        These operations were performed in in order to clean the data:
                Handle Missing data:  none
                Handle Outliers: local outlier factor
                Handle duplicated: dont remove
                Inconsistent data: none
                Handle dates: dont handle dates
```

Figura 5.15: Execução de uma operação específica de limpeza de dados

Execução do passo de engenharia de atributos
Relativamente ao passo de engenharia de atributos, podemos novamente executar o pipeline inteiro, tal como está representado na Figura 5.16, ou podemos executar uma operação em específico, representado na Figura 5.17.

No primeiro caso, temos a oportunidade de decidir que tipo de algoritmos entram para a configuração do pipeline. Isto é, podemos decidir se queremos que operações relacionadas com transformação de atributos, geração de atributos ou selecção de atributos entrem para a configuração. No cenário da Figura 5.16, foi expresso a utilização de todos os tipos de algoritmos.

```
Your input ->  I want to maximize the potential of my data. Can you run the feature engineering pipeline
Going to execute feature engineering pipeline
Do you want to include feature transformation in the pipeline?
Your input ->  Yes
Do you want to include feature generation in the pipeline?
Your input ->  Sure
Do you want to include feature selection in the pipeline?
Your input ->  Yes, go ahead
I just finished running feature engineering pipeline.
        Data was saved in test/data/Binary Classification/1591689705.129597_data_after_feature_engineering.csv
        These operations were performed:
                Feature Transformation: min max scaler
                Feature Generation: group by then max
                Feature Selection: select k best
```

Figura 5.16: Execução do passo de engenharia de atributos

No contexto de executar uma operação em específico, podemos directamente dizer que tipo de operação (transformação de atributos, geração de atributos ou selecção de atributos) queremos executar ou podemos apenas dizer que queremos executar uma operação de engenharia de atributos e depois o assistente irá perguntar qual é a operação. O primeiro caso, está representado na Figura 5.17, em que o utilizador diz que quer executar uma operação de transformação de atributos e o assistente pergunta qual é a operação. O segundo caso, é parecido ao que se passa na Figura 5.15 mas com foco nas operações de engenharia de atributos.

```
Your input ->  Can you execute a Feature Transformation method?
Going to execute feature engineering method
Which feature transformation method do you like to execute?
        - Standard Scaler
        - Min Max Scaler
        - Normalizer
        - Robust Scaler
Your input ->  Normalizer
Successfuly executed Normalizer
 File was saved in test/data/Binary Classification/1591689945.7350578_data_after_NORMALIZER.csv
```

Figura 5.17: Execução de uma operação específica de engenharia de atributos

Execução do passo de selecção do modelo

Por fim, existe também a possibilidade de executar o passo de selecção do modelo, representado na Figura 5.18 e de executar um modelo em específico, retratado na Figura 5.19. Em ambos os casos, o assistente pergunta se queremos executar o pipeline normal, isto é, executar a ferramenta *TPOT*, ou se queremos executar uma operação em específico.

No primeiro cenário, dizemos ao assistente que queremos executar o pipeline normal. O assistente irá então fazer o pedido HTTP correspondente, e no fim mostra a informação retornada. Esta informação é relativa à performance obtida assim como a localização do ficheiro *python* guardado.

```
Your input ->  What is the best model?
Going to execute model selection with TPOT
Do you want to execute a specific machine learning algorithm? If no, all machine learning algorithms will enter tpot's pipeline
Your input ->  No, you can use the default
I just finished running the pipeline.
        Best model score: 0.7758620689655172
        Code with the selected model was saved in: test/data/Binary Classification//1591690140.7674663_model.py
```

Figura 5.18: Execução do passo de selecção do modelo

No segundo cenário, dizemos ao assistente que queremos executar uma operação em específico e isto fará com que o assistente pergunte qual é o algoritmo de aprendizagem computacional que queremos executar. Se o tipo de problema, introduzido anteriormente pelo utilizador, for de classificação, o assistente irá mostrar algoritmos de classificação. Por outro lado, se o tipo de problema for regressão, o assistente irá mostrar os algoritmos de regressão. Depois de executar a função do módulo AutoML, o assistente mostra a informação retornada.

```
Your input ->  I want to select a algorithm
Going to execute model selection with TPOT
Do you want to execute a specific machine learning algorithm? If no, all machine learning algorithms will enter tpot's pipeline
Your input ->  Yes
Which model do you want to execute?
 Classification options:
        -Gaussian NB
        -Bernoulli NB
        -Mutinomial NB
        -Decision Tree Classifier
        -Extra Trees Classifier
        -Random Forest Classifier
        -Grandient Boosting Classifier
        -K Neighbors Classifier
        -Linear SVC
        -Logistic Regression
        -XGB Classifier
        -SGD Classifier
Your input ->  Run Decision Tree Classifier
I just finished running the pipeline.
        Best model score: 0.75
        Code with the selected model was saved in: test/data/Binary Classification//1591690442.4195685_model.py
```

Figura 5.19: Execução de um algoritmo de aprendizagem em específico

Apesar do assistente virtual desenvolvido ser apenas uma prova de conceito, consegue fazer face a todas as necessidades desejadas e releva que um assistente deste tipo é extremamente útil. Permite a execução de diferentes operações relativas a Aprendizagem Computacional com recurso a comunicação em linguagem natural e sem a necessidade de implementar qualquer código. Este assistente é bastante robusto, conseguindo responder às necessidades para as quais foi implementado. É possível executar os diferentes cenários utilizando uma variedade de frases, tornando-o assim capaz de responder às mais variadas perguntas que possam vir do utilizador. Sempre que o assistente não percebe uma mensagem, ou essa mensagem não faz parte das funcionalidades que o assistente consegue executar, o mesmo informa o utilizador que não está preparado para realizar tal pedido.

O assistente desenvolvido poderá ser utilizado em diversos contextos. Desde centros de investigação a museus científicos. No primeiro caso, existem diversos investigadores que não sabem programar e por isso não podem beneficiar das capacidades de Aprendizagem Computacional de forma a tirarem o maior proveito da enorme quantidade de dados que possuem. Com o assistente virtual, não precisam de implementar qualquer código, podendo tirar beneficio de Aprendizagem Computacional com recurso a comunicação em linguagem natural. O segundo exemplo, é relativo a museus científicos e pedagógicos onde crianças aprendem diversos temas. Num museu relacionado com informática, poderá existir um computador equipado com o assistente virtual e as crianças poderão interagir com o assistente virtual e executarem operações de Aprendizagem Computacional, aprendendo assim um pouco destas duas áreas.

A grande vantagem deste assistente é que derruba duas barreiras no que toca à implementação de projectos de Aprendizagem Computacional. A primeira barreira é o não saber programar e com isso não conseguir implementar código com métodos de Aprendizagem

Computacional. E a segunda, é não ter experiência nenhuma em Aprendizagem Computacional. No primeiro contexto, a utilização de linguagem natural substitui a necessidade de saber programar. E no segundo contexto, a utilização de um módulo de AutoML substitui a necessidade do utilizador saber que operações devem ser aplicadas aos dados de forma a tirar proveito dos mesmos.

Capítulo 6

Metodologia e Planeamento

Este capítulo oferece uma visão geral acerca do trabalho realizado ao longo do estágio. Na Secção 6.1 será apresentada a metodologia de desenvolvimento adoptada neste trabalho. E, na Secção 6.2 será apresentado o não só o trabalho planeado, como também o trabalho feito em ambos os semestres.

6.1 Metodologia de Desenvolvimento

A equipa na qual o estudante foi inserido segue a metodologia de desenvolvimento *Scrum*[21]. Neste sentido, *Scrum* foi utilizado para o desenvolvimento das ferramentas apresentadas neste relatório.

Scrum é uma ferramenta que fornece um conjunto de processos e técnicas para o desenvolvimento incremental de um produto. Compreende um conjunto de cargos, artefactos, regras e eventos que pretendem facilitar o processo de implementar um produto.
Foca-se numa abordagem iterativa, incremental e adaptativa de forma a simplificar a adaptação a problemas que possam surgir ou a mudanças na decisão do cliente. Permite assim a rápida adaptação à mudança, algo essencial nos projectos de hoje em dia.

Relativamente aos cargos, estes existem para que as tarefas sejam divididas e cada pessoa da equipa possa realizar o seu trabalho sem depender de outros. Os cargos são: proprietário do produto, equipa de desenvolvimento e *Scrum Master*.
O proprietário do produto é responsável por verificar que tudo está a correr bem dentro da equipa e que o produto tem, incrementalmente, mais valor. É encarregado pela ligação entre os clientes e a equipa de desenvolvimento. Deste modo, percebe o produto que é pedido e como tal gere o *Product Backlog* que será introduzido mais à frente.
A equipa de desenvolvimento realiza tarefas que vão, incrementalmente, acrescentar valor ao produto. É responsável pelo desenvolvimento do produto. A cada *Sprint* são definidas as tarefas para cada pessoa da equipa e estas têm a duração da *Sprint* para as realizar.
Por fim, o *Scrum Master* é responsável por promover as melhores práticas e princípios do *Scrum*.

Neste projecto em particular, o proprietário do produto foi Tiago Baptista (orientador da CSW), o *Scrum master* foi o Rui Lopes (funcionário da CSW) e a equipa de desenvolvi-

mento foi apenas o Filipe Good.

Em relação aos eventos, existem um conjunto de eventos que fornecem práticas para que seja possível reagir rapidamente a uma possível mudança ou problema. Desta forma, os eventos permitem que as equipas inspeccionem uma determinada mudança e consigam-se adaptar.
Os eventos mais importantes, ou pelo menos os que acrescentam directamente valor ao produto, são as *Sprints*. As *Sprints* são fases de desenvolvimento que, incrementalmente, adicionam valor ao produto. No caso do projecto, as *Sprints* duraram duas semanas e incluíram os seguintes eventos:

- `Planeamento da Sprint:` neste evento, defini-se as tarefas que vão ser realizadas numa determinada *Sprint*. Com base nas tarefas presentes no *Product Backlog*, são escolhidas um número de tarefas que têm de ser realizadas até ao fim da *Sprint*;
- `Revisão da Sprint:` no fim das duas semanas, existe uma reunião para analisar o que foi feito durante a *Sprint*. Esta reunião também serve para averiguar se algum membro da equipa teve um determinado problema. Nesta reunião o *Product Backlog* pode ser actualizado com base em erros encontrados ou mudanças na visão do produto;

Por fim, existem artefactos. Estes permitem que toda a equipa tenha acesso a informação essencial acerca do projecto. Os artefactos utilizados foram os seguintes: *Product Backlog*, *Sprint Backlog* e *Increment*.

O *Product Backlog* contém toda a informação de tarefas que precisam de ser realizadas para a conclusão do desenvolvimento do produto. É composto por uma lista ordenada de tarefas baseadas nos requisitos e na visão do cliente para o produto. Este artefacto é dinâmico. Devido à capacidade de adaptação, o *Product Backlog* está em constante mudança de forma a satisfazer as necessidades do cliente. O *Product Backlog* é utilizado para descrever o trabalho futuro.

O *Sprint Backlog* é um conjunto de tarefas seleccionadas a partir do *Product Backlog* que estão planeadas para serem realizadas ao longo da *Sprint*. No início de cada *Sprint* é feito o planeamento da *Sprint*. Este planeamento permite identificar o que deve ser realizado ao longo da *Sprint*.

O *Increment* é a soma de todas as tarefas completadas. No fim de cada *Sprint*, são revisitadas as tarefas que foram planeadas. As completadas são acrescentadas ao *Increment*.

6.2 Planeamento

Neste capítulo, são apresentados e descritos os planos de trabalho para o projecto. Na Secção 6.2.1 é apresentado o trabalho realizado no primeiro semestre com recurso a uma breve descrição e a um Diagrama de *Gantt*. Na Secção 6.2.2, é comparado o trabalho que estava planeado para o segundo semestre com o trabalho realizado no mesmo semestre.

6.2.1 Primeiro Semestre

Na Figura 6.1 está o Diagrama de *Gantt* para o primeiro semestre que descreve as principais tarefas realizadas. Este semestre iniciou-se com uma forte componente de aprendizagem visto que o estudante não tinha tido contacto com os dois conceitos que foram ser abordados ao longo do estágio. Seguiu-se uma análise do estado da arte para averiguar e avaliar o trabalho realizado na área de AutoML. De seguida, o trabalho realizado focou-se na especificação projecto a implementar. Primeiro foi ponderado que fases seriam úteis e interessantes de implementar de forma automatizada e depois, foram pensadas as abordagens que iriam automatizar cada uma das fases. A seguir, foram executados testes de forma a comparar duas ferramentas que automatizam a fase de selecção do modelo e, foi escolhida uma dessas ferramentas para integrar no módulo. Por fim, deu-se a escrita do relatório intermédio.

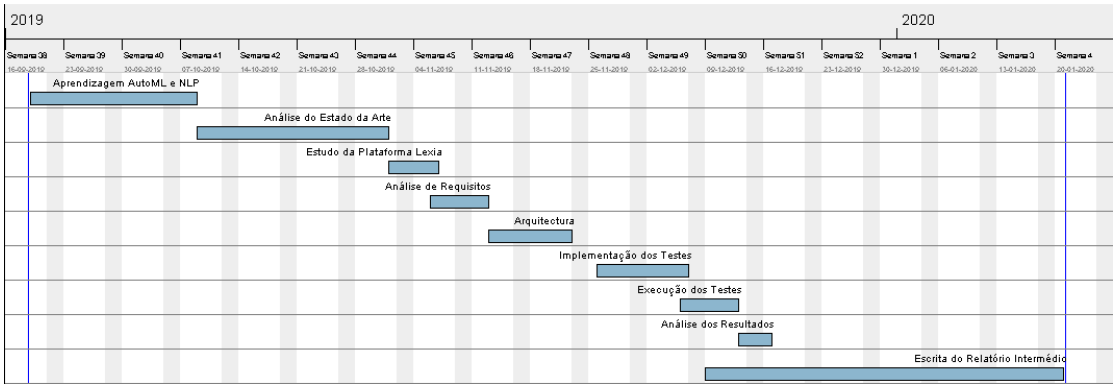


Figura 6.1: Diagrama de *Gantt* para o primeiro semestre

6.2.2 Segundo Semestre

Nesta secção, é detalhado o trabalho realizado ao longo do segundo semestre e, o mesmo é comparado com o trabalho que estava planeado.

Trabalho esperado

A Figura 6.2 mostra o plano de trabalho que deveria ser seguido no segundo semestre. Este plano, que foi realizado em Dezembro de 2019, serviu como suporte para o trabalho efectivamente realizado ao longo do semestre. Previa a implementação incremental do módulo AutoML, seguida de testes e validação. No fim, seria escrito o relatório final.

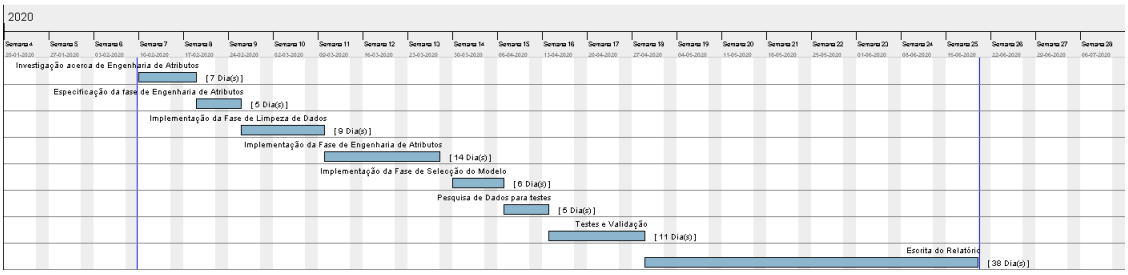


Figura 6.2: Diagrama de *Gantt* planeado para o segundo semestre

Trabalho realizado

Na Figura 6.3 é apresentado o trabalho efectivamente realizado ao longo do semestre. Em termos de tarefas realizadas, não difere muito com o plano apresentado anteriormente, contudo, o tempo que levou a executar certas tarefas difere bastante do tempo planeado. De seguida, iremos primeiro detalhar o que foi realizado em cada tarefa e explicar o desfasamento do tempo previsto para o tempo real.

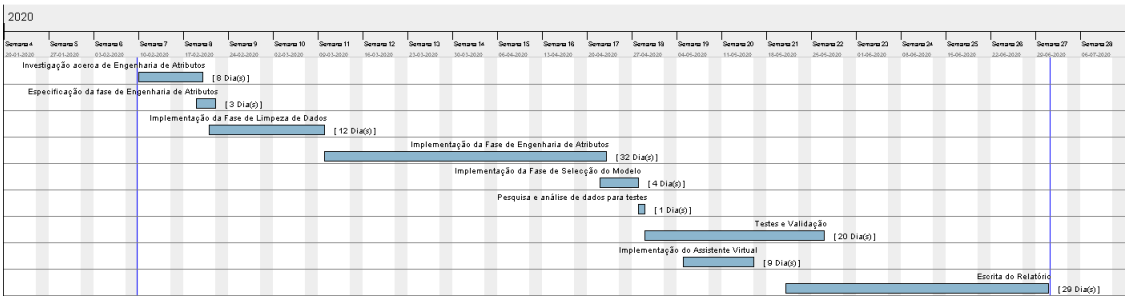


Figura 6.3: Diagrama de *Gantt* do segundo semestre

Foram realizadas as seguintes tarefas:

- `Investigação acerca de Engenharia de Atributos:` a primeira tarefa realizada no segundo semestre teve em vista a análise das abordagens apresentadas no semestre anterior. Esta análise serviu para avaliar qual seria a melhor abordagem a seguir para automatizar o submódulo de engenharia de atributos. Foram exploradas as técnicas de *Meta-Learning*, Algoritmos Evolucionários e RL. Para cada técnica, foi estudado como é que a automatização da fase de engenharia de atributos podia ser feita com a mesma. Depois de avaliar todas as desvantagens e vantagens de cada técnica, decidiu-se utilizar RL.

- `Especificação da fase de Engenharia de Atributos:` com a técnica já decidida, foram especificados todos os detalhes da implementação da técnica no contexto do submódulo de engenharia de atributos.

- `Implementação da fase de Limpeza de Dados:` a implementação módulo AutoML foi dividida em três tarefas grandes relacionadas com os submódulos do módulo AutoML. A primeira tarefa de implementação foi relativa à implementação do submódulo de limpeza de dados. Nesta tarefa, primeiro foram implementadas todas as operações de limpeza de dados previstas e de seguida foi implementado o algoritmo que escolhe a conjunto de operações mais eficaz.

- `Implementação da fase de Engenharia de Atributos:` nesta fase do projecto, focámos na implementação do submódulo de engenharia de atributos. Primeiro foram implementadas todas as operações previstas para este submódulo, ou seja, as operações de transformação de atributos, de geração de atributos e de selecção de atributos. De seguida, foi implementado o algoritmo de *Deep Q Learning* personalizado para o treino de agentes relativos ao submódulo de engenharia de atributos. E, por fim, os agentes foram treinados.

- `Implementação da Fase de Selecção do Modelo:` esta tarefa foi relativa à implementação do submódulo de selecção do modelo. Esta tarefa foi mais simples pois apenas foi necessário integrar a ferramenta *TPOT* no módulo AutoML.

- `Pesquisa e análise de dados para testes:` com o módulo já implementado, deu-se a tarefa em que pesquisámos e analisámos os dados que seriam utilizados para os testes. A componente da análise foi muito importante pois queríamos seleccionar dados bastante sujos que fossem bons para testar o módulo.

- `Testes e Validação:` nesta tarefa foram implementados vários *scripts* que testaram o módulo em várias vertentes.

- `Implementação do Assistente Virtual:` esta tarefa não estava planeada, contudo, dadas as adversidades descritas anteriormente, foi realizada enquanto os testes estavam a executar. Primeiro foi implementado o serviço *REST* e de seguida foi implementado e treinado o assistente virtual.

- `Escrita do relatório:` a realização desta tarefa produziu o presente relatório.

Comparando o trabalho planeado com o trabalho realizado é possível constatar que existem duas grandes diferenças: foi realizada mais uma tarefa e o esforço previsto para certas tarefas foi subestimado.
Em relação à primeira diferença, a adição da tarefa de implementação do assistente virtual foi necessária em virtude dos contratempos verificados. No primeiro semestre estava planeado integrar o módulo AutoML no Lexia, que é um uma ferramenta de assistente virtual que está a ser desenvolvido pela Critical Software, contudo, tal não foi possível e por isso foi necessário alocar tempo para implementar um assistente virtual. Este assistente foi implementado quando os testes estavam a ser executados e, por isso, a sua implementação não causou grandes danos temporais no projecto inteiro.
A segunda diferença diz respeito ao facto do tempo utilizado para certas tarefas ter sido mal estimado. As principais discrepância em relação ao tempo, verificam-se na tarefa de implementação do submódulo de engenharia de atributos e na tarefa de testar e avaliar o módulo AutoML. Em relação à tarefa de implementação do submódulo de engenharia de atributos, a grande alocação de tempo deveu-se ao facto da abordagem escolhida ter tido uma grande curva de aprendizagem. Esta abordagem, com a qual o aluno nunca tinha tido contacto, revelou-se ser um pouco mais complexa do esperado e por isso grande parte do tempo alocado na tarefa em questão esteve relacionado com aprendizagem em conjunto com um processo de tentativa erro. Notar ainda que, o início desta tarefa coincidiu exactamente com a altura em que o aluno começou a trabalhar a partir casa, em consequência da pandemia actual, o que teve influências negativas não só nesta tarefa mas também no estágio em geral. Em relação à tarefa de testar e validar o módulo implementado, esta acabou por demorar o dobro do esperado porque foi necessário executar os testes duas vezes. Na primeira tentativa de execução dos testes, as métricas foram mal calculadas e por isso foi necessário executar uma segunda vez. O problema com o cálculo das métricas esteve relacionado com a falta de experiência do aluno assim como a dificuldade de comunicação,

imposta pelo trabalho remoto, com o orientador da Critical Software.

Numa análise geral, apesar das diferenças de tempo entre o trabalho planeado e o trabalho realizado, todo o trabalho que estava planeado foi correctamente executado resultando na implementação do módulo. Foi ainda implementado um assistente virtual que não estava planeado, resultando assim num trabalho ainda mais composto. Dadas todas as adversidades, quer tenham sido impostas pela pandemia *Covid-19*, quer tenham sido impostas por contratempos relacionados com o desenvolvimento do Lexia, o aluno foi capaz de se adaptar e efectuar não só o trabalho planeado, como também trabalho adicional.

Capítulo 7

Conclusão

O crescimento notável da área de Aprendizagem Computacional conduziu ao desenvolvimento de inúmeros sistemas dotados de inteligência e ao aumento incontestável da importância que esta área tem no nosso dia-a-dia. A investigação relativa a este tópico, tem demonstrado a capacidade de ultrapassar várias barreiras no que toca à aprendizagem por parte dos algoritmos. Contudo, um obstáculo constante é a complexidade associada a todos os algoritmos produzidos pela investigação, limitando assim o desenvolvimento de um projecto de Aprendizagem Computacional a profissionais experientes. Com o propósito de reduzir a barreira da complexidade surgiu o conceito de AutoML.

Neste relatório foi apresentado o trabalho desenvolvido que explorou o conceito de AutoML e as suas competências actuais, com o objectivo de implementar um módulo de AutoML e de utilizar as capacidades do mesmo num novo cenário. Em específico, foi implementado um módulo composto por três submódulos que automatizam três fases importantes e, foi desenvolvido um assistente virtual que permite a execução destas fases com recurso a linguagem natural.

No contexto do módulo, foi decidido implementar as três fases que, correspondem à maior parte do trabalho associado com projectos de Aprendizagem Computacional, uma vez que grande parte das ferramentas existentes não automatizam estas três fases em conjunto. Das três fases implementadas, a fase da limpeza de dados e a fase de engenharia de atributos foram as maiores contribuições, pelo facto de terem sido implementadas abordagens novas que automatizam estas fases.

No que toca à limpeza de dados, a inovação reflectiu-se numa nova função de avaliação, diferenciando-se assim das funções de avaliação utilizadas nas investigações descritas no Capítulo 2. A função de avaliação implementada avalia a importância de cada atributo em vez de avaliar a performance obtida por meio de algoritmos de aprendizagem. Esta última, que é a abordagem mais utilizada, peca no facto de uma limpeza eficaz não ter de resultar necessariamente numa melhoria da performance. A abordagem proposta não cai neste erro ganancioso porque se preocupa com a informação que cada atributo tem acerca da variável a prever, deixando a tentativa de melhoria da performance para a fase de engenharia de atributos. Relativamente aos resultados, notámos que é difícil avaliar correctamente uma ferramenta de limpeza de dados. Apesar dos problemas apresentados no Capítulo 5, consideramos que o submódulo implementado cumpre os requisitos dado que é capaz de se adequar a cada conjunto de dados, escolhendo as melhores combinações

de operações para cada conjunto de dados em específico.

Em relação à fase de engenharia de atributos, a abordagem implementada propõe imitar a forma como o cientista de dados aprende a seleccionar as operações. Neste caso, a experiência obtida ao realizar a tarefa centenas de vezes leva a que as escolhas de operações feitas pelo cientista de dados sejam cada vez mais satisfatórias. Desta maneira, recorremos a técnicas de RL para treinar agentes que aprendem e aperfeiçoam as suas escolhas com base na experiência. Esta abordagem mostrou ser viável, uma vez que, através dos resultados apresentados constatámos que os agentes treinados conseguiram aprender as relações entre as características dos dados, as operações e as respectivas recompensas. Através das diferenças de resultados dos agentes de classificação e dos agentes de regressão, observámos que um treino mais robusto e com mais conjuntos de dados de exemplo resultará em agentes ainda mais capazes, como foi o caso dos agentes de classificação. Importa destacar que, embora os resultados não tenham sido necessariamente negativos, foram amplamente influenciados pelos problemas associados ao submódulo de limpeza de dados. Não obstante, o submódulo implementado é capaz de analisar as características dos dados e seleccionar as operações mais adequadas aos dados em específico.

No geral, o módulo de AutoML satisfaz os objectivos principais de AutoML: retirar a complexidade e diminuir o tempo empregue. A grande diferença, que por sua vez é uma vantagem, do módulo para outras ferramentas de AutoML é que o módulo implementado automatiza as fases mais importantes e críticas de um projecto de Aprendizagem Computacional, em vez de apenas automatizar a fase da selecção do modelo. Desta maneira, permite que indivíduos não especializados possam tirar benefícios de Aprendizagem Computacional e implementar um projecto inteiro sem que seja necessário o conhecimento desta área.

Analisando os contributos do assistente virtual, constatamos que, embora seja apenas uma prova de conceito, é uma ferramenta útil e reduz mais uma barreira associada com projectos de Aprendizagem Computacional. Apesar das dificuldades descritas e da mudança de plano, foi possível implementar um assistente que possibilita a execução de operações de Aprendizagem Computacional com recurso à comunicação em linguagem natural. Esta implementação prova que existe espaço para que técnicas de AutoML sejam aplicadas em diferentes cenários.

Resumindo, este projecto, com a sua forte componente de investigação, teve em vista avaliar de que forma AutoML podia reduzir dois entraves na implementação de projectos de Aprendizagem Computacional: a necessidade de conhecimento profundo acerca da área e a necessidade de experiência de programação. Ambos os obstáculos foram, de certa forma, ultrapassados e dominados com a implementação do módulo de AutoML e a integração deste módulo num assistente virtual, respectivamente.

Trabalho Futuro

Sendo AutoML uma área bastante recente e complexa a nível prático de implementação, é natural que existam aspectos que possam ser melhorados. No decorrer do desenvolvimento das ferramentas e dos testes das mesmas, foram notadas algumas fragilidades associadas com as abordagens escolhidas. Neste sentido, o trabalho futuro passa maioritariamente por solucionar os problemas abordados relativos ao módulo AutoML assim como integrar

o módulo no assistente virtual.

No caso do submódulo de limpeza de dados, o trabalho futuro deve ter em vista o aperfeiçoamento da função de avaliação. Esta, é de extrema importância, visto que, influência directamente a escolha da melhor combinação de operações de limpeza de dados que, por sua vez, influência a qualidade da limpeza. A solução passaria por escolher outra forma de avaliar as combinações ou ajustar a abordagem já implementada de forma a corrigir os erros da mesma. A segunda hipótese pode ser a mais adequada visto que a função de avaliação mostrou-se eficaz apesar favorecer conjuntos de dados com mais atributos. Implementado uma das soluções apresentadas no Capítulo 5 seria o suficiente para que conjuntos de dados com muitos atributos não tivessem uma vantagem sobre conjuntos de dados com menos atributos e potencialmente melhores.

Em relação ao submódulo de engenharia de atributos, a abordagem seleccionada mostrou-se viável contudo encontrámos as dificuldades conhecidas da técnica escolhida. Estas dificuldades, que são características de *Deep Q Learning*, já foram estudadas por diversos investigadores que apresentaram soluções para as mesmas. Deste modo, o trabalho futuro relativamente a este submódulo passa por implementar uma dessas soluções, em concreto a utilização de uma segunda rede neuronal na fase de treino.
Também seria interessante treinar os agentes com mais conjuntos de dados de exemplo, algo que não foi possível devido ao tempo que demora treinar um agente com muitos conjuntos de dados. Esta solução trivial resultaria em mais robustez no que toca aos agentes.

A estrutura módulo AutoML fornece uma base concreta de automatização das diferentes fases, permitindo também a inclusão de mais operações de forma trivial. Assim sendo, algo útil seria a introdução de mais operações, quer seja de limpeza de dados ou de engenharia de atributos. Por exemplo, em relação a limpeza de dados, existem diversos tipos de erros que foram deixados de fora e a sua inclusão neste submódulo tornaria o mesmo mais geral e preparado para responder às inúmeras necessidades de diferentes conjuntos de dados.

O trabalho futuro deverá ainda passar por cumprir um dos objectivos inicias deste estágio: a inclusão do módulo AutoML no sistema Lexia. O assistente virtual implementado permitiu perceber que um assistente deste tipo é benéfico para diversos contextos, sendo por isso uma prova de conceito positiva. Neste sentido, o próximo passo será integrar o módulo no Lexia que é um sistema mais capaz e robusto que a prova de conceito implementada.

Finalizando, as ferramentas desenvolvidas são já uma base de trabalho interessante, no entanto, sendo AutoML uma área bastante recente e em constante evolução, existirá sempre espaço para melhorias.

Esta página foi intencionalmente deixada em branco.

Referências

[1] Pedro Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, New York, 2015.

[2] Radwa El Shawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *CoRR*, abs/1906.02287, 2019.

[3] Quanming Yao, Mengshuo Wang, Hugo Jair Escalante, Isabelle Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *CoRR*, abs/1810.13306, 2018.

[4] Xin He and Xiaowen Chu Kaiyong Zhao. Automl: A survey of the state-of-the-art. *CoRR*, 2019.

[5] Cognilytica Research. Data engineering, preparation, and labeling for ai 2019 [cgr-de100], 2019. Last accessed 16 September 2017.

[6] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. Activeclean: An interactive data cleaning framework for modern machine learning. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2117–2120, New York, NY, USA, 2016. ACM.

[7] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1247–1261, New York, NY, USA, 2015. ACM.

[8] Udayan Khurana, Horst Samulowitz, and Deepak S. Turaga. Feature engineering for predictive modeling using reinforcement learning. *CoRR*, abs/1709.07150, 2017.

[9] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *ICDM*, pages 979–984. IEEE Computer Society, 2016.

[10] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B. Khalil, and Deepak Turaga. Learning feature engineering for classification. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 2529–2535. AAAI Press, 2017.

[11] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. Autolearn—automated feature generation and selection. In *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 217–226. IEEE, 2017.

[12] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[13] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[14] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[16] Jason Zhi Liang, Elliot Meyerson, Babak Hodjat, Daniel Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary neural automl for deep learning. *CoRR*, abs/1902.06827, 2019.

[17] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[18] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

[19] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[21] Ken Schwaber. *Agile Project Management with Scrum.* Microsoft Press, Redmond, WA, 2004.

Apêndices

Esta página foi intencionalmente deixada em branco.

.1 Apêndice A: Conjuntos de dados utilizados

Nome	**Fonte**
Adult	https://www.openml.org/d/179
Airlines	https://www.openml.org/d/1169
Car	https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
CMC	https://www.openml.org/d/23
CRX	https://archive.ics.uci.edu/ml/datasets/Credit+Approval
Magic04	http://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope
Sat	https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)
Baseball	https://www.openml.org/d/41021
Bike Sharing	http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset/
Houses	https://www.openml.org/d/537
Kin8	https://www.openml.org/d/189
Space	https://www.openml.org/d/507
Water	https://www.openml.org/d/565
Wind	https://www.openml.org/d/503

Esta página foi intencionalmente deixada em branco.

.2 Apêndice B: Resultados Classificação

Dados	**Classificador**	**Accuracy Teste 1**	**F1 Score Teste 1**	**Accuracy Teste 2**	**F1 Score Teste 2**
Adult	SVC	0,8457	0,8377	0,845	0,8366
	Random Forest	0,8316	0,8294	0,8293	0,8255
	TPOT	0,852	0,8472	0,8525	0,8474
Airlines	SVC	0,648	0,6124	0,6559	0,6229
	Random Forest	0,6881	0,6786	0,692	0,6821
	TPOT	0,7005	0,6811	0,7051	0,6882
Car	SVC	0,8994	0,8886	0,8508	0,8273
	Random Forest	0,9852	0,9852	0,9795	0,9787
	TPOT	0,9792	0,979	1	1
CMC	SVC	0,4827	0,4806	0,5349	0,5289
	Random Forest	0,5241	0,5255	0,5034	0,4973
	TPOT	0,4758	0,4782	0,5209	0,517
CRX	SVC	0,88	0,8798	0,8906	0,8906
	Random Forest	0,9279	0,9277	0,875	0,875
	TPOT	0,8959	0,8955	0,875	0,8746
Magic04	SVC	0,8682	0,8639	0,8696	0,8654
	Random Forest	0,8782	0,8761	0,884	0,8822
	TPOT	0,8759	0,874	0,878	0,8766
Sat	SVC	0,8985	0,8952	0,8771	0,8729
	Random Forest	0,9233	0,9209	0,8996	0,8966
	TPOT	0,9244	0,9225	0,8861	0,8878

Tabela 1: Resultados dos conjuntos de classificação originais

Dados	Classificador	Accuracy Teste 1	F1 Score Teste 1	Accuracy Teste 2	F1 Score Teste 2
Adult	SVC	0.8546	0.849	0,8509	0,8446
	Random Forest	0,8346	0,8327	0,8331	0,8297
	TPOT	0,8609	0,8564	0,8569	0,8519
Airlines	SVC	0,6732	0,6533	0,6814	0,6604
	Random Forest	0,6872	0,6786	0,6883	0,6793
	TPOT	0,703	0,6898	0,7045	0,6922
Car	SVC	0,945	0,9448	0,9421	0,9407
	Random Forest	0,9393	0,9393	0,9364	0,9352
	TPOT	0,9248	0,9252	0,945	0,9438
CMC	SVC	0,4813	0,4787	0,5389	0,5322
	Random Forest	0,5355	0,5362	0,4881	0,4811
	TPOT	0,5084	0,5158	0,5423	0,5405
CRX	SVC	0,884	0,8843	0,8671	0,8672
	Random Forest	0,9202	0,9201	0,875	0,875
	TPOT	0,9057	0,9046	0,8828	0,8828
Magic04	SVC	0,8683	0,8638	0,8689	0,8645
	Random Forest	0,8767	0,8743	0,886	0,8844
	TPOT	0,8772	0,8751	0,8818	0,8805
Sat	SVC	0,8697	0,8648	0,8589	0,8528
	Random Forest	0,9139	0,9095	0,8974	0,8934
	TPOT	0,9209	0,9176	0,8986	0,8971

Tabela 2: Resultados dos conjuntos de classificação depois da limpeza de dados

Dados	Classificador	Accuracy Teste 1	F1 Score Teste 1	Accuracy Teste 2	F1 Score Teste 2
Adult	SVC	0,8458	0.8359	0,883	0,8736
	Random Forest	0,8473	0,8422	0,8917	0,8863
	TPOT	0,8499	0,8457	0,8996	0,8939
Airlines	SVC	0,6718	0,6294	0,691	0,6597
	Random Forest	0,6718	0,6619	0,6962	0,6879
	TPOT	0,691	0,6769	0,7021	0,6894
Car	SVC	0,951	0,9509	0,9422	0,9909
	Random Forest	0,9481	0,9481	0,9365	0,9854
	TPOT	0,9568	0,9564	0,9451	0,9945
CMC	SVC	0,5198	0,515	0,4509	0,4577
	Random Forest	0,5774	0,579	0,5594	0,5566
	TPOT	0,5368	0,541	0,5526	0,5502
CRX	SVC	0,8977	0,8987	0,8732	0,8731
	Random Forest	0,9122	0,9127	0,8732	0,8731
	TPOT	0,9194	0,9193	0,8732	0,8731
Magic04	SVC	0,8915	0,8885	0,8969	0,8945
	Random Forest	0,9044	0,903	0,9074	0,9063
	TPOT	0,9034	0,9019	0,9066	0,9053
Sat	SVC	0,8853	0,8715	0,8958	0,8907
	Random Forest	0,9201	0,915	0,9121	0,9092
	TPOT	0,9097	0,9038	0,9167	0,9138

Tabela 3: Resultados dos conjuntos de classificação depois da tarefa de engenharia de atributos

Esta página foi intencionalmente deixada em branco.

.3 Apêndice C: Resultados Regressão

Dados	Regressor	MAE Teste 1	MAPE Teste 1	MAE Teste 2	MAPE Teste 2
Baseball	SVR	60,99	7,44	70,66	8,27
	Random Forest	21,05	2,71	17,82	2,12
	TPOT	14	1,75	19,66	2,36
Bike Sharing	SVR	1500,92	63,92	1606,38	66,07
	Random Forest	81,08	2,54	60,4	1,78
	TPOT	0,01	0,01	0,01	0,01
Houses	SVR	88574,95	52,48	88262,34	51,49
	Random Forest	32276,48	17,28	31721,69	16,92
	TPOT	33306,49	17,73	32779,07	17,45
Kin8	SVR	0,07	11,63	0,07	11,34
	Random Forest	0,11	20,63	0,11	20,42
	TPOT	0,09	15,46	0,09	14,47
Space	SVR	0,09	16,33	0,09	16,37
	Random Forest	0,08	15,74	0,09	15,87
	TPOT	0,07	12,88	0,08	15,12
Water	SVR	0,66	0,7	0,28	0,28
	Random Forest	0,32	0,34	0,07	0,07
	TPOT	0,24	0,26	0,18	0,18
Wind	SVR	2,38	19,18	2,42	19,17
	Random Forest	2,4	19,72	2,47	19,79
	TPOT	2,38	19,3	2,38	18,92

Tabela 4: Resultados dos conjuntos de regressão originais

Dados	**Regressor**	**MAE Teste 1**	**MAPE Teste 1**	**MAE Teste 2**	**MAPE Teste 2**
Baseball	SVC	55,85	8,03	62,44	8,68
	Random Forest	17,8	2,57	19,08	2,68
	TPOT	16,32	2,34	17,44	2,44
Bike Sharing	SVC	1441,69	59,54	1583,17	64,8
	Random Forest	74,09	2,13	62,86	1,73
	TPOT	0	0	0	0
Houses	SVC	88574,95	52,48	88262,33	51,48
	Random Forest	32276,47	17,27	31721,68	16,91
	TPOT	32095,03	16,84	31098,98	16,41
Kin8	SVC	0,06	11,6	0,06	11,33
	Random Forest	0,1	20,62	0,1	20,41
	TPOT	0,08	15,45	0,08	14,46
Space	SVC	0,08	16,2	0,08	15,99
	Random Forest	0,07	15,91	0,07	15,35
	TPOT	0,06	14,26	0,07	13,88
Water	SVC	0,62	0,66	0,38	0,38
	Random Forest	0,39	0,41	0,22	0,22
	TPOT	0,48	0,5	0,18	0,18
Wind	SVC	2,36	19,14	2,41	19,16
	Random Forest	2,38	19,65	2,46	19,78
	TPOT	2,36	19,05	2,37	18,8

Tabela 5: Resultados dos conjuntos de regressão depois da limpeza de dados

Dados	**Regressor**	**MAE Teste 1**	**MAPE Teste 1**	**MAE Teste 2**	**MAPE Teste 2**
Baseball	SVC	30,05	4,3	34,14	4,71
	Random Forest	18,31	2,63	18,19	2,56
	TPOT	16,89	2,42	17,59	2,45
Bike Sharing	SVC	1495,85	61,68	1550,02	63,16
	Random Forest	70,3	1,85	43,38	1,29
	TPOT	22,95	0,75	32,71	1,15
Houses	SVC	88601,04	51,79	88357,62	48,98
	Random Forest	34707,21	19,25	35772,24	19,75
	TPOT	35253,25	19,14	35726,34	19,68
Kin8	SVC	0,06	11,34	0,07	13,6
	Random Forest	0,08	16,88	0,08	15,77
	TPOT	0,09	17,87	0,08	16,42
Space	SVC	0,07	16,01	0,07	15,76
	Random Forest	0,06	14,07	0,07	14,33
	TPOT	0,06	13,67	0,07	14,37
Water	SVC	0,49	0,52	0,22	0,22
	Random Forest	0,2	0,21	0,11	0,11
	TPOT	0,16	0,16	0,07	0,07
Wind	SVC	2,48	19,9	2,44	19,38
	Random Forest	2,36	19,58	2,4	19,37
	TPOT	2,32	18,83	2,4	18,92

Tabela 6: Resultados dos conjuntos de regressão depois da tarefa de engenharia de atributos